Problem A. Oversized Pancake Flipper

Small input 5 points	Solve A-small
Large input 10 points	Solve A-large

Problem

Last year, the Infinite House of Pancakes introduced a new kind of pancake. It has a happy face made of chocolate chips on one side (the "happy side"), and nothing on the other side (the "blank side").

You are the head cook on duty. The pancakes are cooked in a single row over a hot surface. As part of its infinite efforts to maximize efficiency, the House has recently given you an oversized pancake flipper that flips exactly **K** consecutive pancakes. That is, in that range of **K** pancakes, it changes every happy-side pancake to a blank-side pancake, and vice versa; it does *not* change the left-to-right order of those pancakes.

You cannot flip fewer than **K** pancakes at a time with the flipper, even at the ends of the row (since there are raised borders on both sides of the cooking surface). For example, you can flip the first **K** pancakes, but not the first **K** - 1 pancakes.

Your apprentice cook, who is still learning the job, just used the old-fashioned single-pancake flipper to flip some individual pancakes and then ran to the restroom with it, right before the time when customers come to visit the kitchen. You only have the oversized pancake flipper left, and you need to use it quickly to leave all the cooking pancakes happy side up, so that the customers leave feeling happy with their visit.

Given the current state of the pancakes, calculate the minimum number of uses of the oversized pancake flipper needed to leave all pancakes happy side up, or state that there is no way to do it.

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 **S** and an integer **K**. **S** represents the row of pancakes: each of its characters is either + (which represents a pancake that is initially happy side up) or - (which represents a pancake that is initially blank side up).

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 way to get all the pancakes happy side up, or an integer representing the the minimum number of times you will need to use the oversized pancake flipper to do it.

Limits

1 ≤ **T** ≤ 100.
Every character in **S** is either + or -.
2 ≤ **K** ≤ length of **S**.

Small dataset

2 ≤ length of **S** ≤ 10.

Large dataset

2 ≤ length of **S** ≤ 1000.

Sample

Input	Output
3 ---+-++- 3 +++++ 4 -+-+- 4	Case #1: 3 Case #2: 0 Case #3: IMPOSSIBLE

In Case #1, you can get all the pancakes happy side up by first flipping the leftmost 3 pancakes, getting to ++++-++-, then the rightmost 3, getting to ++++---+, and finally the 3 pancakes that remain blank side up. There are other ways to do it with 3 flips or more, but none with fewer than 3 flips.

In Case #2, all of the pancakes are already happy side up, so there is no need to flip any of them.

In Case #3, there is no way to make the second and third pancakes from the left have the same side up, because any flip flips them both. Therefore, there is no way to make all of the pancakes happy side up.

Problem B. Tidy Numbers

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 5 points	Solve B-small
Large input 15 points	Solve B-large

Problem

Tatiana likes to keep things tidy. Her toys are sorted from smallest to largest, her pencils are sorted from shortest to longest and her computers from oldest to newest. One day, when practicing her counting skills, she noticed that some integers, when written in base 10 with no leading zeroes, have their digits sorted in non-decreasing order. Some examples of this are 8, 123, 555, and 224488. She decided to call these numbers *tidy*. Numbers that do not have this property, like 20, 321, 495 and 999990, are not tidy.

She just finished counting *all* positive integers in ascending order from 1 to **N**. What was the last tidy number she counted?

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line describes a test case with a single integer **N**, the last number counted by Tatiana.

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 last tidy number counted by Tatiana.

Limits

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

Small dataset

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

Large dataset

$1 \leq \mathbf{N} \leq 10^{18}$.

Sample

Input	Output
4 132 1000 7 11111111111111110	Case #1: 129 Case #2: 999 Case #3: 7 Case #4: 99999999999999999

Note that the last sample case would not appear in the Small dataset.

Problem C. Bathroom Stalls

Small input 1 5 points	Solve C-small-1
Small input 2 10 points	Solve C-small-2
Large input 15 points	Solve C-large

Problem

A certain bathroom has **N** + 2 stalls in a single row; the stalls on the left and right ends are permanently occupied by the bathroom guards. The other **N** stalls are for users.

Whenever someone enters the bathroom, they try to choose a stall that is as far from other people as possible. To avoid confusion, they follow deterministic rules: For each empty stall S, they compute two values L_S and R_S, each of which is the number of empty stalls between S and the closest occupied stall to the left or right, respectively. Then they consider the set of stalls with the farthest closest neighbor, that is, those S for which $\min(L_S, R_S)$ is maximal. If there is only one such stall, they choose it; otherwise, they choose the one among those where $\max(L_S, R_S)$ is maximal. If there are still multiple tied stalls, they choose the leftmost stall among those.

K people are about to enter the bathroom; each one will choose their stall before the next arrives. Nobody will ever leave.

When the last person chooses their stall S, what will the values of $\max(L_S, R_S)$ and $\min(L_S, R_S)$ be?

Solving this problem

This problem has 2 Small datasets and 1 Large dataset. You must solve the first Small dataset before you can attempt the second Small dataset. You will be able to retry either of the Small datasets (with a time penalty). You will be able to make a single attempt at the Large, as usual, only after solving both Small datasets.

Input

The first line of the input gives the number of test cases, **T**. **T** lines follow. Each line describes a test case with two integers **N** and **K**, as described above.

Output

For each test case, output one line containing `Case #x: y z`, where x is the test case number (starting from 1), y is $\max(L_S, R_S)$, and z is $\min(L_S, R_S)$ as calculated by the last person to enter the bathroom for their chosen stall S.

Limits

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

Small dataset 1

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

Small dataset 2

$1 \le \mathbf{N} \le 10^6$.

Large dataset

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

Sample

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

```
1000 1000    Case #5: 500 499
1000 1
```

In Case #1, the first person occupies the leftmost of the middle two stalls, leaving the following configuration (0 stands for an occupied stall and `.` for an empty one): `0.0..0`. Then, the second and last person occupies the stall immediately to the right, leaving 1 empty stall on one side and none on the other.

In Case #2, the first person occupies the middle stall, getting to `0..0..0`. Then, the second and last person occupies the leftmost stall.

In Case #3, the first person occupies the leftmost of the two middle stalls, leaving `0..0...0`. The second person then occupies the middle of the three consecutive empty stalls.

In Case #4, every stall is occupied at the end, no matter what the stall choices are.

In Case #5, the first and only person chooses the leftmost middle stall.

Problem D. Fashion Show

Small input 10 points	Solve D-small
Large input 25 points	Solve D-large

Problem

You are about to host a fashion show to show off three new styles of clothing. The show will be held on a stage which is in the most fashionable of all shapes: an **N**-by-**N** grid of cells.

Each cell in the grid can be empty (which we represent with a `.` character) or can contain one fashion model. The models come in three types, depending on the clothing style they are wearing: `+`, `x`, and the super-trendy `o`. A cell with a `+` or `x` model in it adds 1 *style point* to the show. A cell with an `o` model in it adds 2 style points. Empty cells add no style points.

To achieve the maximum artistic effect, there are rules on how models can be placed relative to each other.

- Whenever any two models share a row or column, at least one of the two must be a `+`.
- Whenever any two models share a diagonal of the grid, at least one of the two must be an `x`.

Formally, a model located in row i_0 and column j_0 and a model located in row i_1 and column j_1 share a row if and only if $i_0 = i_1$, they share a column if and only if $j_0 = j_1$, and they share a diagonal if and only if $i_0 + j_0 = i_1 + j_1$ or $i_0 - j_0 = i_1 - j_1$.

For example, the following grid is not legal:

```
...
x+o
.+.
```

The middle row has a pair of models (`x` and `o`) that does not include a `+`. The diagonal starting at the `+` in the bottom row and running up to the `o` in the middle row has two models, and neither of them is an `x`.

However, the following grid is legal. No row, column, or diagonal violates the rules.

```
+.x
+x+
o..
```

Your artistic advisor has already placed **M** models in certain cells, following these rules. You are free to place any number (including zero) of additional models of whichever types you like. You may not remove existing models, but you may upgrade as many existing `+` and `x` models into `o` models as you wish, as long as the above rules are not violated.

Your task is to find a legal way of placing and/or upgrading models that earns the maximum possible number of style points.

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 two integers **N** and **M**, as described above. Then, **M** more lines follow; the i-th of these lines has a `+`, `x`, or `o` character (the type of the model) and two integers $\mathbf{R_i}$ and $\mathbf{C_i}$ (the position of the model). The rows of the grid are numbered 1 through **N**, from top to bottom. The columns of the grid are numbered 1 through **N**, from left to right.

