Code Jam 2019 - Round 2

New Elements: Part 1

Problem

The first two paragraphs (not counting this one) of this problem and "New Elements: Part 2" are identical. The problems can otherwise be solved independently; you do not need to read or solve one in order to read or solve the other.

Muriel is on the path to discovering two new elements that she has named Codium and Jamarium. She has not been able to isolate them yet, but she wants to start investigating some important properties, like their atomic weights, by indirect means. Since Muriel is working with a single isotope of Codium and a single isotope of Jamarium, their atomic weights are strictly positive integers.

Muriel managed to create **N** different molecules, each of which contains one or more atoms of Codium and one or more atoms of Jamarium, and no other elements. For each molecule, she knows how many atoms of each element are present in it. The molecular weight of a molecule is the sum of the atomic weights of all the atoms it contains.

As a first step towards figuring out exact molecular weights for the molecules and atomic weights for the two elements, Muriel wants to sort the molecules by strictly increasing molecular weight. To assess the difficulty of that task, she wants to know how many orders are valid considering only the information she has right now. An ordering of the molecules is considered valid if there exist values for the atomic weights of Codium and Jamarium such that the ordering is *strictly* increasing in molecular weight.

To give an example, we represent each molecule by the ordered pair of the number of atoms of Codium and Jamarium it contains. If Muriel has 3 molecules represented by (1, 1), (2, 1) and (1, 2), there are two possible orderings that can be strictly increasing in molecular weight: (1, 1), (1, 2), (2, 1) and (1, 1), (2, 1), (1, 2). The first ordering is valid for any assignment of atomic weights in which Codium is the heaviest of the two elements, and the second is valid for any assignment in which Jamarium is the heaviest. The only case remaining is when both Codium and Jamarium have the same atomic weight, in which case (1, 2) and (2, 1) have the same molecular weight, so no strictly increasing ordering can be produced for that scenario.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of a test case contains a single integer **N**, the number of molecules. Each of the next **N** lines describes a different molecule with two integers $\mathbf{C_i}$ and $\mathbf{J_i}$ that represent the number of Codium and Jamarium atoms in the i-th molecule, respectively.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1) and `y` is the total number of valid orderings as defined above.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{C_i} \leq 10^9$, for all i.
$1 \leq \mathbf{J_i} \leq 10^9$, for all i.
$(\mathbf{C_i}, \mathbf{J_i}) \neq (\mathbf{C_j}, \mathbf{J_j})$ for all $i \neq j$. (All molecules are different.)

Test set 1 (Visible)

$2 \leq \mathbf{N} \leq 6$.

Test set 2 (Hidden)

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

Sample

Sample Input

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

Sample Output

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

Sample Case #1 is explained in the statement.

In Sample Case #2, the two valid orderings are (1, 2), (2, 1), (2, 4), (4, 2) and (2, 1), (1, 2), (4, 2), (2, 4). Notice that the ordering (1, 2), (2, 1), (4, 2), (2, 4) is invalid because if (1, 2) is strictly less heavy than (2, 1), then (2, 4), which is exactly twice as heavy as (1, 2), must be strictly less heavy than (4, 2), which is exactly twice as heavy as (2, 1).

Code Jam 2019 - Round 2

Pottery Lottery

Problem

The Pottery Palace is going to run a lottery featuring some valuable vases by the artist Cody-Jamal. The lottery works as follows:

- 100 people get to play in the lottery. Each player has a unique number between 1 and 100, and is given a single token with that number.
- There are 20 empty clay vases on a table, numbered 1 through 20. The vases have narrow openings that are large enough to accept a token, but small enough that players cannot look inside to see the contents.
- On the i-th day of the lottery, the player with token number i chooses a vase and puts their token in that vase. Since the vases are all identical (apart from their labels), every player will choose one uniformly at random and independently of all other players' choices.
- On day 100, after player number 100 has inserted their token, the organizers shake the vases to determine how many tokens are inside each one. If there is *exactly* one vase that has fewer tokens than any other vase, then that one is the "winning vase". The organizers then pour out all of the tokens in that vase, and every player whose number is written on one of those poured-out tokens wins a vase! If multiple vases have the same minimal amount of tokens, nobody wins anything.

You have been hired to test the security of the lottery, and you will participate in some trial runs. The company will always assign you the number 100 — that is, you replace player 100.

You have found some ways to tamper with the lottery at night, but security is tight, so you can only do so much! Specifically, after each of the first 99 days of the lottery, you may do exactly *one* of the following:

- forge a token with the player number of your choice (between 1 and 100, inclusive), and add it to a vase of your choice. You are a very good forger: if there is a winning vase, any forged tokens in that vase will cause the players with those numbers to win (with one exception; see below).
- use a special camera to see the numbers on all of the tokens in one vase of your choice

You may perform different actions on different nights, and you may choose dynamically: you do not need to decide on all of your actions in advance.

On the 100th day, it is your turn to insert your token into a vase of your choice (you do not need to choose uniformly at random). You cannot perform any other actions on that day.

