Problem A. Snapper Chain

Problem

The *Snapper* is a clever little device that, on one side, plugs its input plug into an output socket, and, on the other side, exposes an output socket for plugging in a light or other device.

When a *Snapper* is in the ON state and is receiving power from its input plug, then the device connected to its output socket is receiving power as well. When you snap your fingers -- making a clicking sound -- any *Snapper* receiving power at the time of the snap toggles between the ON and OFF states.

In hopes of destroying the universe by means of a singularity, I have purchased **N** *Snapper* devices and chained them together by plugging the first one into a power socket, the second one into the first one, and so on. The light is plugged into the **N**th *Snapper*.

Initially, all the *Snapper*s are in the OFF state, so only the first one is receiving power from the socket, and the light is off. I snap my fingers once, which toggles the first *Snapper* into the ON state and gives power to the second one. I snap my fingers again, which toggles both *Snapper*s and then promptly cuts power off from the second one, leaving it in the ON state, but with no power. I snap my fingers the third time, which toggles the first *Snapper* again and gives power to the second one. Now both *Snapper*s are in the ON state, and if my light is plugged into the second *Snapper* it will be *on*.

I keep doing this for hours. Will the light be *on* or *off* after I have snapped my fingers **K** times? The light is *on* if and only if it's receiving power from the *Snapper* it's plugged into.

Input

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

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is either "ON" or "OFF", indicating the state of the light bulb.

Limits

$1 \le \mathbf{T} \le 10{,}000$.

Small dataset

$1 \le \mathbf{N} \le 10$;
$0 \le \mathbf{K} \le 100$;

Large dataset

$1 \le \mathbf{N} \le 30$;
$0 \le \mathbf{K} \le 10^8$;

Sample

Input	Output
4	Case #1: OFF
1 0	Case #2: ON
1 1	Case #3: OFF
4 0	Case #4: ON
4 47	

Problem B. **Fair Warning**

Problem

On our planet, Jamcode IX, three Great Events occurred. They happened 26000, 11000 and 6000 slarboseconds ago. In 4000 slarboseconds, the amount of time since all of those events will be multiples of 5000 slarboseconds, the largest possible amount... and the apocalypse will come.

Luckily for you, you live on Jamcode X! The apocalypse came on Jamcode IX less than a year ago. But Jamcode X has a worrying prophecy: "After the moment of reckoning, on the first *optimum anniversary* of the **N** Great Events, the apocalypse will come. 64 bits will not save you. You have been warned."

The people of Jamcode X are very concerned by this prophecy. All of the Great Events have already happened, and their times have been measured to the nearest slarbosecond; but nobody knows when their *optimum anniversary* will occur. After studying the diary of a scientist from Jamcode IX, scientists working on the problem have come up with a theory:

The *moment of reckoning* is now, the moment you solve this problem. At some time $\mathbf{y} \geq \mathbf{0}$ slarboseconds from now, the number of slarboseconds since each of the Great Events will be divisible by some maximum number **T**. If you can find the smallest value of **y** that gives this largest possible **T**, that will give you the *optimum anniversary* when the apocalypse will come.

On Jamcode IX, for example, there were 3 Great Events and they happened 26000, 11000 and 6000 slarboseconds before the moment of reckoning. 4000 slarboseconds later, the amount of time since each event was a multiple of T=5000 slarboseconds, and the apocalypse came.

Your job is to compute the amount of time until the apocalypse comes. But remember the prophecy: even though the people of Jamcode X have been solving problems for two years, and 64-bit integers have always been enough, they might not always be enough now or in the future.

Input

The first line of the input gives the number of test cases, **C**. **C** lines follow. Each starts with a single integer **N**, which is followed by a space and then **N** space-separated integers $\mathbf{t_i}$, the number of slarboseconds since Great Event **i** occurred.

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 minimum number of slarboseconds until $\mathbf{t_i} + \mathbf{y}$ is a multiple of the largest possible integer factor **T** for all **i**.

Limits

$1 \leq \mathbf{C} \leq 100$.
$\mathbf{t_i} \neq \mathbf{t_j}$ for some **i**, **j**.

Small dataset

$2 \leq \mathbf{N} \leq 3$.
$1 \leq \mathbf{t_i} \leq 10^8$.

Large dataset

$2 \leq \mathbf{N} \leq 1000$.
$1 \leq \mathbf{t_i} \leq 10^{50}$.

Sample

Input

```
3
3 26000000 11000000 6000000
3 1 10 11
2 800000000000000000001 900000000000000000001
```

Output

```
Case #1: 4000000
Case #2: 0
Case #3: 99999999999999999999
```

Epilogue

Fortunately for the peoples of the Jamcode system, "the apocalypse" turned out to be a mistranslation of "the giant party." Nobody from Jamcode IX bothered to pass this along, because they were having so much fun.

Submissions

Snapper Chain

10pt Not attempted **9461/11212 users** correct (84%)

23pt Not attempted **7957/9406 users** correct (85%)

Fair Warning

10pt Not attempted **3312/4340 users** correct (76%)

23pt Not attempted **2469/3001 users** correct (82%)

Theme Park

10pt Not attempted **8033/8501 users** correct (94%)

23pt Not attempted **3050/7644 users** correct (40%)

Top Scores

neal.wu	99
LayCurse	99
eireksten	99
agus.mw	99
lympanda	99
pmnox	99
levlam	99
ZhukovDmitry	99
kmod	99
stubbscroll	99

Problem C. Theme Park

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 10 points — Solve C-small

Large input 23 points — Solve C-large

Problem

Roller coasters are so much fun! It seems like everybody who visits the theme park wants to ride the roller coaster. Some people go alone; other people go in groups, and don't want to board the roller coaster unless they can all go together. And *everyone* who rides the roller coaster wants to ride again. A ride costs 1 Euro per person; your job is to figure out how much money the roller coaster will make today.

The roller coaster can hold **k** people at once. People queue for it in groups. Groups board the roller coaster, one at a time, until there are no more groups left or there is no room for the next group; then the roller coaster goes, whether it's full or not. Once the ride is over, all of its passengers re-queue in the same order. The roller coaster will run **R** times in a day.

For example, suppose **R**=4, **k**=6, and there are four groups of people with sizes: 1, 4, 2, 1. The first time the roller coaster goes, the first two groups [1, 4] will ride, leaving an empty seat (the group of 2 won't fit, and the group of 1 can't go ahead of them). Then they'll go to the back of the queue, which now looks like 2, 1, 1, 4. The second time, the coaster will hold 4 people: [2, 1, 1]. Now the queue looks like 4, 2, 1, 1. The third time, it will hold 6 people: [4, 2]. Now the queue looks like [1, 1, 4, 2]. Finally, it will hold 6 people: [1, 1, 4]. The roller coaster has made a total of 21 Euros!

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, with each test case consisting of two lines. The first line contains three space-separated integers: **R**, **k** and **N**. The second line contains **N** space-separated integers $\mathbf{g_i}$, each of which is the size of a group that wants to ride. $\mathbf{g_0}$ is the size of the first group, $\mathbf{g_1}$ is the size of the second group, etc.

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 number of Euros made by the roller coaster.

Limits

$1 \le \mathbf{T} \le 50$.
$\mathbf{g_i} \le \mathbf{k}$.

Small dataset

$1 \le \mathbf{R} \le 1000$.
$1 \le \mathbf{k} \le 100$.
$1 \le \mathbf{N} \le 10$.
$1 \le \mathbf{g_i} \le 10$.

Large dataset

$1 \le \mathbf{R} \le 10^8$.
$1 \le \mathbf{k} \le 10^9$.
$1 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{g_i} \le 10^7$.

Sample

Input

```
3
4 6 4
1 4 2 1
100 10 1
1
5 5 10
2 4 2 3 4 2 1 2 1 3
```

Output

```
Case #1: 21
Case #2: 100
Case #3: 20
```

Problem A. Rotate

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 11 points — Solve A-small

Large input 12 points — Solve A-large

Problem

In the exciting game of Join-**K**, red and blue pieces are dropped into an **N**-by-**N** table. The table stands up vertically so that pieces drop down to the bottom-most empty slots in their column. For example, consider the following two configurations:

```
- Legal Position -

       .......
       .......
       .......
       ....R..
       ...RB..
       ..BRB..
       .RBBR..
```

```
- Illegal Position -

         .......
         .......
         .......
         .......
Bad  ->  ..BR...
         ...R...
         .RBBR..
```

In these pictures, each '.' represents an empty slot, each 'R' represents a slot filled with a red piece, and each 'B' represents a slot filled with a blue piece. The left configuration is legal, but the right one is not. This is because one of the pieces in the third column (marked with the arrow) has not fallen down to the empty slot below it.

A player wins if they can place at least **K** pieces of their color in a row, either horizontally, vertically, or diagonally. The four possible orientations are shown below:

```
- Four in a row -

R   RRRR    R   R
R          R     R
R         R       R
R        R         R
```

In the "Legal Position" diagram at the beginning of the problem statement, both players had lined up two pieces in a row, but not three.