Output

For each test case, first output one line containing `Case #x: y z`, where `x` is the test case number (starting from 1), `y` is the number of style points earned in your arrangement, and `z` is the total number of models you have added and/or substituted in. Then, for each model that you have added or substituted in, output exactly one line in exactly the same format described in the Input section, where the character is the type of the model that you have added or substituted in. These `z` lines can be in any order.

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

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 100$.
$1 \leq \mathbf{C_i} \leq \mathbf{N}$, for all i.
$0 \leq \mathbf{M} \leq \mathbf{N}^2$.
No two pre-placed models appear in the same cell.
It is guaranteed that the set of pre-placed models follows the rules.

Small dataset

$\mathbf{R_i} = 1$, for all i. (Any models that are pre-placed are in the top row. Note that you may add/replace models in that row and/or add models in other rows.)

Large dataset

$1 \leq \mathbf{R_i} \leq \mathbf{N}$, for all i.

Sample

Input	Output
3 2 0 1 1 o 1 1 3 4 + 2 3 + 2 1 x 3 1 + 2 2	Case #1: 4 3 o 2 2 + 2 1 x 1 1 Case #2: 2 0 Case #3: 6 2 o 2 3 x 1 2

The sample output displays one set of answers to the sample cases. Other answers may be possible. Note that the last sample case would not appear in the Small dataset.

In sample case #1, the grid is 2-by-2 and is initially blank. The output corresponds to the following grid. (In these explanations, we will use `.` to denote a blank cell.)

```
x.
+o
```

In sample case #2, the only cell is already occupied by an o model, and it is impossible to add a new model or replace the o model.

In sample case #3, the grid looks like this before you place any models:

```
...
+++
x..
```

The output corresponds to this grid:

```
.x.
++o
x..
```

Problem A. Alphabet Cake

Small input 8 points	Solve A-small
Large input 13 points	Solve A-large

Problem

You are catering a party for some children, and you are serving them a cake in the shape of a grid with **R** rows and **C** columns. Your assistant has started to decorate the cake by writing every child's initial in icing on exactly one cell of the cake. Each cell contains at most one initial, and since no two children share the same initial, no initial appears more than once on the cake.

Each child wants a single rectangular (grid-aligned) piece of cake that has their initial and no other child's initial(s). Can you find a way to assign every blank cell of the cake to one child, such that this goal is accomplished? It is guaranteed that this is always possible. There is no need to split the cake evenly among the children, and one or more of them may even get a 1-by-1 piece; this will be a valuable life lesson about unfairness.

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 **R** and **C**. Then, there are **R** more lines of **C** characters each, representing the cake. Each character is either an uppercase English letter (which means that your assistant has already added that letter to that cell) or ? (which means that that cell is blank).

Output

For each test case, output one line containing `Case #x:` and nothing else. Then output **R** more lines of **C** characters each. Your output grid must be identical to the input grid, but with *every* ? replaced with an uppercase English letter, representing that that cell appears in the slice for the child who has that initial. You may not add letters that did not originally appear in the input. In your grid, for each letter, the region formed by all the cells containing that letter must be a single grid-aligned rectangle.

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

Limits

$1 \leq \mathbf{T} \leq 100$.
There is at least one letter in the input grid.
No letter appears in more than one cell in the input grid.
It is guaranteed that at least one answer exists for each test case.

Small dataset

$1 \leq \mathbf{R} \leq 12$.
$1 \leq \mathbf{C} \leq 12$.
$\mathbf{R} \times \mathbf{C} \leq 12$.

Large dataset

$1 \leq \mathbf{R} \leq 25$.
$1 \leq \mathbf{C} \leq 25$.

Sample

Input	Output
3 3 3 G?? ?C? ??J 3 4 CODE ???? ?JAM 2 2 CA KE	Case #1: GGJ CCJ CCJ Case #2: CODE COAE JJAM Case #3: CA KE

The sample output displays one set of answers to the sample cases. Other answers may be possible.

Round 1A 2017

A. Alphabet Cake
B. Ratatouille
C. Play the Dragon

Contest Analysis
Questions asked 1

Submissions

Alphabet Cake
8pt | Not attempted 4837/5898 users correct (82%)
13pt | Not attempted 4296/4790 users correct (90%)

Ratatouille
12pt | Not attempted 1939/2782 users correct (70%)
23pt | Not attempted 1337/1709 users correct (78%)

Play the Dragon
19pt | Not attempted 723/1359 users correct (53%)
25pt | Not attempted 8/124 users correct (6%)

Top Scores

Eryx	100
pperm	100
xyz111	100
Nore	100
kmjp	100
mk.al13n	100
Rafbill	77
johngs	75
burunduk3	75
Errichto.rekt	75

Problem B. Ratatouille

Small input 12 points	Solve B-small
Large input 23 points	Solve B-large

Problem

You've discovered it: the ultimate recipe for ratatouille, the famous French dish! You know which ingredients to use, and how many grams of each one to use, in order to make one serving of ratatouille. But you believe that anyone can cook, and so you want to share the recipe with the world... and make some money in the process!

You have ordered some ingredient packages that are easy to ship. Each package contains some amount of one ingredient; different packages may have different amounts even if they contain the same ingredient. For convenience, you ordered the same number of packages of each ingredient.

You would like to use these packages to form as many ratatouille *kits* as possible to send to customers. A kit consists of *exactly one* package of each ingredient, and a label with the integer number of servings of ratatouille that the kit makes. Since you do not want to shortchange customers or waste food, each package must contain between 90 and 110 percent (inclusive) of the amount of that ingredient that is actually needed to make the number of servings of ratatouille on the kit's label.

For example, suppose that one serving of ratatouille takes 500 g of tomato and 300 g of onion. Suppose that you have a 900 g package of tomato and a 660 g package of onion. You could form these into a kit that makes two servings of ratatouille. To make two servings, 1000 g of tomato and 600 g of onion are required. Since the 900 g of tomato you have is within [90, 110]% of the 1000 g of tomato required, and the 660 g of onion you have is within [90, 110]% of the 600 g of onion required, this is acceptable. However, you could not say that the kit makes one or three servings of ratatouille, nor could you say that it makes 1.999 servings (the number of servings must be an integer).

Note that there are some sets of packages that could never form a kit. Continuing with our recipe above, if you have a 1500 g package of tomato and an 809 g package of onion, for example, there is no amount of servings that you can make. Three servings would take 1500 g of tomato and 900 g of onion, and the amount of onion is not within the [90, 110]% range. No other integer amount of servings works, either.

You want to share your recipe with as many customers as possible, so you want to produce the maximum number of valid kits. (Of course, each package can be used in at most one kit.) What is the largest number of kits that you can form? Note that you are *not* required to maximize the total number of servings of ratatouille formed.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case consists of the following:

- One line with two integers **N**: the number of ingredients, and **P**, the number of packages of each ingredient.
- One line with **N** integers $\mathbf{R_i}$. The i-th of these represents the number of grams of the i-th ingredient needed to make one serving of ratatouille.
- **N** more lines of **P** integers each. The j-th value on the i-th of these lines, $\mathbf{Q_{ij}}$, represents the quantity, in grams, in the j-th package of the i-th ingredient.

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 maximum number of kits you can produce, as described above.

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{R_i} \leq 10^6$, for all i.
$1 \leq \mathbf{Q_{ij}} \leq 10^6$, for all i and j.

Small dataset

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

$1 \leq \mathbf{P} \leq 8$.

Large dataset

$1 \leq \mathbf{N} \leq 50$.
$1 \leq \mathbf{P} \leq 50$.
$\mathbf{N} \times \mathbf{P} \leq 1000$.

Sample

Input	Output
6 2 1 500 300 900 660 2 1 500 300 1500 809 2 2 50 100 450 449 1100 1101 2 1 500 300 300 500 1 8 10 11 13 17 11 16 14 12 18 3 3 70 80 90 1260 1500 700 800 1440 1600 1700 1620 900	Case #1: 1 Case #2: 0 Case #3: 1 Case #4: 0 Case #5: 3 Case #6: 3

Note that the last sample case would not appear in the Small dataset.

Sample cases #1 and #2 are the ones described in the problem statement.

In sample case #3, you can form a kit out of the 450 g package of the first ingredient and the 1100 g package of the second ingredient, and say that the kit makes 10 servings of ratatouille. That number of servings requires 500 g of the first ingredient; you have 450 g, which is 90% of 500 and within the allowed limit. It requires 1000 g of the second ingredient; you have 1100 g, which is 110% of 1000 and within the allowed limit.

Once you form this kit, however, you cannot form the remaining packages into a kit. 449 g of the first ingredient and 1101 g of the second ingredient would not be able to form 10 (or any other number of) servings. In fact, the (450 g, 1100 g) kit is the only kit that can be formed from these packages.

