“Ability is what you're capable of doing. Motivation determines what you do. Attitude determines how well you do it.” - Lou Holtz

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

$1 \le \mathbf{T} \le 10{,}000$.
$1 \le \mathbf{N} \le 30$;
$0 \le \mathbf{K} \le 10^8$;

Sample Input

```
4
1 0
1 1
4 0
4 47
```

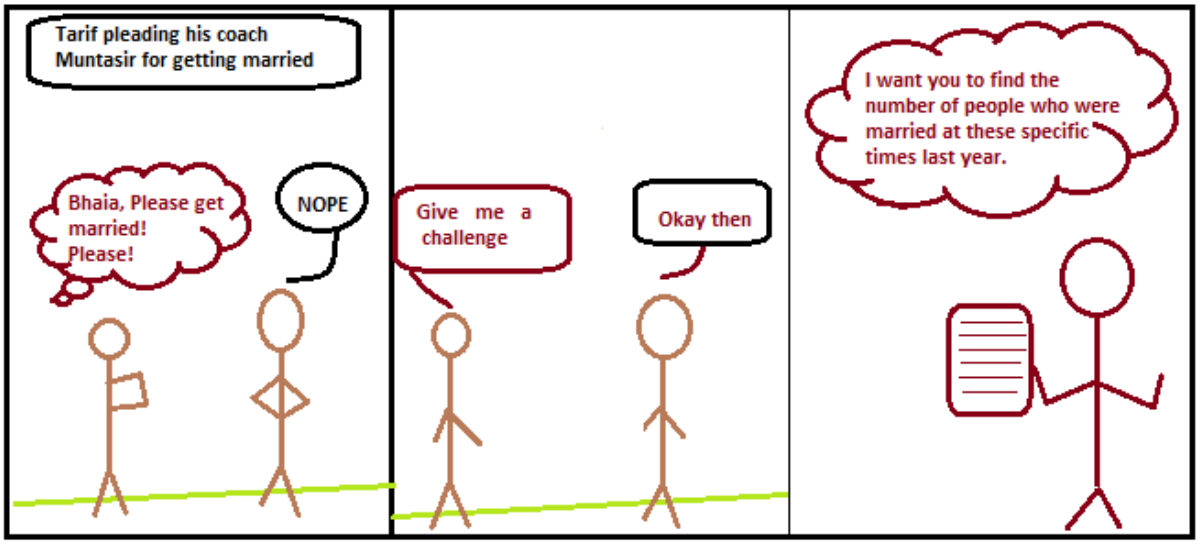
Sample Output

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

Tarif's Challenge

As you all know, angry young man Tarif wants to get married badly. But because of an unbreakable law, he cannot get married before his coaches. But for his tough luck, none of his coaches agreed to get married. Fortunately, Tarif's hopes aren't over yet. Another law dictates that should a contestant desire, he could force one of his coaches to marry, by performing a potentially problematic challenge asked by them. If he could complete the challenge, the coach is bound by law to get married. But should the contestant fail, he would not be able to get married ever again.

After a lot of thinking, Tarif decided to approach one of his favorite mentors Muntasir Azam Khan and ask for a challenge.



Tarif was not allowed to use computers and he had to do everything by hand. But he was a clever boy and figured out a loop hole. He discovered that according to Muntasir's specifications, there was no restriction on outsourcing that job to someone else. So, he asked you to solve the task for him. As you are capable of solving it with the help of a computer, he asked for your help.

Input Specification:

The first line of input will contain **T (1 <= T <= 20)**. The next line will house one integer **Q (1 <= Q <= 300000).** Then **Q** lines follow, each of them will be one of the following types:

- "**A** + **B**" – means that A and B got married.
- "**A** – **B**" – means that A and B got divorced.
  Constraints: **(1 <= A , B <= 100000)**

Output Specification:

For each of the **Q** queries, print one line containing the number of people married, in the following format.

Input	Output
1 3 1 + 2 3 + 4 1 – 2	Case 1: 2 4 2

Beware of the constraints

Use Fast I/O like printf, scanf

etc.

“The Enemy of the best is the good. If you're always settling with what's good, you'll never be the best.”
— Jerry Rice

Problem C. Theme Park

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

$1 \le \mathbf{T} \le 50$.
$\mathbf{g_i} \le \mathbf{k}$.
$1 \le \mathbf{R} \le 1000$.
$1 \le \mathbf{k} \le 100$.
$1 \le \mathbf{N} \le 10$.

$1 \le \mathbf{g}_i \le 10$.