As it turns out, you are right now playing a very exciting game of Join-**K**, and you have a tricky plan to ensure victory! When your opponent is not looking, you are going to rotate the board 90 degrees clockwise onto its side. Gravity will then cause the pieces to fall down into a new position as shown below:

```
- Start -      - Rotate -     - Gravity -

 .......        .......        .......
 .......        R......        .......
 .......        BB.....        .......
 ...R...        BRRR...        R......
 ...RB..        RBB....        BB.....
 ..BRB..        .......        BRR....
 .RBBR..        .......        RBBR...
```

Unfortunately, you only have time to rotate once before your opponent will notice.

All that remains is picking the right time to make your move. Given a board position, you should determine which player (or players!) will have **K** pieces in a row after you rotate the board clockwise and gravity takes effect in the new direction.

Notes

- You can rotate the board only once.
- Assume that gravity only takes effect after the board has been rotated completely.
- Only check for winners after gravity has finished taking effect.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, each beginning with a line containing the integers **N** and **K**. The next **N** lines will each be exactly **N** characters long, showing the initial position of the board, using the same format as the diagrams above.

The initial position in each test case will be a legal position that can occur during a game of Join-**K**. In particular, neither player will have already formed **K** pieces in a row.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1), and y is one of "Red", "Blue", "Neither", or "Both". Here, y indicates which player or players will have **K** pieces in a row after you rotate the board.

Limits

1 ≤ **T** ≤ 100.
3 ≤ **K** ≤ **N**.

Small dataset

3 ≤ **N** ≤ 7.

Large dataset

3 ≤ **N** ≤ 50.

Sample

Input

```
4
7 3
.......
.......
.......
...R...
...BB..
..BRB..
.RRBR..
6 4
......
......
.R...R
.R..BB
.R.RBR
RB.BBB
4 4
R...
BR..
BR..
BR..
3 3
B..
RB.
RB.
```

Output

```
Case #1: Neither
Case #2: Both
Case #3: Red
Case #4: Blue
```

Problem B. **Make it Smooth**

Problem

You have a one-dimensional array of **N** pixels. Each pixel has a value, represented by a number between 0 and 255, inclusive. The *distance* between two pixels is the absolute difference of their numbers.

You can perform each of the following operations zero or more times:

1. With cost **D**, delete any pixel, so its original neighbors become neighboring pixels.
2. With cost **I**, insert one pixel of any value into any position -- either between two existing pixels, or before the first pixel, or after the last pixel.
3. You can change the value of any pixel. The cost is the absolute difference of the old value of the pixel and the new value of the pixel.

The array is *smooth* if any neighboring pixels have distance at most **M**. Find the minimum possible cost of a sequence of operations that makes the array smooth.

Note: The empty array -- the array containing no pixels -- is considered to be smooth.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, each with two lines. The first line is in the form "**D I M N**", the next line contains **N** numbers $\mathbf{a_i}$: the values of the pixels from left to right.

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 minimum cost to make the input array smooth.

Limits

All the numbers in the input are integers.
$1 \le \mathbf{T} \le 100$
$0 \le \mathbf{D}, \mathbf{I}, \mathbf{M}, \mathbf{a_i} \le 255$

Small dataset

$1 \le \mathbf{N} \le 3$.

Large dataset

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

Sample

Input	Output
2 6 6 2 3 1 7 5 100 1 5 3 1 50 7	Case #1: 4 Case #2: 17

Explanation

In Case #1, decreasing the 7 to 3 costs 4 and is the cheapest solution. In Case #2, deleting is extremely expensive; it's cheaper to insert elements so your final array looks like `[1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 50, 45, 40, 35, 30, 25, 20, 15, 10, 7]`.

Submissions

Rotate

11pt	Not attempted **2076/2436 users** correct (85%)
12pt	Not attempted **1855/2071 users** correct (90%)

Make it Smooth

12pt	Not attempted **509/954 users** correct (53%)
24pt	Not attempted **319/482 users** correct (66%)

Number Game

16pt	Not attempted **680/1091 users** correct (62%)
25pt	Not attempted **244/450 users** correct (54%)

Top Scores

rng..58	100
peter50216	100
cgy4ever	100
rem	100
XiaoZiqian	100
qizichao	100
exod40	100
GarnetCrow	100
hos.lyric	100
ACRush	100

Problem C. **Number Game**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 16 points	Solve C-small
Large input 25 points	Solve C-large

Problem

Arya and Bran are playing a game. Initially, two positive integers **A** and **B** are written on a blackboard. The players take turns, starting with Arya. On his or her turn, a player can replace **A** with **A** - **k*B** for any positive integer **k**, or replace **B** with **B** - **k*A** for any positive integer **k**. The first person to make one of the numbers drop to zero or below loses.

For example, if the numbers are initially (12, 51), the game might progress as follows:

- Arya replaces 51 with 51 - 3*12 = 15, leaving (12, 15) on the blackboard.
- Bran replaces 15 with 15 - 1*12 = 3, leaving (12, 3) on the blackboard.
- Arya replaces 12 with 12 - 3*3 = 3, leaving (3, 3) on the blackboard.
- Bran replaces one 3 with 3 - 1*3 = 0, and loses.

We will say (**A**, **B**) is a *winning* position if Arya can always win a game that starts with (**A**, **B**) on the blackboard, no matter what Bran does.

Given four integers $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{B_1}$, $\mathbf{B_2}$, count how many winning positions (**A**, **B**) there are with $\mathbf{A_1} \le \mathbf{A} \le \mathbf{A_2}$ and $\mathbf{B_1} \le \mathbf{B} \le \mathbf{B_2}$.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow, one per line. Each line contains the four integers $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{B_1}$, $\mathbf{B_2}$, separated by spaces.

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 number of winning positions (**A**, **B**) with $\mathbf{A_1} \le \mathbf{A} \le \mathbf{A_2}$ and $\mathbf{B_1} \le \mathbf{B} \le \mathbf{B_2}$.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{A_1} \le \mathbf{A_2} \le 1{,}000{,}000$.
$1 \le \mathbf{B_1} \le \mathbf{B_2} \le 1{,}000{,}000$.

Small dataset

$\mathbf{A_2} - \mathbf{A_1} \le 30$.
$\mathbf{B_2} - \mathbf{B_1} \le 30$.

Large dataset

$\mathbf{A_2} - \mathbf{A_1} \le 999{,}999$.
$\mathbf{B_2} - \mathbf{B_1} \le 999{,}999$.

No additional constraints.

Sample

Input	Output
<pre>3 5 5 8 8 11 11 2 2 1 6 1 6</pre>	<pre>Case #1: 0 Case #2: 1 Case #3: 20</pre>

Problem A. File Fix-it

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 12 points	Solve A-small
Large input 14 points	Solve A-large

Problem

On Unix computers, data is stored in *directories*. There is one *root directory*, and this might have several directories contained inside of it, each with different names. These directories might have even more directories contained inside of them, and so on.

A directory is uniquely identified by its name and its parent directory (the directory it is directly contained in). This is usually encoded in a *path*, which consists of several parts each preceded by a forward slash ('/'). The final part is the name of the directory, and everything else gives the path of its parent directory. For example, consider the path:

```
/home/gcj/finals
```

This refers to the directory with name "finals" in the directory described by "/home/gcj", which in turn refers to the directory with name "gcj" in the directory described by the path "/home". In this path, there is only one part, which means it refers to the directory with the name "home" in the root directory.

To create a directory, you can use the *mkdir* command. You specify a path, and then *mkdir* will create the directory described by that path, but *only if* the parent directory already exists. For example, if you wanted to create the "/home/gcj/finals" and "/home/gcj/quals" directories from scratch, you would need four commands:

```
mkdir /home
mkdir /home/gcj
mkdir /home/gcj/finals
mkdir /home/gcj/quals
```

Given the full set of directories already existing on your computer, and a set of new directories you want to create if they do not already exist, how many *mkdir* commands do you need to use?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with a line containing two integers **N** and **M**, separated by a space.

The next **N** lines each give the path of one directory that already exists on your computer. This list will include every directory already on your computer other than the root directory. (The root directory is on every computer, so there is no need to list it explicitly.)

The next **M** lines each give the path of one directory that you want to create.

Each of the paths in the input is formatted as in the problem statement above. Specifically, a path consists of one or more lower-case alpha-numeric strings (i.e., strings containing only the symbols 'a'-'z' and '0'-'9'), each preceded by a single forward slash. These alpha-numeric strings are never empty.

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 number of *mkdir* you need.

Limits

$1 \le \mathbf{T} \le 100$.
No path will have more than 100 characters in it.
No path will appear twice in the list of directories already on your computer, or in the list of directories you wish to create. A path may appear once in both lists however. (See example case #2 below.)
If a directory is listed as being on your computer, then its parent directory will also be listed, unless the parent is the root directory.
The input file will be no longer than 100,000 bytes in total.

Small dataset

0 ≤ **N** ≤ 10.
1 ≤ **M** ≤ 10.

Large dataset

0 ≤ **N** ≤ 100.
1 ≤ **M** ≤ 100.