In sample case #4, no kits can be formed. Note that the recipe requires particular amounts of particular ingredients *in the given order*; the ingredients are not interchangeable. This is fine French cuisine, after all!

In sample case #5, the recipe has only one ingredient — how elegantly simple! A single serving cannot use more than 11 g, and two servings cannot use fewer than 18 g. It is possible to form three kits: two with an 11 g package, and one with an 18 g package.

In sample case #6, you can form three valid kits: (700 g, 800 g, 900 g), which makes 10 servings, and (1500 g, 1600 g, 1700 g) and (1260 g, 1440 g, 1620 g), each of which makes 20 servings. Note that you could also say that the (1260 g, 1440 g, 1620 g) kit makes 17, 18, or 19 servings, but it does not matter how many servings a kit makes as long as the kit is valid.

Problem C. Play the Dragon

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 19 points	Solve C-small
Large input 25 points	Solve C-large

Problem

You are a friendly dragon fighting to protect your lair from a greedy knight! You have $\mathbf{H_d}$ health points and an attack power of $\mathbf{A_d}$, and the knight has $\mathbf{H_k}$ health points and an attack power of $\mathbf{A_k}$. If your health drops to 0 or below at any point; you are knocked out and you instantly lose; if the knight's health drops to 0 or below at any point, the knight is knocked out and you win!

You will battle the knight in a series of turns. On each turn, you go first, and you can choose and execute any one of the following actions.

- Attack: Reduce the opponent's health by your own attack power.
- Buff: Increase your attack power by **B** for the rest of the battle.
- Cure: Your health becomes $\mathbf{H_d}$.
- Debuff: Decrease the opponent's attack power by **D** for the rest of the battle. If a Debuff would cause the opponent's attack power to become less than 0, it instead sets it to 0.

Then, if the knight's health is greater than 0 following your action, the knight will execute an Attack action. After that, the turn ends. (Note that a turn in which you defeat the knight still counts as a turn even though the knight does not get to act.)

Note that buffs stack with each other; every buff adds an additional **B** to your attack power. Similarly, debuffs stack with each other.

You would like to defeat the knight as fast as possible (if it is possible) so that you will not be late to help the villagers roast marshmallows at tonight's festival. Can you determine the minimum number of turns in which you can defeat the knight, or that it is `IMPOSSIBLE` to do so?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with six integers $\mathbf{H_d}$, $\mathbf{A_d}$, $\mathbf{H_k}$, $\mathbf{A_k}$, **B**, and **D**, as described above.

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 not possible to defeat the knight, or the minimum number of turns needed to defeat the knight.

Limits

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

Small dataset

$1 \leq \mathbf{H_d} \leq 100$.
$1 \leq \mathbf{A_d} \leq 100$.
$1 \leq \mathbf{H_k} \leq 100$.
$1 \leq \mathbf{A_k} \leq 100$.
$0 \leq \mathbf{B} \leq 100$.
$0 \leq \mathbf{D} \leq 100$.

Large dataset

$1 \leq \mathbf{H_d} \leq 10^9$.
$1 \leq \mathbf{A_d} \leq 10^9$.
$1 \leq \mathbf{H_k} \leq 10^9$.
$1 \leq \mathbf{A_k} \leq 10^9$.
$0 \leq \mathbf{B} \leq 10^9$.
$0 \leq \mathbf{D} \leq 10^9$.

Sample

Input	Output
4 11 5 16 5 0 0 3 1 3 2 2 0 3 1 3 2 1 0 2 1 5 1 1 1	Case #1: 5 Case #2: 2 Case #3: IMPOSSIBLE Case #4: 5

In Case #1, you have 11 health and 5 attack, and the knight has 16 health and 5 attack. One possible optimal sequence of actions is:

- Turn 1: Attack, reducing the knight's health to 11. Then the knight attacks and reduces your health to 6.
- Turn 2: Attack, reducing the knight's health to 6. Then the knight attacks and reduces your health to 1.
- Turn 3: Cure, restoring your health to 11. Then the knight attacks and reduces your health to 6. (If you had attacked instead this turn, the knight's next attack would have caused you to lose.)
- Turn 4: Attack, reducing the knight's health to 1. Then the knight attacks and reduces your health to 1.
- Turn 5: Attack, reducing the knight's health to -4. You instantly win and the knight does not get another attack.

In Case #2, one possible optimal sequence of actions is:

- Turn 1: Buff, increasing your attack power to 3. Then the knight attacks and reduces your health to 1.
- Turn 2: Attack, reducing the knight's health to 0. You instantly win and the knight does not get another attack.

In Case #3, the knight only needs two attacks to defeat you, and you cannot do enough damage fast enough to defeat the knight. You can indefinitely extend the combat by executing the Cure action after every attack, but it is impossible to actually defeat the knight.

In Case #4, one possible optimal sequence of actions is: Attack, Debuff, Buff, Attack, Attack.

Round 1B 2017

A. Steed 2: Cruise Control

B. Stable Neigh-bors

C. Pony Express

Contest Analysis

Questions asked

Problem A. Steed 2: Cruise Control

Small input 11 points	Solve A-small
Large input 14 points	Solve A-large

Problem

Annie is a bus driver with a high-stress job. She tried to unwind by going on a Caribbean cruise, but that also turned out to be stressful, so she has recently taken up horseback riding.

Today, Annie is riding her horse to the east along a long and narrow one-way road that runs west to east. She is currently at kilometer 0 of the road, and her destination is at kilometer **D**; kilometers along the road are numbered from west to east.

There are **N** other horses traveling east on the same road; all of them will go on traveling forever, and all of them are currently between Annie's horse and her destination. The i-th of these horses is initially at kilometer $\mathbf{K_i}$ and is traveling at its maximum speed of $\mathbf{S_i}$ kilometers per hour.

Horses are very polite, and a horse H_1 will not pass (move ahead of) another horse H_2 that started off ahead of H_1. (Two or more horses can share the same position for any amount of time; you may consider the horses to be single points.) Horses (other than Annie's) travel at their maximum speeds, except that whenever a horse H_1 catches up to another slower horse H_2, H_1 reduces its speed to match the speed of H_2.

Annie's horse, on the other hand, does not have a maximum speed and can travel at any speed that Annie chooses, as long as it does not pass another horse. To ensure a smooth ride for her and her horse, Annie wants to choose a single constant "cruise control" speed for her horse for the entire trip, from her current position to the destination, such that her horse will not pass any other horses. What is the maximum such speed that she can choose?

Input

The first line of the input gives the number of test cases, **T**; **T** test cases follow. Each test case begins with two integers **D** and **N**: the destination position of all of the horses (in kilometers) and the number of other horses on the road. Then, **N** lines follow. The i-th of those lines has two integers $\mathbf{K_i}$ and $\mathbf{S_i}$: the initial position (in kilometers) and maximum speed (in kilometers per hour) of the i-th of the other horses on the road.

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 maximum constant speed (in kilometers per hour) that Annie can use without colliding with other horses. y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$0 < \mathbf{K_i} < \mathbf{D} \le 10^9$, for all i.
$\mathbf{K_i} \ne \mathbf{K_j}$, for all $i \ne j$. (No two horses start in the same position.)
$1 \le \mathbf{S_i} \le 10000$.

Small dataset

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

Large dataset

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

Sample

Input	Output
3 2525 1	Case #1: 101.000000 Case #2: 100.000000

```
2400 5    Case #3: 33.333333
300 2
120 60
60 90
100 2
80 100
70 10
```

In sample case #1, there is one other (very slow!) horse on the road; it will reach Annie's destination after 25 hours. Anything faster than 101 kilometers per hour would cause Annie to pass the horse before reaching the destination.

In sample case #2, there are two other horses on the road. The faster horse will catch up to the slower horse at kilometer 240 after 2 hours. Both horses will then go at the slower horse's speed for 1 more hour, until the horses reach Annie's destination at kilometer 300. The maximum speed that Annie can choose without passing another horse is 100 kilometers per hour.

Round 1B 2017

A. Steed 2: Cruise Control
B. Stable Neigh-bors
C. Pony Express

Contest Analysis
Questions asked

Problem B. Stable Neigh-bors

Small input 13 points	Solve B-small
Large input 22 points	Solve B-large

Problem

You are lucky enough to own **N** pet unicorns. Each of your unicorns has either one or two of the following kinds of hairs in its mane: red hairs, yellow hairs, and blue hairs. The color of a mane depends on exactly which sorts of colored hairs it contains:

- A mane with only one color of hair appears to be that color. For example, a mane with only blue hairs is blue.
- A mane with red and yellow hairs appears orange.
- A mane with yellow and blue hairs appears green.
- A mane with red and blue hairs appears violet.