You know that if there is a winning vase with more than one token for the same player, it will be obvious that cheating has occurred and nobody will win. However, it does not matter if other vases contain more than one token for the same player because the organizers never see those tokens.

Your goal is to be a winner in at least 90% of the test cases.

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially, your program should read a single line containing a single integer **T** indicating the number of test cases. Then, you need to process **T** test cases.

At the start of a test case, the judge outputs one line with one integer: the number of the current day. (The judge starts on day 1, and on the i-th day, it prints i.) After your program reads the integer, it should write a line containing two integers V and P, with $1 \le V \le 20$, and $0 \le P \le 100$. The judge will interpret these as follows:

- If $1 \le P \le 100$, you put a token for player P in vase V. The judge does not write anything back as a response.

- If P = 0, you inspect the contents of vase V. The judge writes one line containing integers. The first integer is **N**, the number of tokens in vase V, and then there are **N** more integers: the player numbers on each of the tokens, in non-decreasing order.

Notice that on turn 100, you must put your own token in, so P must be 100.

Remember that on the i-th day, for 1 ≤ i ≤ 99, the judge simulates the action of the i-th player, as described in the statement. This happens before your own action on that day.

After you send your move for turn 100, your program should terminate if it was the last test case; otherwise, it should start reading data for the next test case. (Notice that the judge does not tell you whether you got each case correct or incorrect.) The judge will only check whether you have enough correct answers after you have attempted all **T** test cases, so you should not stop early! For example, if you answer the first 225 out of 250 cases correctly and then exit, or provide malformed input, your solution will not be considered correct.

If your program outputs something illegal (e.g., gives an invalid value for P or V, or tries to inspect a vase on turn 100), the judge will send one line containing `-1` to your input stream, and it will not send any other output after that. If your program continues to wait for the judge after receiving `-1`, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the total memory is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

Test set 1 (Visible)

T = 250.
Time limit (for the entire test set): 40 seconds.
Memory limit: 1GB.

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Sample Interaction

```
t = readline_int()            // reads 250 into t
curr_day = readline_int()     // reads 1 (day 1)
printline 8 100 to stdout     // puts a token for player 100 into vase 8
flush stdout
curr_day = readline_int()     // reads 2 (day 2)
printline 8 99 to stdout      // puts a token for player 99 into vase 8
flush stdout
curr_day = readline_int()     // reads 3 (day 3)
printline 8 100 to stdout     // puts a token for player 100 into vase 8
flush stdout
curr_day = readline_int()     // reads 4 (day 4)
printline 20 7 to stdout      // puts a token for player 7 into vase 20
flush stdout
curr_day = readline_int()     // reads 5 (day 5)
printline 8 0 to stdout       // inspects vase 8
flush stdout
tokens = readline_int_list() // reads 5 2 5 99 100 100 (players 2 and 5
```

```
                                //   happen to have chosen vase 8)
curr_day = readline_int()       // reads 6 (day 6)
printline 8 101 to stdout       // tries to add a token with a bad player number
flush stdout
curr_day = readline_int()       // reads -1 (judge has decided our solution is
                                //   incorrect)
exit                            // exits to avoid an ambiguous TLE error
```

Code Jam 2019 - Round 2

New Elements: Part 2

Problem

The first two paragraphs (not counting this one) of this problem and "New Elements: Part 1" are identical. The problems can otherwise be solved independently; you do not need to read or solve one in order to read or solve the other.

Muriel is on the path to discovering two new elements that she has named Codium and Jamarium. She has not been able to isolate them yet, but she wants to start investigating some important properties, like their atomic weights, by indirect means. Since Muriel is working with a single isotope of Codium and a single isotope of Jamarium, their atomic weights are strictly positive integers.

Muriel managed to create **N** different molecules, each of which contains one or more atoms of Codium and one or more atoms of Jamarium, and no other elements. For each molecule, she knows how many atoms of each element are present in it. The molecular weight of a molecule is the sum of the atomic weights of all the atoms it contains.

As a first step, Muriel sorted the molecules by strictly increasing molecular weight. Now she wants to find out possible integer values for the atomic weights of both Codium and Jamarium that are consistent with the ordering. Since she is aware there could be many consistent pairs of values, she wants one that minimizes the atomic weight of Codium. If there are multiple pairs in which Codium's atomic weight is minimum, she wants the one in which Jamarium's atomic weight is minimum.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. The first line of a test case contains a single integer **N**, the number of molecules. Each of the next **N** lines describes a different molecule with two integers $\mathbf{C_i}$ and $\mathbf{J_i}$ that represent the number of Codium and Jamarium atoms in the i-th molecule, respectively. The molecules are given in strictly increasing order of molecular weight.