Sample

Input	Output
<pre>3 0 2 /home/gcj/finals /home/gcj/quals 2 1 /chicken /chicken/egg /chicken 1 3 /a /a/b /a/c /b/b</pre>	<pre>Case #1: 4 Case #2: 0 Case #3: 4</pre>

Problem B. Picking Up Chicks

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 13 points	Solve B-small
Large input 17 points	Solve B-large

Problem

A flock of chickens are running east along a straight, narrow road. Each one is running with its own constant speed. Whenever a chick catches up to the one in front of it, it has to slow down and follow at the speed of the other chick. You are in a mobile crane behind the flock, chasing the chicks towards the barn at the end of the road. The arm of the crane allows you to pick up any chick momentarily, let the chick behind it pass underneath and place the picked up chick back down. This operation takes no time and can only be performed on a pair of chicks that are immediately next to each other, even if 3 or more chicks are in a row, one after the other.

Given the initial locations ($\mathbf{X_i}$) at time 0 and natural speeds ($\mathbf{V_i}$) of the chicks, as well as the location of the barn (**B**), what is the minimum number of swaps you need to perform with your crane in order to have at least **K** of the **N** chicks arrive at the barn no later than time **T**?

You may think of the chicks as points moving along a line. Even if 3 or more chicks are at the same location, next to each other, picking up one of them will only let one of the other two pass through. Any swap is instantaneous, which means that you may perform multiple swaps at the same time, but each one will count as a separate swap.

Input

The first line of the input gives the number of test cases, **C**. **C** test cases follow. Each test case starts with 4 integers on a line -- **N**, **K**, **B** and **T**. The next line contains the **N** different integers $\mathbf{X_i}$, in increasing order. The line after that contains the **N** integers $\mathbf{V_i}$. All distances are in meters; all speeds are in meters per second; all times are in seconds.

Output

For each test case, output one line containing "Case #x: **S**", where x is the case number (starting from 1) and **S** is the smallest number of required swaps, or the word "IMPOSSIBLE".

Limits

$1 \le \mathbf{C} \le 100$;
$1 \le \mathbf{B} \le 1{,}000{,}000{,}000$;
$1 \le \mathbf{T} \le 1{,}000$;
$0 \le \mathbf{X_i} < \mathbf{B}$;
$1 \le \mathbf{V_i} \le 100$;
All the $\mathbf{X_i}$'s will be distinct and in increasing order.

Small dataset

$1 \le \mathbf{N} \le 10$;
$0 \le \mathbf{K} \le \min(3, \mathbf{N})$;

Large dataset

$1 \le \mathbf{N} \le 50$;
$0 \le \mathbf{K} \le \mathbf{N}$;

Sample

Input	Output
3 5 3 10 5 0 2 5 6 7 1 1 1 1 4 5 3 10 5 0 2 3 5 7 2 1 1 1 4 5 3 10 5 0 2 3 4 7 2 1 1 1 4	Case #1: 0 Case #2: 2 Case #3: IMPOSSIBLE

Problem C. **Your Rank is Pure**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input
14 points
Solve C-small

Large input
30 points
Solve C-large

Problem

Pontius: You know, I like this number 127, I don't know why.
Woland: Well, that is an object so pure. You know the *prime numbers*.
Pontius: Surely I do. Those are the objects possessed by our ancient masters hundreds of years ago. Oh, yes, why then? 127 is indeed a prime number as I was told.
Woland: Not... only... that. 127 is the 31st prime number; then, 31 is itself a prime, it is the 11th; and 11 is the 5th; 5 is the 3rd; 3, you know, is the second; and finally 2 is the 1st.
Pontius: Heh, that is indeed... purely prime.

The game can be played on any subset S of positive integers. A number in S is considered pure with respect to S if, starting from it, you can continue taking its rank in S, and get a number that is also in S, until in finite steps you hit the number 1, which is not in S.

When **n** is given, in how many ways you can pick S, a subset of {2, 3, ..., n}, so that **n** is pure, with respect to S? The answer might be a big number, you need to output it modulo 100003.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains a single integer **n**.

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 answer as described above.

Limits

T ≤ 100.

Small dataset

2 ≤ **n** ≤ 25.

Large dataset

2 ≤ **n** ≤ 500.

Sample

Input	Output
2 5 6	Case #1: 5 Case #2: 8

Problem A. Rope Intranet

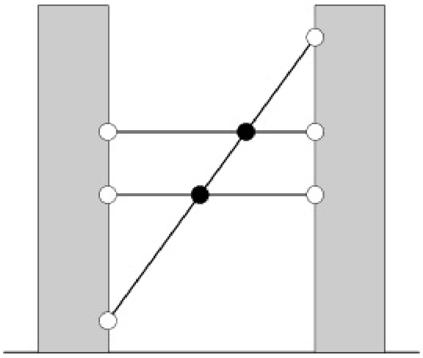
This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 9 points	Solve A-small
Large input 13 points	Solve A-large

Problem

A company is located in two very tall buildings. The company intranet connecting the buildings consists of many wires, each connecting a window on the first building to a window on the second building.

You are looking at those buildings from the side, so that one of the buildings is to the left and one is to the right. The windows on the left building are seen as points on its right wall, and the windows on the right building are seen as points on its left wall. Wires are straight segments connecting a window on the left building to a window on the right building.



You've noticed that no two wires share an endpoint (in other words, there's at most one wire going out of each window). However, from your viewpoint, some of the wires intersect midway. You've also noticed that exactly two wires meet at each intersection point.

On the above picture, the intersection points are the black circles, while the windows are the white circles.

How many intersection points do you see?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with a line containing an integer **N**, denoting the number of wires you see.

The next **N** lines each describe one wire with two integers $\mathbf{A_i}$ and $\mathbf{B_i}$. These describe the windows that this wire connects: $\mathbf{A_i}$ is the height of the window on the left building, and $\mathbf{B_i}$ is the height of the window on the right building.

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 number of intersection points you see.

Limits

$1 \le \mathbf{T} \le 15$.
$1 \le \mathbf{A_i} \le 10^4$.
$1 \le \mathbf{B_i} \le 10^4$.
Within each test case, all $\mathbf{A_i}$ are different.
Within each test case, all $\mathbf{B_i}$ are different.
No three wires intersect at the same point.

Small dataset

$1 \le \mathbf{N} \le 2$.

Large dataset

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

Sample

Input	Output
`` 2 3 1 10 5 5 7 7 2 1 1 2 2 ```	``` Case #1: 2 Case #2: 0 ```

Problem B. Load Testing

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 14 points	Solve B-small
Large input 22 points	Solve B-large

Problem

Now that you have won Code Jam and been hired by Google as a software engineer, you have been assigned to work on their wildly popular programming contest website.

Google is expecting a lot of participants (**P**) in Code Jam next year, and they want to make sure that the site can support that many people at the same time. During Code Jam 2010 you learned that the site could support at least **L** people at a time without any errors, but you also know that the site can't yet support **P** people.

To determine how many more machines you'll need, you want to know within a factor of **C** how many people the site can support. This means that there is an integer **a** such that you know the site can support **a** people, but you know the site can't support **a** * **C** people.

You can run a series of *load tests*, each of which will determine whether the site can support at least **X** people for some integer value of **X** that you choose. If you pick an optimal strategy, choosing what tests to run based on the results of previous tests, how many load tests do you need in the worst case?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow, each of which contains space-separated integers **L**, **P** and **C** in that order.

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 number of load tests you need to run in the worst case before knowing within a factor of **C** how many people the site can support.

Limits

1 ≤ **T** ≤ 1000.
2 ≤ **C** ≤ 10.
L, **P** and **C** are all integers.

Small dataset

1 ≤ **L** < **P** ≤ 10^3.

Large dataset

1 ≤ **L** < **P** ≤ 10^9.

Sample

Input	Output
4 50 700 2 19 57 3 1 1000 2 24 97 2	Case #1: 2 Case #2: 0 Case #3: 4 Case #4: 2

Explanation

In Case #2, we already know that the site can support between 19 and 57 people. Since those are a factor of 3 apart, we don't need to do any testing.

In Case #4, we can test 48; but if the site can support 48 people, we need more testing, because 48*2 < 97. We could test 49; but if the site can't support 49 people, we need more testing, because 24 * 2 < 49. So we need two tests.

Round 1C 2010

A. Rope Intranet
B. Load Testing
C. Making Chess Boards

Contest Analysis
Questions asked

Problem C. Making Chess Boards

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 18 points	Solve C-small
Large input 24 points	Solve C-large

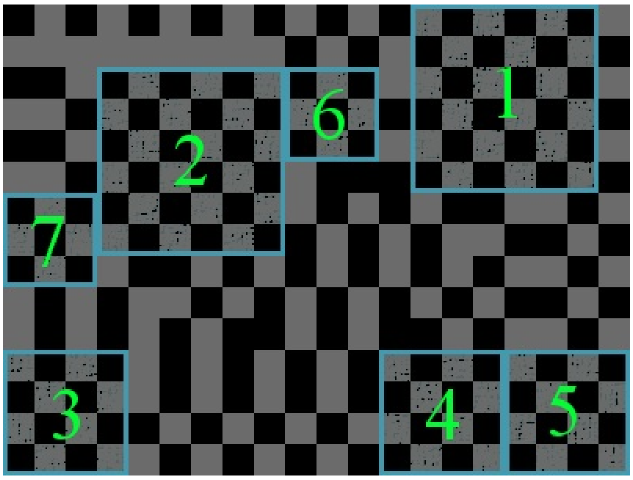
Problem

The chess board industry has fallen on hard times and needs your help. It is a little-known fact that chess boards are made from the bark of the extremely rare Croatian Chess Board tree, (*Biggus Mobydiccus*). The bark of that tree is stripped and unwrapped into a huge rectangular sheet of chess board material. The rectangle is a grid of black and white squares.