You have **R**, **O**, **Y**, **G**, **B**, and **V** unicorns with red, orange, yellow, green, blue, and violet manes, respectively.

You have just built a circular stable with **N** stalls, arranged in a ring such that each stall borders two other stalls. You would like to put exactly one of your unicorns in each of these stalls. However, unicorns need to feel rare and special, so no unicorn can be next to another unicorn that shares at least one of the hair colors in its mane. For example, a unicorn with an orange mane cannot be next to a unicorn with a violet mane, since both of those manes have red hairs. Similarly, a unicorn with a green mane cannot be next to a unicorn with a yellow mane, since both of those have yellow hairs.

Is it possible to place all of your unicorns? If so, provide any one arrangement.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with seven integers: **N**, **R**, **O**, **Y**, **G**, **B**, and **V**.

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` if it is not possible to place all the unicorns, or a string of **N** characters representing the placements of unicorns in stalls, starting at a point of your choice and reading clockwise around the circle. Use `R` to represent each unicorn with a red mane, `O` to represent each unicorn with an orange mane, and so on with `Y`, `G`, `B`, and `V`. This arrangement must obey the rules described in the statement above.

If multiple arrangements are possible, you may print any of them.

Limits

$1 \leq \mathbf{T} \leq 100$.
$3 \leq \mathbf{N} \leq 1000$.
$\mathbf{R} + \mathbf{O} + \mathbf{Y} + \mathbf{G} + \mathbf{B} + \mathbf{V} = \mathbf{N}$.
$0 \leq Z$ for each Z in {**R**, **O**, **Y**, **G**, **B**, **V**}.

Small dataset

$\mathbf{O} = \mathbf{G} = \mathbf{V} = 0$. (Each unicorn has only one hair color in its mane.)

Large dataset

No restrictions beyond the general limits. (Each unicorn may have either one or two hair colors in its mane.)

Sample

Input	Output
4 6 2 0 2 0 2 0 3 1 0 2 0 0 0 6 2 0 1 1 2 0 4 0 0 2 0 0 2	Case #1: RYBRBY Case #2: IMPOSSIBLE Case #3: YBRGRB Case #4: YVYV

Note that the last two sample cases would not appear in the Small dataset.

For sample case #1, there are many possible answers; for example, another is BYBRYR. Note that BYRYRB would *not* be a valid answer; remember that the stalls form a ring, and the first touches the last!

In sample case #2, there are only three stalls, and each stall is a neighbor of the other two, so the two unicorns with yellow manes would have to be neighbors, which is not allowed.

For sample case #3, note that arranging the unicorns in the same color pattern as the Google logo (BRYBGR) would not be valid, since a unicorn with a blue mane would be a neighbor of a unicorn with a green mane, and both of those manes share blue hairs.

In sample case #4, no two unicorns with yellow manes can be neighbors, and no two unicorns with violet manes can be neighbors.

Problem C. Pony Express

Small input 16 points	Solve C-small
Large input 24 points	Solve C-large

Problem

It's the year 1860, and the Pony Express is the fastest mail delivery system joining the East and West coasts of the United States. This system serves **N** different cities. In each city, there is one horse (as in the expression "one-horse town"); each horse travels at a certain constant speed and has a maximum total distance it can travel before it becomes too tired to continue.

The Pony Express rider starts off on the starting city's horse. Every time the rider reaches a city, they may continue to use their current horse or switch to that city's horse; switching is instantaneous. Horses never get a chance to rest, so whenever part of a horse's maximum total distance is "used up", it is used up forever! When the rider reaches the destination city, the mail is delivered.

The routes between cities were established via complicated negotiations between company owners, lawmakers, union delegates, and cousin Pete. That means that the distances between cities do not necessarily follow common sense: for instance, they do not necessarily comply with the triangle inequality, and the distance from city A to city B might be different from the distance from city B to city A!

You are a time traveling entrepreneur, and you have brought a fast computer from the future. A single computer is not enough for you to set up an e-mail service and make the Pony Express obsolete, but you can use it to make optimal routing plans for the Pony Express. Given all data about routes between cities and the horses in each city, and a list of pairs of starting and ending cities, can you quickly calculate the minimum time necessary for each delivery? (You should treat all of these deliveries as independent; using cities/horses on one route does not make them unavailable on other routes.)

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case is described as follows:

- One line with two integers: **N**, the number of cities with horses, and **Q**, the number of pairs of stops we are interested in. Cities are numbered from 1 to **N**.
- **N** lines, each containing two integers $\mathbf{E}_i$, the maximum total distance, in kilometers, the horse in the i-th city can go and $\mathbf{S}_i$, the constant speed, in kilometers per hour, at which the horse travels.
- **N** lines, each containing **N** integers. The j-th integer on the i-th of these lines, $\mathbf{D}_{ij}$, is -1 if there is no direct route from the i-th to the j-th city, and the length of that route in kilometers otherwise.
- **Q** lines containing two integers $\mathbf{U}_k$ and $\mathbf{V}_k$, the starting and destination point, respectively, of the k-th pair of cities we want to investigate.

Output

For each test case, output one line containing `Case #x:` y_1 y_2 `...` $y_\mathbf{Q}$, where x is the test case number (starting from 1) and y_k is the minimum time, in hours, to deliver a letter from city $\mathbf{U}_k$ to city $\mathbf{V}_k$.

Each y_k will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 100$.
$1 \le \mathbf{E}_i \le 10^9$, for all i.
$1 \le \mathbf{S}_i \le 1000$, for all i.
$-1 \le \mathbf{D}_{ij} \le 10^9$, for all i, j.
$\mathbf{D}_{ii} = -1$, for all i. (There are no direct routes from a city to itself.)
$\mathbf{D}_{ij} \ne 0$, for all i, j.
$\mathbf{U}_k \ne \mathbf{V}_k$, for all k.
It is guaranteed that the delivery from $\mathbf{U}_k$ to $\mathbf{V}_k$ can be accomplished with the given horses, for all k.
$\mathbf{U}_l \ne \mathbf{U}_m$ and/or $\mathbf{V}_l \ne \mathbf{V}_m$, for all different l, m. (No ordered pair of cities to

investigate is repeated within a test case.)

Small dataset

$\mathbf{D}_{ij} = -1$, for all i, j where $i + 1 \neq j$. (The cities are in a single line; each route goes from one city to the next city in line.)
$\mathbf{Q} = 1$.
$\mathbf{U}_1 = 1$.
$\mathbf{V}_1 = \mathbf{N}$. (The only delivery to calculate is between the first and last cities in the line).

Large dataset

$1 \leq \mathbf{Q} \leq 100$.
$1 \leq \mathbf{U}_k \leq \mathbf{N}$, for all k.
$1 \leq \mathbf{V}_k \leq \mathbf{N}$, for all k.

Sample

Input	Output

```
3
3 1
2 3
2 4
4 4
-1 1 -1
-1 -1 1
-1 -1 -1
1 3
4 1
13 10
1 1000
10 8
5 5
-1 1 -1 -1
-1 -1 1 -1
-1 -1 -1 10
-1 -1 -1 -1
1 4
4 3
30 60
10 1000
12 5
20 1
-1 10 -1 31
10 -1 10 -1
-1 -1 -1 10
15 6 -1 -1
2 4
3 1
3 2
```

```
Case #1: 0.583333333
Case #2: 1.2
Case #3: 0.51 8.01 8.0
```

Note that the last sample case would not appear in the Small dataset.

In Case #1 there are two options: use the horse in city 1 for the entire trip, or change horses in city 2. Both horses have enough endurance, so both options are viable. Since the horse in city 2 is faster, it is better to change, for a total time of 1/3 + 1/4.

In Case #2 there are two intermediate cities in which you can change horses. If you change horses in city 2, however, your new horse, while blazingly fast, will not have enough endurance, so you will be forced to change again in city 3. If you keep your horse, you will have the option to change horses (or not) in city 3. So, the three options, with their total times, are:

1. Change horses in both city 2 and 3 (1/10 + 1/1000 + 10/8 = 1.351).
2. Change horses just in city 3 (2/10 + 10/8 = 1.45).
3. Never change horses (12/10 = 1.2).

In Case #3, there are lots of alternatives for each delivery. The optimal one for the first delivery (city 2 to city 4) is to go to city 1 in time 10/1000, change horses, and then go to cities 2, 3 and 4, in that order, using the horse from city 1, which takes time (10 + 10 + 10) / 60.