Output

For each test case, output one line containing `Case #x: y`, where `x` is the test case number (starting from 1), and `y` is `IMPOSSIBLE` (in uppercase) if there is no pair of integer atomic weights that would make the order of the molecules strictly increasing in molecular weight. Otherwise, `y` should be two integers `c j` where c is the atomic weight of Codium and j is the atomic weight of Jamarium, chosen according to the rules above.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \leq \mathbf{T} \leq 100$.
$2 \leq \mathbf{N} \leq 10$.
$(\mathbf{C_i}, \mathbf{J_i}) \neq (\mathbf{C_j}, \mathbf{J_j})$ for all $i \neq j$. (All molecules are different.)

Test set 1 (Visible)

$1 \leq \mathbf{C_i} \leq 100$, for all i.
$1 \leq \mathbf{J_i} \leq 100$, for all i.

Test set 2 (Hidden)

$1 \leq \mathbf{C_i} \leq 10^9$, for all i.
$1 \leq \mathbf{J_i} \leq 10^9$, for all i.

Sample

Sample Input

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

Sample Output

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

In Sample Case #1, the difference between the last two molecules is having an extra atom of one element or the other. Given that the one having the extra Codium is heavier overall, we conclude that Codium must be heavier than Jamarium. The values 2 and 1 for the atomic weights of Codium and Jamarium make the molecular weights 1 × 2 + 1 × 1 = 3, 1 × 2 + 2 × 1 = 4, and 2 × 2 + 1 × 1 = 5, respecting the strict ordering. Since Codium is heavier than Jamarium in this case, 2 is Codium's minimum atomic weight, and 1 is of course Jamarium's minimum atomic weight.

Let a, b, c and d be the molecular weights of the molecules in Sample Case #2, in increasing order of molecular weight. By their atom contents, d = 2 × a and c = 2 × b. It follows from a < b that d = 2 × a < 2 × b = c, which means there is no pair of values for the atomic weights that would make the ordering strictly increasing.

In Sample Case #3, notice that the molecules happen to be sorted in strictly increasing order of total number of atoms. Therefore, assigning both elements an atomic weight of 1 makes the atomic weights be sorted in strictly increasing order.

Code Jam 2019 - Round 2

Contransmutation

Problem

Last year, we asked you to help us convert expensive metals into lead. (You do not need to know anything about the previous problem to solve this one.) But your country's leader is still greedy for more lead!

There are **M** metals known in the world; lead is metal number 1 on your periodic table. Your country's leader has asked you to use the metals in the treasury to make as much lead as possible.

For each metal (including lead), you know exactly one formula that lets you destroy one gram of that metal and create one gram each of two metals. (It is best not to think too much about the principle of mass conservation!) Note that it is possible that the formula for the i-th metal might produce the i-th metal as one of the products. The formulas do not work with partial grams. However, you can use each formula as often as you would like (or not at all), as long as you have a gram of the required ingredient.

If you make optimal choices, what is the largest number of grams of lead you can end up with, or is it unbounded? If it is not unbounded: since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 10^9+7 (that is, 1000000007).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with an integer **M**: the number of metals known in the world. Then there are **M** more lines with two integers $\mathbf{R_{i1}}$ and $\mathbf{R_{i2}}$ each; the i-th of these lines, counting starting from 1, indicates that you can destroy one gram of metal i to create one gram of metal $\mathbf{R_{i1}}$ and one gram of metal $\mathbf{R_{i2}}$. Finally, there is one line with **M** integers $\mathbf{G_1}$, $\mathbf{G_2}$, ..., $\mathbf{G_M}$; $\mathbf{G_i}$ is the number of grams of metal i in the treasury. Lead is metal 1.

Output

For each test case, output one line containing `Case #x: y` where `x` is the test case number (starting from 1). If there is no bound on the maximum amount of lead that can be produced, `y` must be `UNBOUNDED`. Otherwise, `y` must be the largest amount of lead, in grams, that you can end up with, modulo the prime 10^9+7 (that is, 1000000007).

Limits

$1 \le \mathbf{R_{i1}} < \mathbf{R_{i2}} \le \mathbf{M}$, for all i.
Time limit: 20 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{M} \le 10$.
$0 \le \mathbf{G_i} \le 10$, for all i.

Test set 2 (Hidden)

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{M} \le 100$.
$0 \le \mathbf{G_i} \le 10^9$, for all i.

Test set 3 (Hidden)

$1 \le \mathbf{T} \le 5$.
$2 \le \mathbf{M} \le 10^5$.
$0 \le \mathbf{G_i} \le 10^9$, for all i.

Sample

Sample Input

```
3
2
1 2
1 2
1 0
2
1 2
1 2
0 0
4
2 4
3 4
2 4
2 3
10 10 10 10
```

Sample Output

```
Case #1: UNBOUNDED
Case #2: 0
Case #3: 10
```

In Sample Case #1, you have one formula that turns 1 gram of lead into 1 gram of lead and 1 gram of the second metal, and another formula that turns 1 gram of the second metal into 1 gram of lead and 1 gram of the second metal. You can alternate between these formulas to produce as much of both metals as you want.

Sample Case #2 has the same formulas as Sample Case #1, but you have no metals to start with!

In Sample Case #3, none of the formulas help you produce more lead, so you cannot end up with more lead than you started with.