Your task is to make as many large square chess boards as possible. A chess board is a piece of the bark that is a square, with sides parallel to the sides of the bark rectangle, with cells colored in the pattern of a chess board (no two cells of the same color can share an edge).

Each time you cut out a chess board, you must choose the largest possible chess board left in the sheet. If there are several such boards, pick the topmost one. If there is still a tie, pick the leftmost one. Continue cutting out chess boards until there is no bark left. You may need to go as far as cutting out 1-by-1 mini chess boards.

Here is an example showing the bark of a Chess Board tree and the first few chess boards that will be cut out of it.



Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one starts with a line containing the dimensions of the bark grid, **M** and **N**. **N** will always be a multiple of 4. The next **M** lines will each contain an (**N**/4)-character hexadecimal integer, representing a row of the bark grid. The binary representation of these integers will give you a strings of **N** bits, one for each row. Zeros represent black squares; ones represent white squares of the grid. The rows are given in the input from top to bottom. In each row, the most-significant bit of the hexadecimal integer corresponds to the leftmost cell in that row.

Output

For each test case, output one line containing "Case #x: **K**", where x is the case number (starting from 1) and **K** is the number of different chess board sizes that you can cut out by following the procedure described above. The next **K** lines should contain two integers each -- the size of the chess board (from largest to smallest) and the number of chess boards of that size that you can cut out.

Limits

1 ≤ **T** ≤ 100;
N will be divisible by 4;
Each hexadecimal integer will contain exactly **N**/4 characters.
Only the characters 0-9 and A-F will be used.

Small dataset

1 ≤ **M** ≤ 32;
1 ≤ **N** ≤ 32.

Large dataset

1 ≤ **M** ≤ 512;
1 ≤ **N** ≤ 512;
The input file will be at most 200kB in size.

Sample

Input	Output
4 15 20 55555 FFAAA 2AAD5 D552A 2AAD5 D542A 4AD4D B52B2 52AAD AD552 AA52D AAAAA 5AA55 A55AA 5AA55 4 4 0 0 0 0 4 4 3 3 C C 4 4 6 9 9 6	Case #1: 5 6 2 4 3 3 7 2 15 1 57 Case #2: 1 1 16 Case #3: 2 2 1 1 12 Case #4: 1 2 4

The first example test case represents the image above.

Problem A. **Elegant Diamond**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 4 points — Solve A-small

Large input 8 points — Solve A-large

Problem

The king has hired you to make him an elegant diamond. An elegant diamond is a two-dimensional object made out of digits that's symmetric about a horizontal and a vertical axis. For example, the following four shapes are elegant diamonds:

```
  2       8       3      7
 3 3     8 8     2 2
4 1 4     8       3
 3 3
  2
```

These three shapes are diamonds, but are not elegant:

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

These three shapes are not diamonds:

```
 1     2      8   8
1 1   222       0
       2      00000
```

The king will start by giving you a diamond, which may not be elegant. Your job is to make it elegant by *enhancing* it, adding digits on to make a bigger diamond. Because you don't want to spend too much money, you want to do it with as little *cost* as possible.

Definitions

A **diamond of size *k*** is 2k-1 lines of digits, 0-9, separated by single spaces, organized in the following way:

- Line i (1 ≤ i ≤ k) contains k-i spaces, then i digits separated by single spaces.
- Line i (k < i < 2k) contains i-k spaces, then 2k-i digits separated by single spaces.

An **elegant diamond of size *k*** is a diamond of size k with the following two symmetry properties:

- Horizontal symmetry: Let c_i be the number of digits on line i. The j^{th} digit on line i (where j=1 for the first digit) must be the same as the c_i+1-j^{th} digit.
- Vertical symmetry: The j^{th} digit on line i (where i=1 for the first line) must be the same as the j^{th} digit on line 2k-i.

A diamond of size k can be **enhanced** by adding digits to it. The result of enhancing a diamond of size k has the following properties:

- The result is a diamond of size ≥ k.
- The original diamond is part of the result. In other words, there exist some X and some Y such that, for all values of i and j such that the j^{th} character of the i^{th} line of the original is a digit (as opposed to a space), the $j+X^{th}$ character on the $i+Y^{th}$ line of the result is also a digit and it's the same as the j^{th} character on the i^{th} line of the original.

The **cost** of enhancing a diamond is equal to the number of digits in the result of the enhancement, minus the number of digits in the original diamond.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of a single integer **k** in a line on its own, followed by a diamond of size **k**.

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 minimum cost required to enhance the given diamond into an elegant diamond. If the diamond is already elegant, y=0.

Limits

1 ≤ **T** ≤ 100.

Small dataset

1 ≤ **k** ≤ 10.

Large dataset

1 ≤ **k** ≤ 51.

Sample

Input	Output
<pre>4 1 0 2 1 2 2 1 2 1 1 2 1 3 1 6 3 9 5 5 6 3 1</pre>	<pre>Case #1: 0 Case #2: 0 Case #3: 5 Case #4: 7</pre>

Explanation

There are four cases. The first two cases start as elegant diamonds of size 1 and 2, respectively, and don't need to be enhanced; so the cost is 0. The third case can be enhanced to look like:

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

There are several possible enhancements, but this is one with the lowest possible cost, which is 5. In the fourth case we can enhance the diamond into the following elegant diamond:

```
   9
  1 1
 6 3 6
9 5 5 9
 6 3 6
  1 1
   9
```

...for a cost of 7.

Round 2 2010

A. Elegant Diamond
B. World Cup 2010
C. Bacteria
D. Grazing Google Goats

Contest Analysis

Questions asked 2

Submissions

Elegant Diamond

4pt Not attempted **540/1183 users** correct (46%)

8pt Not attempted **472/531 users** correct (89%)

World Cup 2010

10pt Not attempted **1456/1614 users** correct (90%)

15pt Not attempted **848/972 users** correct (87%)

Bacteria

6pt Not attempted **1655/1870 users** correct (89%)

25pt Not attempted **60/294 users** correct (20%)

Grazing Google Goats

7pt Not attempted **194/333 users** correct (58%)

25pt Not attempted **2/11 users** correct (18%)

Top Scores

bmerry	75
ZhukovDmitry	75
winger	75
stgatilov	75
Progbeat	75
pashka	75
halyavin	69
Zhuojie	68
wata	68
rng..58	68

Problem B. World Cup 2010

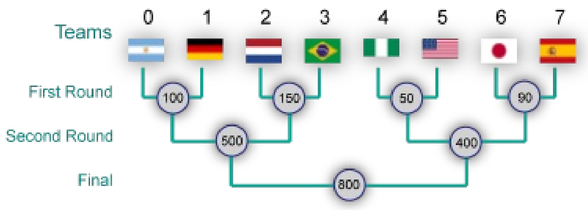
This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 10 points	Solve B-small
Large input 15 points	Solve B-large

Problem

After four years, it is the World Cup time again and Varva is on his way to South Africa, just in time to catch the second stage of the tournament.

In the second stage (also called the knockout stage), each match always has a winner; the winning team proceeds to the next round while the losing team is eliminated from the tournament. There are 2^P teams competing in this stage, identified with integers from 0 to 2^P - 1. The knockout stage consists of P rounds. In each round, each remaining team plays exactly one match. The exact pairs and the order of matches are determined by successively choosing two remaining teams with lowest identifiers and pairing them in a match. After all matches in one round are finished, the next round starts.



In order to help him decide which matches to see, Varva has compiled a list of constraints based on how much he likes a particular team. Specifically, for each team `i` he is **willing to miss at most** `M[i]` matches the team plays in the tournament.

Varva needs to buy a set of tickets that will guarantee that his preferences are satisfied, regardless of how the matches turn out. Other than that, he just wants to spend as little money as possible. Your goal is to find the **minimal amount of money** he needs to spend on the tickets.

Tickets for the matches need to be purchased in advance (before the tournament starts) and the ticket price for each match is known. Note that, in the small input, ticket prices for all matches will be equal, while in the large input, they may be different.

Example

A sample tournament schedule along with the ticket prices is given in the figure above. Suppose that the constraints are given by the array `M = {1, 2, 3, 2, 1, 0, 1, 3}`, the optimal strategy is as follows: Since we can't miss any games of team 5, we'll need to spend 50, 400, and 800 to buy tickets to all the matches team 5 may play in. Now, the constrains for the other teams are also satisfied by these tickets, except for team 0. The best option to fix this is to buy the ticket for team 0's first round match, spending another 100, bringing the total to 1350.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case starts with a line containing a single integer **P**. The next line contains 2^P integers -- the constraints `M[0]`, ..., `M[2`^{P}`-1]`.

The following block of **P** lines contains the ticket prices for all matches: the first line of the block contains 2^{P-1} integers -- ticket prices for first round matches, the second line of the block contains 2^{P-2} integers -- ticket prices for second round matches, etc. The last of the **P** lines contains a single integer -- ticket price for the final match of the World Cup. The prices are listed in the order the matches are played.

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 minimal amount of money Varva needs to spend on tickets as described above.

Limits

1 ≤ **T** ≤ 50
1 ≤ **P** ≤ 10
Each element of **M** is an integer between 0 and **P**, inclusive.