For the second delivery (city 3 to city 1) you have no choice but to first go to city 4 which takes time 10/5. Your relatively fast horse does not have enough endurance to get anywhere else, so you need to grab the horse in city 4. You could use it to get directly to city 1 in time 15, but that would be slower than riding it to city 2 in time 6 and then using the blazingly fast horse in city 2 to get to city 1 in just 10/1000 extra time.

In the third delivery (city 3 to city 2) of Case #3 it is optimal to use the first two steps of the previous one, for a total time of 10/5 + 6 = 8.

Problem A. Ample Syrup

Small input 9 points	Solve A-small
Large input 16 points	Solve A-large

Problem

The kitchen at the Infinite House of Pancakes has just received an order for a stack of **K** pancakes! The chef currently has **N** pancakes available, where **N** ≥ **K**. Each pancake is a cylinder, and different pancakes may have different radii and heights.

As the sous-chef, you must choose **K** out of the **N** available pancakes, discard the others, and arrange those **K** pancakes in a stack on a plate as follows. First, take the pancake that has the largest radius, and lay it on the plate on one of its circular faces. (If multiple pancakes have the same radius, you can use any of them.) Then, take the remaining pancake with the next largest radius and lay it on top of that pancake, and so on, until all **K** pancakes are in the stack and the centers of the circular faces are aligned in a line perpendicular to the plate, as illustrated by this example:

You know that there is only one thing your diners love as much as they love pancakes: syrup! It is best to maximize the total amount of exposed pancake surface area in the stack, since more exposed pancake surface area means more places to pour on delicious syrup. Any part of a pancake that is not touching part of another pancake or the plate is considered to be exposed.

If you choose the **K** pancakes optimally, what is the largest total exposed pancake surface area you can achieve?

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 **N** and **K**: the total number of available pancakes, and the size of the stack that the diner has ordered. Then, **N** more lines follow. Each contains two integers $\mathbf{R_i}$ and $\mathbf{H_i}$: the radius and height of the i-th pancake, in millimeters.

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 maximum possible total exposed pancake surface area, in millimeters squared. y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{K} \le \mathbf{N}$.
$1 \le \mathbf{R_i} \le 10^6$, for all i.
$1 \le \mathbf{H_i} \le 10^6$, for all i.

Small dataset

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

Large dataset

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

Sample

Input	Output

```
4         Case #1: 138230.076757951
2 1       Case #2: 150796.447372310
100 20    Case #3: 43982.297150257
200 10    Case #4: 625.176938064
2 2
100 20
200 10
3 2
100 10
100 10
100 10
4 2
9 3
7 1
10 1
8 4
```

In Sample Case #1, the "stack" consists only of one pancake. A stack of just the first pancake would have an exposed area of $\pi \times \mathbf{R_0}^2 + 2 \times \pi * \mathbf{R_0} \times \mathbf{H_0} = 14000\pi$ mm^2. A stack of just the second pancake would have an exposed area of 44000π mm^2. So it is better to use the second pancake.

In Sample Case #2, we can use both of the same pancakes from case #1. The first pancake contributes its top area and its side, for a total of 14000π mm^2. The second pancake contributes some of its top area (the part not covered by the first pancake) and its side, for a total of 34000π mm^2. The combined exposed surface area is 48000π mm^2.

In Sample Case #3, all of the pancakes have radius 100 and height 10. If we stack two of these together, we effectively have a single new cylinder of radius 100 and height 20. The exposed surface area is 14000π mm^2.

In Sample Case #4, the optimal stack uses the pancakes with radii of 8 and 9.

Problem B. Parenting Partnering

Small input 12 points	Solve B-small
Large input 20 points	Solve B-large

Problem

Cameron and Jamie are longtime life partners and have recently become parents! Being in charge of a baby, exciting as it is, is not without challenges. Given that both parents have a scientific mind, they have decided to take a scientific approach to baby care.

Cameron and Jamie are establishing a daily routine and need to decide who will be the main person in charge of the baby at each given time. They have been equal partners their whole relationship, and they do not want to stop now, so they decided that each of them will be in charge for exactly 12 hours (720 minutes) per day.

Cameron and Jamie have other activities that they either need or want to do on their own. Cameron has $\mathbf{A_C}$ of these and Jamie has $\mathbf{A_J}$. These activities always take place at the same times each day. None of Cameron's activities overlap with Jamie's activities, so at least one of the parents will always be free to take care of the baby.

Cameron and Jamie want to come up with a daily baby care schedule such that:

- Scheduled baby time must not interfere with a scheduled activity. That is, during Cameron's activities, Jamie has to be in charge of the baby, and vice versa.
- Each of Cameron and Jamie must have exactly 720 minutes assigned to them.
- The number of exchanges — that is, the number of times the person in charge of the baby changes from one partner to the other — must be as small as possible.

For example, suppose that Jamie and Cameron have a single activity each: Jamie has a morning activity from 9 am to 10 am, and Cameron has an afternoon activity from 2 pm to 3 pm. One possible but suboptimal schedule would be for Jamie to take care of the baby from midnight to 6 am and from noon to 6 pm, and for Cameron to take care of the baby from 6 am to noon and 6 pm to midnight. That fulfills the first two conditions, and requires a total of 4 exchanges, which happen at midnight, 6 am, noon and 6 pm. If there is an exchange happening at midnight, it is counted exactly once, not zero or two times.

A better option would be for Cameron to take care of the baby from midnight to noon, and Jamie to take care of the baby from noon to midnight. This schedule also fulfills the first two conditions, but it uses only 2 exchanges, which is the minimum possible.

Given Cameron's and Jamie's lists of activities, and the restrictions above, what is the minimum possible number of exchanges in a daily schedule?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a line containing two integers $\mathbf{A_C}$ and $\mathbf{A_J}$, the number of activities that Cameron and Jamie have, respectively. Then, $\mathbf{A_C} + \mathbf{A_J}$ lines follow. The first $\mathbf{A_C}$ of these lines contain two integers $\mathbf{C_i}$ and $\mathbf{D_i}$ each. The i-th of Cameron's activities starts exactly $\mathbf{C_i}$ minutes after the start of the day at midnight and ends exactly $\mathbf{D_i}$ minutes after the start of the day at midnight (taking exactly $\mathbf{D_i} - \mathbf{C_i}$ minutes). The last $\mathbf{A_J}$ of these lines contain two integers $\mathbf{J_i}$ and $\mathbf{K_i}$ each, representing the starting and ending time of one of Jamie's activities, in minutes counting from the start of the day at midnight (same format as Cameron's). No activity spans two days, and no two activities overlap (except that one might end exactly as another starts, but an exchange can still occur at that time).

Output

For each test case, output one line containing `Case #x: y`, where x is the test case number (starting from 1) and y the minimum possible number of exchanges, as described in the statement.

Limits

$1 \le \mathbf{T} \le 100$.
$0 \le \mathbf{C_i} < \mathbf{D_i} \le 24 \times 60$, for all i.
$0 \le \mathbf{J_i} < \mathbf{K_i} \le 24 \times 60$, for all i.
Any two of the intervals of {[$\mathbf{C_i}$, $\mathbf{D_i}$) for all i} union {[$\mathbf{J_i}$, $\mathbf{K_i}$) for all i} have an empty intersection. (The intervals are closed on the left and open on the right, which ensures that two exactly consecutive intervals have nothing in between but do not overlap.)
sum of {$\mathbf{D_i}$ - $\mathbf{C_i}$ for all i} ≤ 720.
sum of {$\mathbf{K_i}$ - $\mathbf{J_i}$ for all i} ≤ 720.

Small dataset

$0 \le \mathbf{A_C} \le 2$.
$0 \le \mathbf{A_J} \le 2$.
$1 \le \mathbf{A_C} + \mathbf{A_J} \le 2$.

Large dataset

$0 \le \mathbf{A_C} \le 100$.
$0 \le \mathbf{A_J} \le 100$.
$1 \le \mathbf{A_C} + \mathbf{A_J} \le 200$.

Sample

Input	Output
5 1 1 540 600 840 900 2 0 900 1260 180 540 1 1 1439 1440 0 1 2 2 0 1 1439 1440 1438 1439 1 2 3 4 0 10 1420 1440 90 100 550 600 900 950 100 150 1050 1400	Case #1: 2 Case #2: 4 Case #3: 2 Case #4: 4 Case #5: 6

Note that Sample Cases #4 and #5 would not appear in the Small dataset.

Sample Case #1 is the one described in the problem statement.

In Sample Case #2, Jamie must cover for all of Cameron's activity time, and then Cameron must cover all the remaining time. This schedule entails four exchanges.

In Sample Case #3, there is an exchange at midnight, from Cameron to Jamie. No matter how the parents divide up the remaining 1438 non-activity minutes of the day, there must be at least one exchange from Jamie to Cameron, and there is no reason to add more exchanges than that.

