Code Jam 2017 - Round 3

Googlements

Problem

Chemists work with periodic table elements, but here at Code Jam, we have been using our advanced number smasher to study *googlements*. A googlement is a substance that can be represented by a string of at most nine digits. A googlement of length L must contain only decimal digits in the range 0 through L, inclusive, and it must contain at least one digit greater than 0. Leading zeroes are allowed. For example, `103` and `001` are valid googlements of length 3. `400` (which contains a digit, 4, greater than the length of the googlement, 3) and `000` (which contains no digit greater than 0) are not.

Any valid googlement can appear in the world at any time, but it will eventually decay into another googlement in a deterministic way, as follows. For a googlement of length L, count the number of `1`s in the googlement (which could be 0) and write down that value, then count the number of `2`s in the googlement (which could be 0) and write down that value to the right of the previous value, and so on, until you finally count and write down the number of Ls. The new string generated in this way represents the new googlement, and it will also have length L. It is even possible for a googlement to decay into itself!

For example, suppose that the googlement `0414` has just appeared. This has one `1`, zero `2`s, zero `3`s, and two `4`s, so it will decay into the googlement `1002`. This has one `1`, one `2`, zero `3`s, and zero `4`s, so it will decay into `1100`, which will decay into `2000`, which will decay into `0100`, which will decay into `1000`, which will continuously decay into itself.

You have just observed a googlement **G**. This googlement might have just appeared in the world, or it might be the result of one or more decay steps. What is the total number of possible googlements it could have been when it originally appeared in the world?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with a string **G**, representing a googlement.

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 number of different googlements that the observed googlement could have been when it first appeared in the world.

Limits

Memory limit: 1 GB.
1 ≤ **T** ≤ 100.
Each digit in **G** is a decimal digit between 0 and the length of **G**, inclusive.
G contains at least one non-zero digit.

Small dataset (Test Set 1 - Visible)

Time limit: 20 seconds.
1 ≤ the length of **G** ≤ 5.

Large dataset (Test Set 2 - Hidden)

Time limit: 60 seconds.
1 ≤ the length of **G** ≤ 9.

Sample

Sample Input	Sample Output
3 20 1 123	Case #1: 4 Case #2: 1 Case #3: 1

In sample case #1, the googlement could have originally been `20`, or it could have decayed from `11`, which could have itself decayed from `12` or `21`. Neither of the latter two could have been a product of decay. So there are four possibilities in total.

In sample case #2, the googlement must have originally been `1`, which is the only possible googlement of length 1.

In sample case #3, the googlement must have been `123`; no other googlement could have decayed into it.

Code Jam 2017 - Round 3

Good News and Bad News

Problem

You would like to get your **F** friends to share some news. You know your friends well, so you know which of your friends can talk to which of your other friends. There are **P** such one-way relationships, each of which is an ordered pair ($\mathbf{A_i}$, $\mathbf{B_i}$) that means that friend $\mathbf{A_i}$ can talk to friend $\mathbf{B_i}$. It does not imply that friend $\mathbf{B_i}$ can talk to friend $\mathbf{A_i}$; however, another of the ordered pairs might make that true.

For *every* such existing ordered pair ($\mathbf{A_i}$, $\mathbf{B_i}$), you want friend $\mathbf{A_i}$ to deliver some news to friend $\mathbf{B_i}$. In each case, this news will be represented by an integer value; the magnitude of the news is given by the absolute value, and the type of news (good or bad) is given by the sign. The integer cannot be 0 (or else there would be no news!), and its absolute value cannot be larger than $\mathbf{F}^2$ (or else the news would be just *too* exciting!). These integer values may be different for different ordered pairs.

Because you are considerate of your friends' feelings, for each friend, the sum of the values of all news given *by* that friend must equal the sum of values of all news given *to* that friend. If no news is given by a friend, that sum is considered to be 0; if no news is given to a friend, that sum is considered to be 0.

Can you find a set of news values for your friends to communicate such that these rules are obeyed, or determine that it is impossible?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with two integers **F** and **P**: the number of friends, and the number of different ordered pairs of friends. Then, **P** more lines follow; the i-th of these lines has two different integers $\mathbf{A_i}$ and $\mathbf{B_i}$ representing that friend $\mathbf{A_i}$ can talk to friend $\mathbf{B_i}$. Friends are numbered from 1 to **F**.

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 either `IMPOSSIBLE` if there is no arrangement satisfying the rules above, or, if there is such an arrangement, **P** integers, each of which is nonzero and lies inside [$-\mathbf{F}^2$, $\mathbf{F}^2$]. The i-th of those integers corresponds to the i-th ordered pair from the input, and represents the news value that the first friend in the ordered pair will communicate to the second. The full set of values must satisfy the conditions in the problem statement.

If there are multiple possible answers, you may output any of them.