Small dataset

All the prices are equal to 1.

Large dataset

All the prices are integers between 0 and 100000, inclusive.

Sample

Input	Output
<pre>2 2 1 1 0 1 1 1 1 3 1 2 3 2 1 0 1 3 100 150 50 90 500 400 800</pre>	<pre>Case #1: 2 Case #2: 1350</pre>

Problem C. **Bacteria**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

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

Problem

A number of bacteria lie on an infinite grid of cells, each bacterium in its own cell.

Each second, the following transformations occur (all simultaneously):

1. If a bacterium has no neighbor to its north and no neighbor to its west, then it will die.
2. If a cell has no bacterium in it, but there are bacteria in the neighboring cells to the north and to the west, then a new bacterium will be born in that cell.

Upon examining the grid, you note that there are a positive, finite number of bacteria in one or more rectangular regions of cells.

Determine how many seconds will pass before all the bacteria die.

Here is an example of a grid that starts with 6 cells containing bacteria, and takes 6 seconds for all the bacteria to die. '1's represent cells with bacteria, and '0's represent cells without bacteria.

```
000010
011100
010000
010000
000000

000000
001110
011000
010000
000000

000000
000110
001100
011000
000000

000000
000010
000110
001100
000000

000000
000000
000010
000110
000000

000000
000000
000000
000010
000000

000000
000000
000000
000000
000000
```

Input

The input consists of:

- One line containing **C**, the number of test cases.

Then for each test case:

- One line containing **R**, the number of rectangles of cells that initially contain bacteria.
- **R** lines containing four space-separated integers $\mathbf{X_1}$ $\mathbf{Y_1}$ $\mathbf{X_2}$ $\mathbf{Y_2}$. This

indicates that all the cells with X coordinate between X_1 and X_2, inclusive, and Y coordinate between Y_1 and Y_2, inclusive, contain bacteria.

The rectangles may overlap.

North is in the direction of decreasing Y coordinate.
West is in the direction of decreasing X coordinate.

Output

For each test case, output one line containing "Case #N: T", where N is the case number (starting from 1), and T is the number of seconds until the bacteria all die.

Limits

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

Small dataset

$1 \le \mathbf{R} \le 10$
$1 \le \mathbf{X_1} \le \mathbf{X_2} \le 100$
$1 \le \mathbf{Y_1} \le \mathbf{Y_2} \le 100$

Large dataset

$1 \le \mathbf{R} \le 1000$
$1 \le \mathbf{X_1} \le \mathbf{X_2} \le 1000000$
$1 \le \mathbf{Y_1} \le \mathbf{Y_2} \le 1000000$

The number of cells initially containing bacteria will be at most 1000000.

Sample

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

Submissions

Elegant Diamond
4pt Not attempted **540/1183 users** correct (46%)
8pt Not attempted **472/531 users** correct (89%)

World Cup 2010
10pt Not attempted **1456/1614 users** correct (90%)
15pt Not attempted **848/972 users** correct (87%)

Bacteria
6pt Not attempted **1655/1870 users** correct (89%)
25pt Not attempted **60/294 users** correct (20%)

Grazing Google Goats
7pt Not attempted **194/333 users** correct (58%)
25pt Not attempted **2/11 users** correct (18%)

Top Scores

bmerry	75
ZhukovDmitry	75
winger	75
stgatilov	75
Progbeat	75
pashka	75
halyavin	69
Zhuojie	68
wata	68
rng..58	68

Problem D. Grazing Google Goats

Small input 7 points | Solve D-small

Large input 25 points | Solve D-large

Problem

Farmer John has recently acquired a nice herd of **N** goats for his field. Each goat i will be tied to a pole at some position $\mathbf{P}_i$ using a rope of length $\mathbf{L}_i$. This means that the goat will be able to travel anywhere in the field that is within distance $\mathbf{L}_i$ of the point $\mathbf{P}_i$, but nowhere else. (The field is large and flat, so you can think of it as an infinite two-dimensional plane.)

Farmer John already has the pole positions picked out from his last herd of goats, but he has to choose the rope lengths. There are two factors that make this decision tricky:

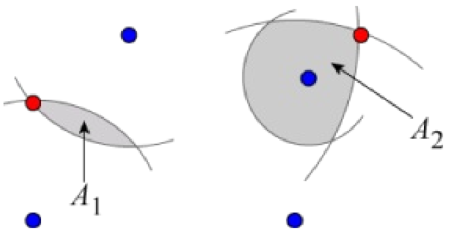
- The goats all need to be able to reach a single water bucket. Farmer John has not yet decided where to place this bucket. He has reduced the choice to a set of positions $\{\mathbf{Q}_1, \mathbf{Q}_2, ..., \mathbf{Q}_M\}$, but he is not sure which one to use.
- The goats are ill-tempered, and when they get together, they sometimes get in noisy fights. For everyone's peace of mind, Farmer John wants to minimize the area **A** that can be reached by all the goats.

Unfortunately, Farmer John is not very good at geometry, and he needs your help for this part!

For each bucket position $\mathbf{Q}_j$, you should choose rope lengths so as to minimize the area $\mathbf{A}_j$ that can be reached by every goat when the bucket is located at position $\mathbf{Q}_j$. You should then calculate each of these areas $\mathbf{A}_j$.

Example

In the picture below, there are four blue points, corresponding to the pole positions: $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{P}_4$. There are also two red points, corresponding to the potential bucket positions: $\mathbf{Q}_1$ and $\mathbf{Q}_2$. You need to calculate $\mathbf{A}_1$ and $\mathbf{A}_2$, the areas of the two shaded regions.



Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with a line containing the integers **N** and **M**.

The next **N** lines contain the positions $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_N$, one per line. This is followed by **M** lines, containing the positions $\mathbf{Q}_1$, $\mathbf{Q}_2$, ..., $\mathbf{Q}_M$, one per line.

Each of these **N** + **M** lines contains the corresponding position's **x** and **y** coordinates, separated by a single space.

Output

For each test case, output one line containing "Case #x: $\mathbf{A}_1$ $\mathbf{A}_2$ ... $\mathbf{A}_M$", where x is the case number (starting from 1), and $\mathbf{A}_1$ $\mathbf{A}_2$ ... $\mathbf{A}_M$ are the values defined above. Answers with a relative or absolute error of at most 10^{-6} will be considered correct.

Limits

All coordinates are integers between -10,000 and 10,000.
The positions $\mathbf{P}_1$, $\mathbf{P}_2$, ..., $\mathbf{P}_N$, $\mathbf{Q}_1$, $\mathbf{Q}_2$, ..., $\mathbf{Q}_M$ are all distinct and no three are

collinear.

Small dataset

1 ≤ **T** ≤ 100.
N = 2.
1 ≤ **M** ≤ 10.

Large dataset

1 ≤ **T** ≤ 10.
2 ≤ **N** ≤ 5,000.
1 ≤ **M** ≤ 1,000.

Sample (Small dataset)

Input	Output
`1` `2 3` `0 20` `20 0` `-20 10` `40 20` `0 19`	`Case #1: 1264.9865911 1713.2741229 0.2939440`

Sample (Large dataset)

Input	Output
`2` `4 2` `0 0` `100 100` `300 0` `380 90` `400 100` `1000 5` `3 1` `0 0` `10 10` `20 0` `10 5`	`Case #1: 1518.9063729 1193932.9692206` `Case #2: 0.0`

Submissions

De-RNG-ed

4pt Not attempted **273/325 users** correct (84%)

10pt Not attempted **179/231 users** correct (77%)

Fence

7pt Not attempted **250/299 users** correct (84%)

22pt Not attempted **77/177 users** correct (44%)

Hot Dog Proliferation

6pt Not attempted **217/249 users** correct (87%)

22pt Not attempted **20/95 users** correct (21%)

Different Sum

7pt Not attempted **102/125 users** correct (82%)

22pt Not attempted **23/47 users** correct (49%)

Top Scores

Burunduk1	100
winger	100
Eryx	100
RAVEman	78
Gennady.Korotkevich	78
nika	78
eatmore	78
pashka	78
Vasyl	78
jakubr	72

Problem A. De-RNG-ed

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 4 points	Solve A-small
Large input 10 points	Solve A-large

Problem

I want to make an online poker website. A very important component of such a system is the random number generator. It needs to be fast and random enough. Here is a compromise I came up with. I need a way to generate random numbers of length at most **D**. My plan is to select a prime number $\mathbf{P} \le 10^{\mathbf{D}}$. I am also going to pick non-negative integers **A** and **B**. Finally, I'm going to pick an integer seed **S** between 0 and **P**-1, inclusive.

To output my sequence of pseudo-random numbers, I'm going to first output **S** and then compute the new value of **S** like this:
`S := (A*S + B) mod P.`

Then I will output the new value of **S** as the next number in the sequence and update **S** again by using the same formula. I can repeat this as many times as I want.

Do you think that this is a good random number generator? Can you write a program that takes **K** consecutive elements of a sequence that was generated by my random number generator, and prints the next element of the sequence?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each one starts with a line containing **D** and **K**. The next line contains **K** consecutive elements generated by a random number generator of the kind described above.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is either the next number in the sequence, or the string "I don't know." if the answer is ambiguous.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{K} \le 10$.
The **K** integers will be consecutive elements of a sequence generated by a random number generator of the type described above.