In Sample Case #4, note that back-to-back activities can exist for the same partner or different partners. There is no exchange at midnight because Cameron has activities both right before and right after that time. However, the schedule needs to add some time for Cameron in between Jamie's activities, requiring a total of 4 exchanges. Notice that it is optimal to add a single interval for Cameron of length 718 somewhere between minutes 2 and 1438, but the exact position of that added interval does not impact the number of exchanges, so there are multiple optimal schedules.

In Sample Case #5, a possible optimal schedule is to assign Cameron to the intervals (in minutes) 100-200, 500-620, and 900-1400.


Powered by
Google Cloud Platform

Problem C. Core Training

Small input 1 15 points	Solve C-small-1
Small input 2 28 points	Solve C-small-2

Problem

Writing Code Jam problems is hard, so we have built an AI to come up with new ideas. To make the AI as creative as possible, we have given it **N** different "cores", each of which has its own "personality". However, just like people, these cores may become distracted or corrupt or may refuse to work; the i-th core has a *success probability* $\mathbf{P_i}$ of functioning properly. As long as at least **K** of the cores function properly, the AI will function properly. Otherwise, it will probably become evil and trap us in a maze of fiendish puzzles of its own design. And who knows what it might do to Code Jam — it might just write a bunch of tough probability problems!

To prevent this from happening, we plan to train one or more of the cores to become more reliable. We have a total of **U** "training units" that we can use to improve the cores. Spending X units on the i-th core will add X to its success probability. We can divide up the units among the cores however we like, and it is possible that one or more cores may not receive any units. Of course, a core's success probability cannot be increased above 1.

If we assign the training units to maximize the probability that the AI will function properly, what is that probability?

Solving this problem

This problem has 2 Small datasets and no Large dataset. You must solve the first Small dataset before you can attempt the second Small dataset. You will be able to retry either of the datasets (with a time penalty).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each consists of three lines. The first line contains two integers **N** and **K**: the total number of cores, and the minimum number of cores that must succeed for the AI to function properly. The second line contains one rational **U**: the number of training units. The third line contains **N** rational numbers $\mathbf{P_i}$; the i-th of these gives the probability that the i-th core will function properly. All of these probabilities are specified to exactly four decimal places of precision.

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 probability that the AI will function properly if the training units are assigned optimally. y will be considered correct if it is within an absolute or relative error of 10^{-6} of the correct answer. See the FAQ for an explanation of what that means, and what formats of real numbers we accept.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 50$.
For all i, $0.0000 \le \mathbf{P_i} \le 1.0000$.
$0.0000 \le \mathbf{U} \le \mathbf{N}$ - the sum of all $\mathbf{P_i}$. (There will not be more training units than can be used.)

Small dataset 1

K = **N**. (All of the cores must function properly for the AI to function properly.)

Small dataset 2

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

Sample

Input	Output
4 4 4 1.4000 0.5000 0.7000 0.8000 0.6000	Case #1: 1.000000 Case #2: 0.250000 Case #3: 0.980000 Case #4: 0.760000

```
2 2
1.0000
0.0000 0.0000
2 1
0.0000
0.9000 0.8000
2 1
0.1000
0.4000 0.5000
```

Note that the last two sample cases would not appear in Small dataset 1.

In Sample Case #1, we have enough training units to spend to give all cores a success probability of 1, so the AI will certainly function properly.

In Sample Case #2, both of the cores must function properly for the AI to function properly, so we must give each core at least some training units. The best option turns out to be to train each one up to 0.5. Then the probability that the AI functions properly is $0.5 \times 0.5 = 0.25$. Any other assignment is inferior; for instance, if we train one core to 0.9 and the other core to 0.1, the probability of success is only $0.9 \times 0.1 = 0.09$.

In Sample Case #3, we have no training units to spend, and at least one of the two cores must function properly for the AI to function properly. We can approach this by first calculating the probability that the AI does *not* function properly, which happens only if both cores fail to function properly. The probability that both cores fail is $(1 - 0.9) \times (1 - 0.8) = 0.02$. So the probability that at least one core functions properly, and thus that the AI functions properly, is $1 - 0.02 = 0.98$.

In Sample Case #4, the optimal strategy is to give all the training units to the second core. That makes the probability of at least one core functioning properly $1 - (0.6 \times 0.4) = 0.76$. All other options are inferior; for example, giving all the training units to the first core only yields 0.75, and dividing them equally among the cores gives 0.7525.

Problem A. Fresh Chocolate

Small input 6 points	Solve A-small
Large input 10 points	Solve A-large

Problem

You are the public relations manager for a chocolate manufacturer. Unfortunately, the company's image has suffered because customers think the owner is cheap and miserly. You hope to undo that impression by offering a free factory tour and chocolate tasting.

Soon after starting the new project, you realized that the company owner's reputation is well-deserved: he only agreed to give away free chocolate if you would minimize the cost. The chocolate to be given away comes in packs of **P** pieces. You would like to open new packs for each tour group, but the owner insists that if there are leftover pieces from one group, they must be used with the next tour group before opening up any new packs.

For instance, suppose that each pack contains **P**=3 pieces, and that a tour group with 5 people comes. You will open two packs to give one piece to each person, and you will have one piece left over. Suppose that after that, another tour group with 6 people comes. They will receive the leftover piece, and then you will open two more packs to finish giving them their samples, and so you will have one piece left over again. If two groups with 4 people each come right after, the first of those will get the leftover piece plus a full pack, and the last 4 person group will get their pieces from two newly opened packs. Notice that you cannot open new packs until all leftovers have been used up, even if you plan on using all of the newly opened pack immediately.

In the example above, 2 out of the 4 groups (the first and last groups) got all of their chocolate from freshly opened packs. The other 2 groups got some fresh chocolate and some leftovers. You know that giving out leftovers is not the best way to undo the owner's miserly image, but you had to accept this system in order to get your cheap boss to agree to the project. Despite the unfavorable context, you are committed to doing a good job.

You have requests from **N** groups, and each group has specified the number of people that will come into the factory. Groups will come in one at a time. You want to bring them in in an order that maximizes the number of groups that get only fresh chocolate and no leftovers. You cannot reject groups, nor have a group get chocolate more than once, and you need to give exactly one piece to each person in each group.

In the example above, if instead of 5, 6, 4, 4, the order were 4, 5, 6, 4, a total of 3 groups (all but the 5 person group) would get only fresh chocolate. For that set of groups, it is not possible to do better, as no arrangement would cause all groups to get only fresh chocolate.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case consists of two lines. The first line contains two integers **N**, the number of groups coming for a tour, and **P**, the number of pieces of chocolate per pack. The second line contains **N** integers $\mathbf{G_1}$, $\mathbf{G_2}$, ..., $\mathbf{G_N}$, the number of people in each of the groups.

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 groups that will receive only fresh chocolate if you bring them in in an order that maximizes that number.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 100$.
$1 \le \mathbf{G_i} \le 100$, for all i.

Small dataset

$2 \le \mathbf{P} \le 3$.

Large dataset

$2 \le \mathbf{P} \le 4$.

Sample

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

Sample Case #1 is the one explained in the statement. Besides the possible optimal order given above, other orders like 6, 5, 4, 4 also maximize the number of groups with only fresh chocolate, although the groups that get the fresh chocolate are not necesarily the same. Notice that we only care about the number of groups that get the best experience, not the total number of people in them.

In Sample Case #2, the groups are the same as in Case #1, but the packs contain two pieces each. In this case, several ways of ordering them — for instance, 4, 4, 6, 5 — make all groups get only fresh chocolate.

In Sample Case #3, all groups are single individuals, and they will all eat from the same pack. Of course, only the first one to come in is going to get a freshly opened pack.

Problem B. Roller Coaster Scheduling

Small input 7 points	Solve B-small
Large input 14 points	Solve B-large

Problem

You created a new roller coaster that is about to open. Its train consists of a single row of **N** seats numbered 1 through **N** from front to back. Of course, seats closer to the front are more valuable. Customers have already purchased opening-day tickets. Each ticket allows a specific customer to take one ride on the coaster in a particular seat. Some customers may have bought more than one ticket, and they expect to go on one ride for each ticket.

You need to decide how many roller coaster rides there will be on opening day. On each ride, one customer can sit in each seat; some seats on a ride might be left empty. You cannot assign a customer to more than one seat in the same ride, nor can you put two customers on the same seat in any given ride.

You wish to minimize the number of rides required to honor all tickets, to reduce operational costs. To reduce the required number of rides, you can *promote* any number of tickets. Promoting a ticket means taking a customer's ticket and giving that customer a new ticket for a seat closer to the front of the train (that is, a seat with a lower number). You would prefer to promote as few tickets as possible, since too many promotions might cause customers to get greedy and ask for more promotions in the future.