Limits

Memory limit: 1 GB.
$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{A_i} \le \mathbf{F}$, for all i.
$1 \le \mathbf{B_i} \le \mathbf{F}$, for all i.

$\mathbf{A_i} \neq \mathbf{B_i}$, for all i. (A friend does not self-communicate.)
$(\mathbf{A_i}, \mathbf{B_i}) \neq (\mathbf{A_j}, \mathbf{B_j})$, for all i ≠ j. (No pair of friends is repeated within a test case in the same order.)

Small dataset (Test Set 1 - Visible)

Time limit: 20 seconds.
2 ≤ **F** ≤ 4.
1 ≤ **P** ≤ 12.

Large dataset (Test Set 2 - Hidden)

Time limit: 40 seconds.
2 ≤ **F** ≤ 1000.
1 ≤ **P** ≤ 2000.

Sample

Sample Input

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

Sample Output

```
Case #1: 1 1
Case #2: IMPOSSIBLE
Case #3: -1 -1 -1
Case #4: 4 -4 -4 8
Case #5: -1 1 1
```

The sample output shows one possible set of valid answers. Other valid answers are possible.

In Sample Case #1, one acceptable arrangement is to have friend 1 deliver news with value 1 to friend 2, and vice versa.

In Sample Case #2, whatever value of news friend 1 gives to friend 2, it must be nonzero. So, the sum of news values given to friend 2 is not equal to zero. However, friend 2 cannot give any news and so that value is 0. Therefore, the sums of given and received news for friend 2 cannot match, and the case is `IMPOSSIBLE`.

In Sample Case #3, each of friends 1, 2, and 3 can deliver news with value -1 to the one other friend they can talk to — an unfortunate circle of bad news! Note that there is a friend 4 who does not give or receive any news; this still obeys the rules.

In Sample Case #4, note that `-5 5 5 -10` would not have been an acceptable answer, because there are 3 friends, and |-10| > 3^2.

In Sample Case #5, note that the case cannot be solved without using at least one negative value.

Code Jam 2017 - Round 3

Mountain Tour

Problem

You are on top of Mount Everest, and you want to enjoy all the nice hiking trails that are up there. However, you know from past experience that climbing around on Mount Everest alone is bad — you might get lost in the dark! So you want to go on hikes at pre-arranged times with tour guides.

There are **C** camps on the mountain (numbered 1 through **C**), and there are 2 × **C** one-way hiking tours (numbered 1 through 2 × **C**). Each hiking tour starts at one camp and finishes at a different camp, and passes through no other camps in between. Mount Everest is sparsely populated, and business is slow; there are exactly 2 hiking tours departing from each camp, and exactly 2 hiking tours arriving at each camp.

Each hiking tour runs daily. Tours 1 and 2 start at camp 1, tours 3 and 4 start at camp 2, and so on: in general, tour 2 × i - 1 and tour 2 × i start at camp i. The i-th hiking tour ends at camp number $\mathbf{E_i}$, leaves at hour $\mathbf{L_i}$, and has a duration of exactly $\mathbf{D_i}$ hours.

It is currently hour 0; the hours in a day are numbered 0 through 23. You are at camp number 1, and you want to do each of the hiking tours *exactly once* and end up back at camp number 1. You cannot travel between camps except via hiking tours. While you are in a camp, you may wait for any number of hours (including zero) before going on a hiking tour, but you can only start a hiking tour at the instant that it departs.

After looking at the tour schedules, you have determined that it is definitely possible to achieve your goal, but you want to do it as fast as possible. If you plan your route optimally, how many hours will it take you to finish all of the tours?

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 **C**: the number of camps. Then, 2 × **C** more lines follow. The i-th of these lines (counting starting from 1) represents one hiking tour starting at camp number floor((i + 1) / 2), and contains three integers $\mathbf{E_i}$, $\mathbf{L_i}$, and $\mathbf{D_i}$, as described above. Note that this format guarantees that exactly two tours start at each camp.

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 minimum number of hours that it will take you to achieve your goal, as described above.

Limits

Time limit: 20 seconds per test set.
Memory limit: 1 GB.
1 ≤ **T** ≤ 100.
1 ≤ $\mathbf{E_i}$ ≤ **C**.
$\mathbf{E_i}$ ≠ ceiling(i / 2), for all i. (No hiking tour starts and ends at the same camp.)
size of {j : $\mathbf{E_j}$ = i} = 2, for all j. (Exactly two tours end at each camp.)

$0 \le \mathbf{L_i} \le 23$.
$1 \le \mathbf{D_i} \le 1000$.
There is at least one route that starts and ends at camp 1 and includes each hiking tour exactly once.

Small dataset (Test Set 1 - Visible)

$2 \le \mathbf{C} \le 15$.

Large dataset (Test Set 2 - Hidden)

$2 \le \mathbf{C} \le 1000$.