Small dataset

$1 \le \mathbf{D} \le 4$.

Large dataset

$1 \le \mathbf{D} \le 6$.

Sample

Input	Output

Input
```
3
2 10
0 1 2 3 4 5 6 7 8 9
3 1
13
1 5
6 6 6 6
```

Output
```
Case #1: 10
Case #2: I don't know.
Case #3: 6
```

Problem B. Fence

Problem

We are looking into building a very long fence. We have already found a nice place to build it, and all that remains is to collect the materials.

From local hardware stores, we can buy unlimited numbers of wooden boards, each of which can come in a variety of different lengths. To avoid waste, we want to make sure that the total length of these boards is *exactly* equal to the length of the fence we are trying to build.

Given the length of the fence, and the possible board lengths that we can use, what is the minimum number of boards that we need to purchase in order to get exactly the right length?

Beware: the fence is going to be very long!

Input

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

Each test case consists of two lines. The first line contains space-separated integers **L** and **N**. These represent the total length of the fence, and the number of different board lengths that can be purchased. The second line contains **N** space-separated integers $\mathbf{B_1}$, $\mathbf{B_2}$, ..., $\mathbf{B_N}$, representing all the possible board lengths.

Output

For each test case, output one line containing "Case #x: M", where x is the case number (starting from 1) and M is as follows:

- If it is possible to purchase one or more boards so that their total length is exactly equal to **L**, then M should be the minimum number of boards required to do this.
- Otherwise, M should be the string "IMPOSSIBLE".

Limits

$1 \le \mathbf{T} \le 50$.
$10^{10} \le \mathbf{L} \le 10^{18}$.
$1 \le \mathbf{N} \le 100$.

Small dataset

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

Large dataset

$1 \le \mathbf{B_i} \le 100000$.

Sample

Input	Output
2 10000000001 3 23 51 100 10000000001 3 100 52 22	Case #1: 100000004 Case #2: IMPOSSIBLE

Explanation

In the first example, the optimal strategy is to use 2 boards of length 23, 5 boards of length 51, and 99999997 boards of length 100. Of course, you could use just 100000001 boards of length 100 to get a total *greater* than **L**, but that is not allowed.

In the second example, it is only possible to get even lengths.

Top Scores

Burunduk1	100
winger	100
Eryx	100
RAVEman	78
Gennady.Korotkevich	78
nika	78
eatmore	78
pashka	78
Vasyl	78
jakubr	72

Problem C. Hot Dog Proliferation

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input
6 points
Solve C-small

Large input
22 points
Solve C-large

Problem

A number of hot dog vendors have started selling hot dogs at corners (intersections) along a very long east-west street. The problem is that multiple vendors might be selling at the same corner, and then they will take each other's business. All is not lost though! The hot dog vendors have a plan.

If there are ever two or more vendors at the same corner, then exactly two of the vendors can perform a **move**, which means:

- One vendor moves one corner further to the east along the street.
- The other vendor moves one corner further to the west along the street.

Remember that the street is really long, so there is no danger of running out of corners. Given the starting positions of all hot dog vendors, you should find the minimum number of moves they need to perform before the vendors are all separated (meaning they are all on different corners).

For example, suppose the street begins with the following number of hot dog vendors on each corner, listed in order from west to east:

```
... 0 0 2 1 2 0 0 ...
```

Then the vendors can be separated in three moves, as shown below:

```
... 0 0 2 1 2 0 0 ...
        |
        +--- Do a move here

... 0 1 0 2 2 0 0 ...
          |
          +--- Do a move here

... 0 1 1 0 3 0 0 ...
            |
            +--- Do a move here

... 0 1 1 1 1 1 0 ...
```

Input

Each street corner is labeled with an integer, positive or negative. For each `i`, corner `i+1` refers to the next corner to the east from corner `i`. We will use this labeling system to describe corners in the input file.

The first line of the input file contains the number of cases, **T**. **T** test cases follow. Each case begins with the number of corners **C** that have at least one hot dog vendor in the starting configuration. The next **C** lines each contain a pair of space-separated integers **P**, **V**, indicating that there are **V** vendors at corner **P**.

Output

For each test case, output one line containing "Case #x: M", where x is the case number (starting from 1) and M is the minimum number of moves that need to be performed before the vendors all end up at different corners from each other.

Limits

$1 \le \mathbf{T} \le 50$.
$1 \le \mathbf{C} \le 200$.
All **P** values are in the range [-1000000, 1000000].
Within each test case, all **P** values are distinct and listed in increasing order.
All **V** values are positive integers. The limit on the sum of all **V** values is listed below.
It will always be possible to separate the hot dog vendors in a finite number of moves.

Small dataset

The total number of hot dog vendors in each test case is at most 200.

Large dataset

The total number of hot dog vendors in each test case is at most 100000.

Sample

Input	Output
2 3 -1 2 0 1 1 2 2 -1000 1 2000 1	Case #1: 3 Case #2: 0

Problem D. Different Sum

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 7 points	Solve D-small
Large input 22 points	Solve D-large

Problem

We have come up with a wonderful problem for Google Code Jam 2010 that involves contestants solving a cryptarithm. But we need your help in creating testcases for the problem; more precisely, we're concerned with addition equations that are good enough (in the sense defined below) for conversion into cryptarithms.

You don't need to know what a cryptarithm is to solve this problem, as we'll provide all required definitions. We define a *cryptarithm equation* to be an addition equation written in such a way that all summands (numbers being added) and the sum are aligned to the same right border like this:

```
124
 31
 25
---
180
```

Additionally, for each column of a cryptarithm equation, all digits of the summands in that column must be different. Note that we don't include the sum in this constraint. So for example in the above equation the first column contains only digit 1, the second column contains digits 2,3 and 2, and the third column contains digits 4, 1 and 5. This equation is not a cryptarithm equation since the second column contains two 2's. However, it would be a cryptarithm equation if we replaced the last summand with 15 (and the sum with 170).

Note that summands in a cryptarithm equation are always positive and written without leading zeros. The order of summands is not important (in other words, two equations which differ only in the order of the summands are considered the same).

The example above was in base 10, but we're also interested in cryptarithm equations in other bases. Note that a "digit" in base b could mean any integer between 0 and b-1. Here is a cryptarithm equation in base 23:

```
 I7B
 JJJ
----
1F47
```

In this example, "I" stands for digit 18, "B" stands for digit 11, "J" stands for digit 19, and "F" stands for digit 15. In decimal notation, the two summands are $18*23^2 + 7*23 + 11 = 9694$ and $19*23^2 + 19*23 + 19 = 10507$, and the sum is $1*23^3 + 15*23^2 + 4*23 + 7 = 20201$. Please note that denoting digits of 10 and more with letters was done purely for the clarity of the example; it doesn't really matter in this problem how exactly we denote such digits in writing.

How many cryptarithm equations are there with the given sum **N** in the given base **B**?

Since the answer might be very large, please output it modulo 1000000007.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each contains two positive integers **N** and **B**. All input numbers are given in base 10.

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 number of different cryptarithm equations with the given sum. Since this number can be very big, please output it modulo 1000000007. Of course, the output itself should be in base 10.

Limits

$1 \le \mathbf{T} \le 20$.

Small dataset

$1 \le \mathbf{N} \le 100$.
$2 \le \mathbf{B} \le 10$.

Large dataset

$1 \le \mathbf{N} \le 10^{18}$.
$2 \le \mathbf{B} \le 70$.

Sample

Input	Output
2 6 10 8 4	Case #1: 4 Case #2: 4

Explanation

Here are the 4 cryptarithm equations with sum 6:

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

And here are the 4 cryptarithm equations in base 4 with sum $8=20_4$:

```
20   11   13   10
--    3    1    3
20   --   --    1
     20   20   --
               20
```

Problem A. **Letter Stamper**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 8 points	Solve A-small
Large input 19 points	Solve A-large

Problem

Roland is a high-school math teacher. Every day, he gets hundreds of papers from his students. For each paper, he carefully chooses a letter grade: 'A', 'B' or 'C'. (Roland's students are too smart to get lower grades like a 'D' or an 'F'). Once the grades are all decided, Roland passes the papers onto his assistant - you. Your job is to stamp the correct grade onto each paper.

You have a low-tech but functional letter stamp that you use for this. To print out a letter, you attach a special plate to the front of the stamp corresponding to that letter, dip it in ink, and then apply it to the paper.

The interesting thing is that instead of removing the plate when you want to switch letters, you can just put a new plate on top of the old one. In fact, you can think of the plates on the letter stamp as being a stack, supporting the following operations:

- Push a letter on to the top of the stack. (This corresponds to attaching a new plate to the front of the stamp.)
- Pop a letter from the top of the stack. (This corresponds to removing the plate from the front of the stamp.)
- Print the letter on the top of the stack. (This corresponds to actually using the stamp.) Of course, the stack must actually have a letter on it for this to work.

Given a sequence of letter grades ('A', 'B', and 'C'), how many operations do you need to print the whole sequence in order? The stack begins empty, and you must empty it when you are done. However, you have unlimited supplies of each kind of plate that you can use in the meantime.