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
```

Sample Output

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

“I learned a long time ago that there is something worse than missing the goal, and that's not pulling the trigger.” - Mia Hamm

Problem D. Magicka

Introduction

Magicka™ is an action-adventure game developed by Arrowhead Game Studios. In Magicka you play a wizard, invoking and combining elements to create Magicks. This problem has a similar idea, but it does not assume that you have played Magicka.
Note: "invoke" means "call on." For this problem, it is a technical term and you don't need to know its normal English meaning.

Problem

As a wizard, you can **invoke** eight elements, which are the "base" elements. Each base element is a single character from {Q, W, E, R, A, S, D, F}. When you invoke an element, it gets appended to your **element list**. For example: if you invoke W and then invoke A, (we'll call that "invoking WA" for short) then your element list will be [W, A].
We will specify pairs of base elements that **combine** to form non-base elements (the other 18 capital letters). For example, Q and F might combine to form T. If the two elements from a pair appear at the end of the element list, then both elements of the pair will be immediately removed, and they will be replaced by the element they form. In the example above, if the element list looks like [A, Q, F] or [A, F, Q] at any point, it will become [A, T].
We will specify pairs of base elements that are **opposed** to each other. After you invoke an element, if it isn't immediately combined to form another element, and it is opposed to something in your element list, then your whole element list will be cleared.
For example, suppose Q and F combine to make T. R and F are opposed to each other. Then invoking the following things (in order, from left to right) will have the following results:

- QF → [T] (Q and F combine to form T)
- QEF → [Q, E, F] (Q and F can't combine because they were never at the end of the element list together)
- RFE → [E] (F and R are opposed, so the list is cleared; then E is invoked)
- REF → [] (F and R are opposed, so the list is cleared)
- RQF → [R, T] (QF combine to make T, so the list is not cleared)
- RFQ → [Q] (F and R are opposed, so the list is cleared)

Given a list of elements to invoke, what will be in the element list when you're done?

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 line, containing the following space-separated elements in order:
First an integer **C**, followed by **C** strings, each containing three characters: two base elements

followed by a non-base element. This indicates that the two base elements combine to form the non-base element. Next will come an integer **D**, followed by **D** strings, each containing two characters: two base elements that are opposed to each other. Finally there will be an integer **N**, followed by a single string containing **N** characters: the series of base elements you are to invoke. You will invoke them in the order they appear in the string (leftmost character first, and so on), one at a time.

Output

For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is a list in the format "[e_0, e_1, ...]" where e_i is the i^{th} element of the final element list. Please see the sample output for examples.

Limits

$1 \leq$ **T** ≤ 100.
Each pair of base elements may only appear together in one combination, though they may appear in a combination and also be opposed to each other.
No base element may be opposed to itself.
Unlike in the computer game Magicka, there is no limit to the length of the element list.
$0 \leq$ **C** ≤ 36.
$0 \leq$ **D** ≤ 28.
$1 \leq$ **N** ≤ 100.

Sample Input	**Sample Output**
5	Case #1: [E, A]
0 0 2 EA	Case #2: [R, I, R]
1 QRI 0 4 RRQR	Case #3: [F, D, T]
1 QFT 1 QF 7 FAQFDFQ	Case #4: [Z, E, R, A]
1 EEZ 1 QE 7 QEEEERA	Case #5: []
0 1 QW 2 QW	

If you think about racing too much you may just lose it a little bit. - Usain Bolt

Problem E. Bot Trust

Problem

Blue and Orange are friendly robots. An evil computer mastermind has locked them up in separate hallways to test them, and then possibly give them cake.
Each hallway contains 100 buttons labeled with the positive integers {1, 2, ..., 100}. Button k is always k meters from the start of the hallway, and the robots both begin at button 1. Over the period of one second, a robot can walk one meter in either direction, or it can press the button at its position once, or it can stay at its position and not press the button. To complete the test, the robots need to push a certain sequence of buttons in a certain order. Both robots know the full sequence in advance. How fast can they complete it?
For example, let's consider the following button sequence:

O 2, B 1, B 2, O 4

Here, O 2 means button 2 in Orange's hallway, B 1 means button 1 in Blue's hallway, and so on. The robots can push this sequence of buttons in 6 seconds using the strategy shown below:

Time	Orange	Blue
1	Move to button 2	Stay at button 1
2	Push button 2	Stay at button 1
3	Move to button 3	Push button 1
4	Move to button 4	Move to button 2
5	Stay at button 4	Push button 2
6	Push button 4	Stay at button 2

Note that Blue has to wait until Orange has completely finished pushing O 2 before it can start pushing B 1.

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 line beginning with a positive integer **N**, representing the number of buttons that need to be pressed. This is followed by **N** terms of the form "$\mathbf{R}_i$ $\mathbf{P}_i$" where $\mathbf{R}_i$ is a robot color (always 'O' or 'B'), and $\mathbf{P}_i$ is a button position.

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 seconds required for the robots to push the given buttons, in order.

Limits

$1 \le \mathbf{P}_i \le 100$ for all i.

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

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

Sample Input

```
3
4 O 2 B 1 B 2 O 4
3 O 5 O 8 B 100
2 B 2 B 1
```

Sample Output

```
Case #1: 6
Case #2: 100
Case #3: 4
```