Sample

Sample Input

```
2
2
2 1 5
2 0 3
1 4 4
1 6 3
4
3 0 24
2 0 24
4 0 24
4 0 24
2 0 24
1 0 24
3 0 24
1 0 24
```

Sample Output

```
Case #1: 32
Case #2: 192
```

In sample case #1, the optimal plan is as follows:

- Wait at camp 1 for an hour, until it becomes hour 1.
- Leave camp 1 at hour 1 to take the 5 hour hiking tour; arrive at camp 2 at hour 6.
- Immediately leave camp 2 at hour 6 to take the 3 hour hiking tour; arrive at camp 1 at hour 9.
- Wait at camp 1 for 15 hours, until it becomes hour 0 of the next day.
- Leave camp 1 at hour 0 to take the 3 hour hiking tour; arrive at camp 2 at hour 3.
- Wait at camp 2 for 1 hour, until it becomes hour 4.
- Leave camp 2 at hour 4 to take the 4 hour hiking tour; arrive at camp 1 at hour 8.

This achieves the goal in 1 day and 8 hours, or 32 hours. Any other plan takes longer.

In sample case #2, all of the tours leave at the same time and are the same duration. After finishing any tour, you can immediately take another tour. If we number the tours from 1 to 8 in the order in which they appear in the test case, one optimal plan is: 1, 5, 4, 7, 6, 2, 3, 8.

Code Jam 2017 - Round 3

Slate Modern

Problem

The prestigious Slate Modern gallery specializes in the latest art craze: grayscale paintings that follow very strict rules. Any painting in the gallery must be a grid with **R** rows and **C** columns. Each cell in the grid is painted with a color of a certain positive integer *brightness value*; to make sure the art is not too visually startling, the brightness values of any two cells that share an edge (not just a corner) must differ by no more than **D** units.

Your artist friend Cody-Jamal is working on a canvas for the gallery. Last night, he became inspired and filled in **N** different particular cells with certain positive integer brightness values. You just told him about the gallery's rules today, and now he wants to know whether it is possible to fill in all of the remaining cells with positive integer brightness values and complete the painting without breaking the gallery's rules. If this is possible, he wants to make the sum of the brightness values as large as possible, to save his black paint. Can you help him find this sum or determine that the task is impossible? 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 (1000000007).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case begins with one line with four integers: **R**, **C**, **N**, and **D**, as described above. Then, **N** lines follow; the i-th of these has three integers $\mathbf{R_i}$, $\mathbf{C_i}$, and $\mathbf{B_i}$, indicating that the cell in the $\mathbf{R_i}$th row and $\mathbf{C_i}$th column of the grid has brightness value $\mathbf{B_i}$. The rows and columns of the grid are numbered starting from 1.

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 either `IMPOSSIBLE` if it is impossible to complete the picture, or else the value of the maximum possible sum of all brightness values modulo the prime 10^9+7 (1000000007).

Limits

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 200$.
$1 \leq \mathbf{D} \leq 10^9$.
$1 \leq \mathbf{R_i} \leq \mathbf{R}$, for all i. $1 \leq \mathbf{C_i} \leq \mathbf{C}$, for all i. $1 \leq \mathbf{B_i} \leq 10^9$, for all i. (Note that the upper bound only applies to cells that Cody-Jamal already painted. You can assign brightness values larger than 10^9 to other cells.)
$\mathbf{N} < \mathbf{R} \times \mathbf{C}$. (There is at least one empty cell.)
$\mathbf{R_i} \neq \mathbf{R_j}$ and/or $\mathbf{C_i} \neq \mathbf{C_j}$ for all $i \neq j$. (All of the given cells are different cells in the grid.)

Small dataset (Test Set 1 - Visible)

Time limit: 40 seconds.
$1 \le \mathbf{R} \le 200$.
$1 \le \mathbf{C} \le 200$.

Large dataset (Test Set 2 - Hidden)

Time limit: 80 seconds.
$1 \le \mathbf{R} \le 10^9$.
$1 \le \mathbf{C} \le 10^9$.

Sample

Sample Input

```
4
2 3 2 2
2 1 4
1 2 7
1 2 1 1000000000
1 2 1000000000
3 1 2 100
1 1 1
3 1 202
2 2 2 2
2 1 1
2 2 4
```

Sample Output

```
Case #1: 40
Case #2: 999999986
Case #3: IMPOSSIBLE
Case #4: IMPOSSIBLE
```

In Sample Case #1, the optimal way to finish the painting is:

```
6 7 9
4 6 8
```

and the sum is 40.

In Sample Case #2, the optimal way to finish the painting is:

```
2000000000 1000000000
```

and the sum is 3000000000; modulo 10^9+7, it is 999999986.

In Sample Case #3, the task is impossible. No matter what value you choose for the cell in row 2, it will be too different from at least one of the two neighboring filled-in cells.

In Sample Case #4, the two cells that Cody-Jamal filled in already have brightness values that are too far apart, so it is impossible to continue.