For example, if you wanted to print the sequence "ABCCBA", then you could do it in 12 operations, as shown below:

```
Operation     Printed so far     Stack

0. -          -                  -
1. Push A     -                  A
2. Print      A                  A
3. Push B     A                  AB
4. Print      AB                 AB
5. Push C     AB                 ABC
6. Print      ABC                ABC
7. Print      ABCC               ABC
8. Pop        ABCC               AB
9. Print      ABCCB              AB
10. Pop       ABCCB              A
11. Print     ABCCBA             A
12. Pop       ABCCBA             -
```

Input

The first line of the input file contains the number of cases, **T**. **T** test cases follow, one per line. Each of these lines contains a single string **S**, representing the sequence of characters that you want to print out in order.

Output

For each test case, output one line containing "Case #x: N", where x is the case number (starting from 1) and N is the minimum number of stack operations required to print out **S**.

Limits

S is a non-empty string containing only the letters 'A', 'B', and 'C'.

Small dataset

$1 \le \mathbf{T} \le 100$.
S has at most 100 characters.

Large dataset

1 ≤ **T** ≤ 20.
S has at most 7000 characters.

Sample

Input	Output
`2` `ABCCBA` `AAABAAB`	`Case #1: 12` `Case #2: 13`

Submissions

Letter Stamper
8pt Not attempted **20/22 users** correct (91%)
19pt Not attempted **5/10 users** correct (50%)

City Tour
4pt Not attempted **21/21 users** correct (100%)
23pt Not attempted **19/21 users** correct (90%)

Candy Store
7pt Not attempted **21/21 users** correct (100%)
20pt Not attempted **12/13 users** correct (92%)

Travel Plan
3pt Not attempted **22/23 users** correct (96%)
30pt Not attempted **17/18 users** correct (94%)

Ninjutsu
11pt Not attempted **6/8 users** correct (75%)
23pt Not attempted **0/2 users** correct (0%)

The Paths of Yin Yang
17pt Not attempted **1/2 users** correct (50%)
35pt Not attempted

Top Scores

Egor	125
krijgertje	114
Burunduk1	112
ACRush	106
marek.cygan	95
meret	95
rng..58	95
pashka	95
iwi	95
eatmore	94

Problem B. City Tour

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 4 points	Solve B-small
Large input 23 points	Solve B-large

Problem

During summer time, old cities in Europe are swarming with tourists who roam the streets and visit points of interest.

Many old cities were built organically and not according to some architecture plan, but, strangely, their growth exhibits a similar pattern: the cities started from three points of interest, with each pair being connected by a bidirectional street; then, gradually, new points of interest were added. Any new point of interest was connected by two new bidirectional streets to two different previous points of interest which were already directly connected by a street.

A tourist visiting such a city would like to do a tour visiting as many points of interest as possible. The tour can start at any point of interest and must end at the same point of interest. The tour may visit each street at most once and each point of interest at most once (with the exception of the first point of interest which is visited exactly twice).

You are given the description of how the city grew. Find the largest number of different points of interest a single tour can visit in this city.

Input

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

Each case begins with the integer **N** - the total number of points of interest in the city. Points are denoted with numbers from 1 to **N**; numbers 1, 2, and 3 denote the three original points when the city started, while numbers 4, ..., **N** denote the other points in the order they were added to the city.

The next **N**-3 lines each contain a pair of space-separated integers **A**, **B**, indicating that the corresponding point of interest was connected by streets to points **A** and **B**. First of these lines corresponds to point number 4, second to point number 5, etc.

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 largest number of points of interest a tour can visit in this city.

Limits

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

Small dataset

$4 \leq \mathbf{N} \leq 15$.

Large dataset

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

Sample

Input

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

Output

```
Case #1: 4
Case #2: 6
```

Problem C. Candy Store

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 7 points	Solve C-small
Large input 20 points	Solve C-large

Problem

Owning a candy store is tough! You have to optimize all kinds of things. Lately you've been selling a very popular kind of candy called Whizboppers. These candies become rotten very quickly, which gives them the following properties:

- You must buy new Whizboppers from your supplier every morning.
- You must sell Whizboppers in the boxes you bought from your supplier that morning.

You can order Whizboppers from your supplier in boxes that contain any integer number of grams.

Every day up to **k** people visit your store, and, starting from the first person, they will choose an integer number of cents to spend on Whizboppers: between 1 and **C** cents inclusive. You're going to sell Whizboppers for 1 cent per gram; so if a person wants to spend 4 cents, you will give that person exactly 4 grams of candy. You might do this by giving the person a 4-gram box, or perhaps a 2-gram box and two 1-gram boxes.

What is the minimum number of boxes you need to order so that, no matter what amount each person orders, you can always give all of the people the mass of Whizboppers they want?

Note: When a person chooses how much candy to buy, you know what other people have already bought, but you don't know what future people will buy.

For example, if up to 2 people visit your store every day, and they spend up to 2 cents each (k=2, C=2), you could buy four 1-gram boxes from your supplier. But you can do better: if you buy two 1-gram boxes and one 2-gram box, you can satisfy your customers. Here's how:

```
First Person   Boxes given   Second Person   Boxes given
--------------------------------------------------------
  2 cents      1 x 2-gram       2 cents       2 x 1-gram
                                1 cent        1 x 1-gram
  ----------------------------------------------------
  1 cent       1 x 1-gram       2 cents       1 x 2-gram
                                1 cent        1 x 1-gram
```

Regardless of what the first person orders, you can give out boxes so that the second person can still get the right amount of candy. So for k=2, C=2, you can serve any sequence of orders with 3 boxes.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow, each of which contains two integers: **k** and **C**, the maximum number of people and the maximum number of cents each person may spend.

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 minimum number of boxes you need to order every day.

Limits

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

Small dataset

$1 \le \mathbf{k} \le 20$.
$1 \le \mathbf{C} \le 3$.

Large dataset

$1 \le \mathbf{k} \le 1000$.
$1 \le \mathbf{C} \le 10^{12}$.

Sample

```
Input      Output

4          Case #1: 3
1 5        Case #2: 3
2 2        Case #3: 19
10 3       Case #4: 11
2 50
```

Explanation

In the first case, you can buy one 1-gram box and two 2-gram boxes. In the second case, you can buy two 1-gram boxes and one 2-gram box.

Submissions

Letter Stamper
8pt Not attempted **20/22 users** correct (91%)
19pt Not attempted **5/10 users** correct (50%)

City Tour
4pt Not attempted **21/21 users** correct (100%)
23pt Not attempted **19/21 users** correct (90%)

Candy Store
7pt Not attempted **21/21 users** correct (100%)
20pt Not attempted **12/13 users** correct (92%)

Travel Plan
3pt Not attempted **22/23 users** correct (96%)
30pt Not attempted **17/18 users** correct (94%)

Ninjutsu
11pt Not attempted **6/8 users** correct (75%)
23pt Not attempted **0/2 users** correct (0%)

The Paths of Yin Yang
17pt Not attempted **1/2 users** correct (50%)
35pt Not attempted

Top Scores

Egor	125
krijgertje	114
Burunduk1	112
ACRush	106
marek.cygan	95
meret	95
rng..58	95
pashka	95
iwi	95
eatmore	94

Problem D. Travel Plan

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 3 points	Solve D-small
Large input 30 points	Solve D-large

Problem

In a yet-to-be-announced and rechecked discovery by Antarctic astronomers, it is written that there are **N** inhabited planets in space, all lying along the same straight line, with the **i**-th planet lying at coordinate $\mathbf{X_i}$ along the line (**i** = 1, 2, ..., **N**). Earth is the first planet, lying at coordinate zero, so $\mathbf{X_1}$ will always be equal to 0.

Being very excited about this fact, you start planning a trip to visit all the planets. Since unknown planets can be dangerous, you want to visit each planet exactly once before returning to Earth. You have **F** units of fuel, and you want to spend as much of it on this trip as possible so that your final landing on Earth is safer. Your spaceship is pretty basic and can only fly along a straight line from any planet **i** to any other planet **j**, consuming $|\mathbf{X_i}-\mathbf{X_j}|$ units of fuel along the way. It can't turn without landing.

So you need to create a travel plan that requires at most **F** units of fuel, starts from Earth, visits each of the other planets exactly once, and then returns to Earth. If there are several such plans, you should find the one that consumes most fuel. Output the amount of fuel consumed.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case description starts with a line containing the number of planets **N**. The next line contains **N** numbers $\mathbf{X_i}$, the coordinates of the planets. The next line contains the amount of fuel **F** that you have.

Output

For each test case, output one line containing either "Case #x: NO SOLUTION", when there's no such travel plan, or "Case #x: y", where x is the case number (starting from 1) and y is the maximum amount of fuel consumed.

Limits

$1 \le \mathbf{F} \le 10^{17}$.
$-10^{15} \le \mathbf{X_i} \le 10^{15}$.
$\mathbf{X_1}=0$.
All $\mathbf{X_i}$ are different.

Small dataset

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 10$.

Large dataset

$1 \le \mathbf{T} \le 20$.
$2 \le \mathbf{N} \le 30$.