Given the positions and buyers of all the tickets that have been sold, what is the minimum number of rides needed to honor all tickets, using as many promotions as needed and scheduling the rides optimally? And what is the minimum number of ticket promotions necessary to attain that number of rides? Note that promoting a given customer on a given ride from seat 4 to seat 2, for example, counts as only one promotion, not two separate ones.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a single line with three integers **N**, the number of seats in the roller coaster, **C**, the number of potential customers, and **M**, the number of tickets sold. The customers are identified with numbers between 1 and **C**. Then, **M** lines follow, each containing two integers: $\mathbf{P}_i$, the position in the roller coaster assigned to the i-th ticket, and $\mathbf{B}_i$, the identifier of the buyer of that ticket.

Output

For each test case, output one line containing `Case #x: y z`, where x is the test case number (starting from 1), y is the minimum number of rides you need to honor all tickets if you use the promotions and schedule the rides optimally, and z is the minimum number of promotions you need to make be able to honor all tickets with y rides.

Limits

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 1000$.
$1 \le \mathbf{M} \le 1000$.
$1 \le \mathbf{P}_i \le \mathbf{N}$.
$1 \le \mathbf{B}_i \le \mathbf{C}$.

Small dataset

$\mathbf{C} = 2$.

Large dataset

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

Sample

Input	Output
5 2 2 2 2 1 2 2	Case #1: 1 1 Case #2: 2 0 Case #3: 2 0 Case #4: 2 1

```
2 2 2         Case #5: 2 1
1 1
1 2
2 2 2
1 1
2 1
1000 1000 4
3 2
2 1
3 3
3 1
3 3 5
3 1
2 2
3 3
2 2
3 1
```

Note that the last two sample cases would not appear in the Small dataset.

In Case #1, both customers purchased a ticket for position 2. It is impossible to honor both tickets with a single ride, but promoting either ticket to position 1 allows you to accommodate both tickets on the same round.

Case #2 is a similar story, except both tickets are for position 1. Since you cannot promote those tickets or exchange them for inferior tickets, you are forced to run 2 separate rides, one per customer.

Case #3 features the same customer purchasing both positions. Since you are forced to have 2 rides for that customer, there is no reason to give out any promotions.

In Case #4, notice that there may be both customers and positions with no tickets assigned. In this case, there are three tickets sold for position three. If you promote customer 2 to position 2, for instance, you can have one ride with customer 1 sitting in position 2 and customer 3 sitting in position 3, and a second ride with customer 2 in position 2 and customer 1 in position 3. Additional promotions will not allow you to decrease the number of rides, because customer 1 has two tickets and you need to honor those in different rides, regardless of position.

In Case #5, one optimal solution is to promote one of the 3 1 tickets to 1 1.

Problem C. Beaming With Joy

Small input 12 points	Solve C-small
Large input 17 points	Solve C-large

Problem

Joy is about to go on a long vacation, so she has hired technicians to install a security system based on infrared laser beams. The technicians have given her a diagram that represents her house as a grid of unit cells with **R** rows and **C** columns. Each cell in this grid contains one of the following:

- `/`: A two-sided mirror that runs from the cell's lower left corner to its upper right corner.
- `\`: A two-sided mirror that runs from the cell's upper left corner to its lower right corner.
- `-`: A beam shooter that shoots horizontal beams out into the cells (if any) to the immediate left and right of this cell.
- `|`: A beam shooter that shoots vertical beams out into the cells (if any) immediately above and below this cell.
- `#`: A wall. (Note that the house is not necessarily surrounded by a border of walls; this is one reason why Joy needs a security system!)
- `.`: Nothing; the cell is empty.

Beams travel in straight lines and continue on through empty cells. When a beam hits a mirror, it bounces 90 degrees off the mirror's surface and continues. When a beam traveling to the right hits a `/` mirror, it bounces off the mirror and starts traveling up; beams traveling up, left, or down that hit a `/` mirror bounce off and travel right, down, or left, respectively. The `\` mirror behaves similarly: when a beam traveling right, up, left or down hits it, it bounces off and starts traveling down, left, up or right, respectively. When a beam hits a wall or goes out of the bounds of the grid, it stops. It is fine for beams to cross other beams, but if a beam hits any beam shooter (including, perhaps, the beam shooter that originated the beam), that beam shooter will be destroyed!

Joy wants to make sure that every empty cell in the house has at least one beam passing through it, and that no beam shooters are destroyed, since that would just be wasting money! Unfortunately, the technicians have already installed the system, so the most Joy can do is rotate some of the existing beam shooters 90 degrees. That is, for any number (including zero) of beam shooters, she can turn `-` into `|` or vice versa.

Can you find any way for Joy to achieve her goal, or determine that it is impossible? Note that it is *not* required to minimize the number of rotations of beam shooters.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line with two integers **R** and **C**: the number of rows and columns in the grid representing the house. Then, **R** lines of **C** characters each follow; each character is `/`, `\`, `-`, `|`, `#`, or `.`, as described in the statement.

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` if Joy cannot accomplish her goal, or `POSSIBLE` if she can. Then, if the case is possible, output the same **R** lines of **C** characters each from the input grid, with zero or more instances of `-` replaced by `|` or vice versa.

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

Limits

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{C} \leq 50$.
Each character in the grid is one of `/`, `\`, `-`, `|`, `#`, or `.`.
The number of `-` characters plus the number of `|` characters (that is, the number of beam shooters) in the grid is between 1 and 100, inclusive.
There is at least 1 `.` character (that is, empty space) in the grid.

Small dataset

$1 \leq \mathbf{R} \leq 5$.
There are no `/` or `\` characters (that is, no mirrors) in the grid.

Large dataset

1 ≤ **R** ≤ 50.

Sample

```
Input     Output

5         Case #1: IMPOSSIBLE
1 3       Case #2: POSSIBLE
-.-       #.##
3 4       #||#
#.##      ####
#--#      Case #3: POSSIBLE
####      |.
2 2       #|
-.        Case #4: POSSIBLE
#|        .-.
4 3       |//
.|.       .|.
-//       #\/
.-.       Case #5: IMPOSSIBLE
#\/
3 3
/|\
\\/
./#
```

Note that the last 2 sample cases would not appear in the Small dataset.

In Sample Case #1, if a beam shooter is positioned to shoot its beam into the empty cell, it will necessarily destroy the other beam shooter. So the case is `IMPOSSIBLE`.

In Sample Case #2, the leftmost beam shooter must be rotated to cover the empty cell. The rightmost beam shooter must also be rotated to avoid destroying the leftmost beam shooter.

In Sample Case #3, the existing beam shooters already cover all empty cells with their beams and do not destroy each other, so outputting the grid from the input would be acceptable. However, notice that the output that we have given is also correct.

In Sample Case #4, one acceptable solution is to rotate all three of the beam shooters. However, note that the following would also be acceptable:

```
.-.
|//
.-.
#\/
```

since it is not necessary for cells with mirrors to have a beam pass through them. (Who would steal giant diagonal mirrors, anyway?)

In Sample Case #5, the beam shooter would destroy itself no matter which orientation Joy chooses for it, so the case is `IMPOSSIBLE`.

Problem D. Shoot the Turrets

Small input 13 points	Solve D-small
Large input 21 points	Solve D-large

Problem

The fight to free the city from extraterrestrial invaders is over! People are happy that love and peace have returned.

The city is represented as a grid with **R** rows and **C** columns. Some cells on the grid are buildings (through which nobody can see, nobody can shoot, and nobody can walk), and some are streets (through which everybody can see, shoot and walk). Unfortunately, during the war, the now-defeated invaders set up automatic security turrets in the city. These turrets are only in streets (not in buildings). They pose a threat to the citizens, but fortunately, there are also some soldiers on the streets (not in buildings). Initially, no soldier is in the same place as a turret.

The invader turrets do not move. They are small, so they don't block sight and shooting. A soldier cannot walk through an active turret's cell, but can walk through it once it is destroyed. A turret can only see soldiers in the cells for which it has a horizontal or vertical line of sight. If a soldier enters such a cell, the turret does not fire. If a soldier attempts to exit such a cell (after entering it, or after starting in that cell), the turret fires. Luckily, a soldier can still shoot from that cell, and the turret will not detect that as movement. It means that none of your soldiers will actually die, because in the worst case they can always wait, motionless, for help (perhaps for a long time). Maybe you will have a chance to rescue them later.