Sample

Input	Output
3 3 0 10 -10 40 5 0 1 2 3 4 13 5 0 1 2 3 4 7	Case #1: 40 Case #2: 12 Case #3: NO SOLUTION

Submissions

Letter Stamper

8pt	Not attempted **20/22 users** correct (91%)
19pt	Not attempted **5/10 users** correct (50%)

City Tour

4pt	Not attempted **21/21 users** correct (100%)
23pt	Not attempted **19/21 users** correct (90%)

Candy Store

7pt	Not attempted **21/21 users** correct (100%)
20pt	Not attempted **12/13 users** correct (92%)

Travel Plan

3pt	Not attempted **22/23 users** correct (96%)
30pt	Not attempted **17/18 users** correct (94%)

Ninjutsu

11pt	Not attempted **6/8 users** correct (75%)
23pt	Not attempted **0/2 users** correct (0%)

The Paths of Yin Yang

17pt	Not attempted **1/2 users** correct (50%)
35pt	Not attempted

Top Scores

Egor	125
krijgertje	114
Burunduk1	112
ACRush	106
marek.cygan	95
meret	95
rng..58	95
pashka	95
iwi	95
eatmore	94

Problem E. **Ninjutsu**

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 11 points	Solve E-small
Large input 23 points	Solve E-large

Problem

Ninjutsu is the martial art of the mysterious Japanese assassins, ninja. As a beginner in the practice of ninjutsu, your first task is to master the use of the grappling hook.

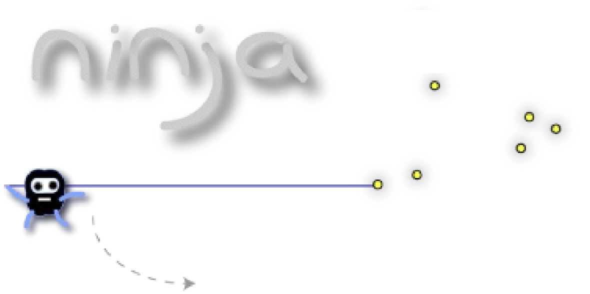
A grappling hook is a technologically-advanced device consisting of a hook tied to some (very strong and very thin) rope. Proper use of a grappling hook involves throwing the hook at a target and hoping that it catches.

This time, it did! You are now hooked onto a target that is located at (0, 0). Your rope extends to the left, and you're at the end of it; when you jump, you will start swinging counter-clockwise around the target. There are other targets located to the right and above (0, 0), at (x_i, y_i)) with $x_i \geq 0$ and $y_i \geq 0$ When an interior point of the rope (not either end) contacts one or more targets, the rope will bend around the target closest to its moving end. Ignore your starting velocity; because you are a ninja, it is fast enough that you will continue bending around targets until you are spinning around a single one.

Your rope currently has length **R**, but you may choose to cut it down to any shorter length **r** (including non-integers) before you start swinging. As such, you will start at (-**r**, 0) and swing down (counter-clockwise) toward (0, -**r**).

What is the maximum number of bends you can put into the rope with one swing? A bend happens when your rope touches a target and then rotates some non-zero number of degrees around that target. The rope will always remain perfectly straight (again, this is possible because you are a ninja), except at bends.

Example



In the example above, there are 6 points:

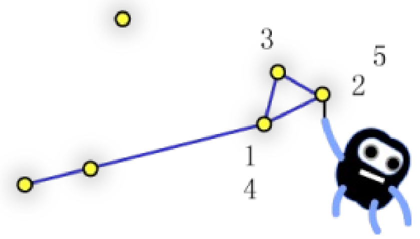
- (0, 0),
- (3, 1),
- (12, 4),
- (14, 5),
- (13, 7), and
- (7, 10).

You have a rope of length 24. If you do not cut the rope, then you will bend around point (12, 4), then around point (14, 5), then around point (13, 7), and finally, you will be stuck orbiting point (7, 10) with about 0.1705 units of rope remaining. This amounts to a total of 4 bends. Although you touch point (3, 1), it does not contribute a bend because it is collinear with the points (0, 0) and (12, 4).

If, however, you cut the rope by 0.18 units, you will not have enough length to reach point (7, 10) and will instead follow the path

```
(0, 0)--(12, 4)--(14, 5)--(13, 7)--(12, 4)--(14, 5)
```

and will end up orbiting point (14, 5) with about 1.3004 units of rope remaining. This path amounts to 5 bends, in total, and is an optimal solution.



Case #1 in the sample input below represents this example.

Input

The input will start with a line containing **T**, the number of test cases to follow. Each test case will start with two integers together on a line: **N** and **R**. The next **N** lines will each contain a pair of integers -- $\mathbf{x_i}$ and $\mathbf{y_i}$ -- the coordinates of the targets, starting with the target at (0, 0).

Output

For each test case, output a line of the form "Case #**C**: **k**", where **C** is the 1-based case number, and **k** is the maximum number of bends that can be made in the rope in one swing.

Limits

1 ≤ **T** ≤ 100
All target coordinates are integers.
All targets will be at different locations.
The first target listed will be located at (0, 0).
There will be at least one value of **r** that gives an optimal solution and has the property that a rope of length **r** - 0.999999 also gives the same solution (the same sequence of bends).

Small dataset

1 ≤ **N** ≤ 10
1 ≤ **R** ≤ 1,000
0 ≤ $\mathbf{x_i}$ ≤ 1,000
0 ≤ $\mathbf{y_i}$ ≤ 1,000

Large dataset

1 ≤ **N** ≤ 1,000
1 ≤ **R** ≤ 10^9
0 ≤ $\mathbf{x_i}$ ≤ 10^9
0 ≤ $\mathbf{y_i}$ ≤ 10^9

Sample

Input

```
6
6 24
0 0
3 1
12 4
14 5
13 7
7 10
2 1
0 0
2 0
2 1
0 0
1 0
2 10
0 0
4 0
3 50
0 0
9 0
10 0
3 12
0 0
3 0
3 4
```

Output

```
Case #1: 5
Case #2: 0
Case #3: 0
Case #4: 2
Case #5: 12
Case #6: 3
```

Problem F. The Paths of Yin Yang

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

Small input 17 points	Solve F-small
Large input 35 points	Solve F-large

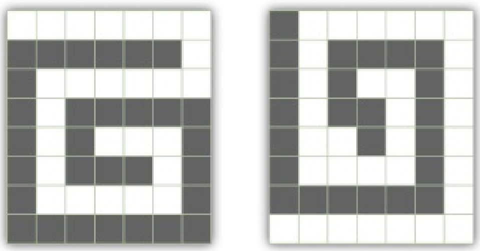
So, If and Else grow out of each other;
Hardness and Tractability complete each other;
Long int and Short int shape each other;
High bits and Low bits determine each other;
Music and Voice give harmony to each other;
Push_front and Push_back give sequence to each other.
-- Tao Te Ching, Laozi, Zhou dynasty, ancient China.
Translated (loosely) by yours truly.

Problem

Given an rectangular grid of **N** rows and **M** columns, each cell can be labeled black (Yin) or white (Yang). Two cells are *neighbors* if they share a common unit-length edge segment. The grid is *valid* if all the black cells form a path, and all the white cells form a path. A *path* is a set S of cells defined as follows:

- The cells form a connected piece. From each cell in S, you can reach any other cell in S by moving between neighbors within S.
- Exactly two cells in S have exactly one neighbor in S each. These are the "ends" of the path.
- Every other cell in S has exactly two neighbors in S.

For example, in the picture below, the first grid is valid, while the second grid is not -- although the black cells form a path, the white cells do not.



Given **N** and **M**, compute the number of valid grids. Note that symmetry doesn't matter -- as long as two valid grids differ in one position they are considered different, even if one can be rotated or flipped to the other.

Input

The first line of the input will be a single integer **T**, the number of test cases. **T** lines follow, each of which contains two integers separated by a space: "**N M**", as defined above.

Output

For each test case, output a line in the form "Case #**x**: **A**", where **x** is the case number, starting from 1, and **A** is the number of valid grids of the specified size.

Limits

1 ≤ **T** ≤ 50

Small dataset

4 ≤ **N**, **M** ≤ 10

Large dataset

For 80% of the test cases, 4 ≤ **N**, **M** ≤ 50
For 90% of the test cases, 4 ≤ **N**, **M** ≤ 70
For all test cases, 4 ≤ **N**, **M** ≤ 100

Sample

Submissions

Letter Stamper
- 8pt Not attempted **20/22 users** correct (91%)
- 19pt Not attempted **5/10 users** correct (50%)

City Tour
- 4pt Not attempted **21/21 users** correct (100%)
- 23pt Not attempted **19/21 users** correct (90%)

Candy Store
- 7pt Not attempted **21/21 users** correct (100%)
- 20pt Not attempted **12/13 users** correct (92%)

Travel Plan
- 3pt Not attempted **22/23 users** correct (96%)
- 30pt Not attempted **17/18 users** correct (94%)

Ninjutsu
- 11pt Not attempted **6/8 users** correct (75%)
- 23pt Not attempted **0/2 users** correct (0%)

The Paths of Yin Yang
- 17pt Not attempted **1/2 users** correct (50%)
- 35pt Not attempted

Top Scores

Egor	125
krijgertje	114
Burunduk1	112
ACRush	106
marek.cygan	95
meret	95
rng..58	95
pashka	95
iwi	95
eatmore	94

Input	Output
3 4 4 4 6 5 5	Case #1: 24 Case #2: 44 Case #3: 48