Each soldier can make a total of **M** unit moves. Each of these moves must be one cell in a horizontal or vertical direction. Soldiers can walk through each other and do not block the lines of sight of other soldiers or turrets. Each soldier also has one bullet. If a soldier has a turret in her horizontal or vertical line of sight, the soldier can shoot and destroy it. Each shot can only destroy one turret, but the soldiers are such excellent shooters that they can even shoot past one or several turrets or soldiers in their line of sight and hit another turret farther away!

You are given a map (with the soldier and turret positions marked). What is the largest number of turrets that the soldiers can destroy?

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 integer **C** (the width of the map), **R** (the height of the map) and **M** (the number of unit moves each soldier can make). The next **R** lines contain **C** characters each, with `.` representing a street, `#` representing a building, `S` representing a soldier and `T` representing a turret.

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 maximum number of turrets that it is possible to destroy. Then **y** lines should follow: each should contain two integers **s_i** and **t_i** denoting that the **i**th thing that happens should be soldier **s_i** destroying turret **t_i** (you don't need to specify exactly how the soldier has to move). If multiple valid strategies exist, you may output any one of them.

Soldiers are numbered from 1, reading from left to right along the top row, then left to right along the next row down from the top, and so on, from top to bottom.

Turrets use their own independent numbers, and are numbered starting from 1, in the same way.

Limits

$1 \leq \mathbf{T} \leq 100$.
$0 \leq \mathbf{M} < \mathbf{C} \times \mathbf{R}$.

Small dataset

$1 \leq \mathbf{C} \leq 30$.
$1 \leq \mathbf{R} \leq 30$.
The number of S symbols is between 1 and 10.
The number of T symbols is between 1 and 10.

Large dataset

$1 \le \mathbf{C} \le 100$.
$1 \le \mathbf{R} \le 100$.
The number of S symbols is between 1 and 100.
The number of T symbols is between 1 and 100.

Sample

Input	Output
4 2 2 1 #S T. 2 6 4 .T .T .T S# S# S# 5 5 4 ..... SS#.T SS#TT SS#.T ..... 3 3 8 S.# .#. #.T	Case #1: 1 1 1 Case #2: 3 3 3 1 1 2 2 Case #3: 3 1 2 2 1 6 3 Case #4: 0

In Case #2, one of the possible solutions is to move soldier 3 up three cells and shoot turret 3. Then soldier 1 can move up one cell and right one cell (to where turret 3 was) and shoot past turret 2 to destroy turret 1. Finally, soldier 2 can move up three cells and shoot turret 2.

In Case #3, soldier 1 can move up one cell, then right three cells and shoot turret 2. Then soldier 2 can move up one cell, then right three cells and shoot turret 1. Finally, soldier 6 can move down one cell, then right three cells and shoot turret 3. Other soldiers have insufficient move range to shoot any other turrets.

In Case #4, the soldier cannot move to within the same row or column as the turret, so the turret cannot be destroyed.

Problem A. Googlements

Small input 3 points	Solve A-small
Large input 10 points	Solve A-large

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 1s in the googlement (which could be 0) and write down that value, then count the number of 2s 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 2s, zero 3s, and two 4s, so it will decay into the googlement `1002`. This has one 1, one 2, zero 3s, and zero 4s, 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

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

1 ≤ the length of **G** ≤ 5.

Large dataset

1 ≤ the length of **G** ≤ 9.

Sample

Input	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.

Problem B. Good News and Bad News

Small input 7 points	Solve B-small
Large input 19 points	Solve B-large

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 (**A_i**, **B_i**) that means that friend **A_i** can talk to friend **B_i**. It does not imply that friend **B_i** can talk to friend **A_i**; however, another of the ordered pairs might make that true.

For *every* such existing ordered pair (**A_i**, **B_i**), you want friend **A_i** to deliver some news to friend **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 **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 **A_i** and **B_i** representing that friend **A_i** can talk to friend **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 [-**F^2**, **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

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{A_i} \le \mathbf{F}$, for all i.
$1 \le \mathbf{B_i} \le \mathbf{F}$, for all i.
$\mathbf{A_i} \ne \mathbf{B_i}$, for all i. (A friend does not self-communicate.)
$(\mathbf{A_i}, \mathbf{B_i}) \ne (\mathbf{A_j}, \mathbf{B_j})$, for all $i \ne j$. (No pair of friends is repeated within a test case in the same order.)

Small dataset

$2 \le \mathbf{F} \le 4$.
$1 \le \mathbf{P} \le 12$.

Large dataset

$2 \le \mathbf{F} \le 1000$.
$1 \le \mathbf{P} \le 2000$.

Sample

Input	Output

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

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.

Problem C. Mountain Tour

Small input 6 points	Solve C-small
Large input 24 points	Solve C-large

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

$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{E_i} \leq \mathbf{C}$.
$\mathbf{E_i} \neq \text{ceiling}(i / 2)$, 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.)
$0 \leq \mathbf{L_i} \leq 23$.
$1 \leq \mathbf{D_i} \leq 1000$.
There is at least one route that starts and ends at camp 1 and includes each hiking tour exactly once.

Small dataset

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

Large dataset

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

Sample

Input	Output
2	Case #1: 32

```
2        Case #2: 192
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
```

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.

Problem D. Slate Modern

Small input 5 points	Solve D-small
Large input 26 points	Solve D-large

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

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{N} \le 200$.
$1 \le \mathbf{D} \le 10^9$.
$1 \le \mathbf{R_i} \le \mathbf{R}$, for all i. $1 \le \mathbf{C_i} \le \mathbf{C}$, for all i. $1 \le \mathbf{B_i} \le 10^9$, 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

$1 \le \mathbf{R} \le 200$.
$1 \le \mathbf{C} \le 200$.

Large dataset

$1 \le \mathbf{R} \le 10^9$.
$1 \le \mathbf{C} \le 10^9$.

Sample

Input	Output
4 2 3 2 2 2 1 4 1 2 7 1 2 1 1000000000 1 2 1000000000 3 1 2 100	Case #1: 40 Case #2: 999999986 Case #3: IMPOSSIBLE Case #4: IMPOSSIBLE

```
1 1 1
3 1 202
2 2 2 2
2 1 1
2 2 4
```

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.

Problem A. Dice Straight

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 A-small
Large input 15 points	Solve A-large

Problem

You have a special set of **N** six-sided dice, each of which has six different positive integers on its faces. Different dice may have different numberings.

You want to arrange some or all of the dice in a row such that the faces on top form a *straight* (that is, they show consecutive integers). For each die, you can choose which face is on top.

How long is the longest straight that can be formed in this way?

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 **N**, the number of dice. Then, **N** more lines follow; each of them has six positive integers $\mathbf{D_{ij}}$. The j-th number on the i-th of these lines gives the number on the j-th face of the i-th die.

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 length of the longest straight that can be formed.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{D_{ij}} \le 10^6$ for all i, j.

Small dataset

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

Large dataset

$1 \le \mathbf{N} \le 50000$.
The sum of **N** across all test cases ≤ 200000.

Sample

Input	Output
3 4 4 8 15 16 23 42 8 6 7 5 30 9 1 2 3 4 55 6 2 10 18 36 54 86 2 1 2 3 4 5 6 60 50 40 30 20 10 3 1 2 3 4 5 6 1 2 3 4 5 6 1 4 2 6 5 3	Case #1: 4 Case #2: 1 Case #3: 3

In sample case #1, a straight of length 4 can be formed by taking the 2 from the fourth die, the 3 from the third die, the 4 from the first die, and the 5 from the second die.

In sample case #2, there is no way to form a straight larger than the trivial straight of length 1.

In sample case #3, you can take a 1 from one die, a 2 from another, and a 3 from the remaining unused die. Notice that this case demonstrates that there can be multiple dice with the same set of values on their faces.

Problem B. Operation

Small input 10 points	Solve B-small
Large input 20 points	Solve B-large

Problem

Here at Code Jam, we love to play a game called "Operation". (No, it has nothing to do with surgery; why would you think that?) The game is played with cards, each card is labeled with a basic arithmetic operation (addition, subtraction, multiplication or division) $\mathbf{O_i}$ and an integer right operand $\mathbf{V_i}$ for that operation. For example, a card might say `+ 0`, or `- -2`, or `/ -4` — note that operands can be negative or zero, although a card with a division operation will never have 0 as an operand.

In each round of the game, a starting integer value **S** is chosen, and a set of **C** cards is laid out. The player must choose an order for the cards, using each card exactly once. After that, the operations are applied, in order, to the starting value **S**, and a final result is obtained.

Although all of the operands on the cards are integers, the operations are executed on rational numbers. For instance, suppose that the initial value is 5, and the cards are `+ 1`, `- 2`, `* 3`, and `/ -2`. If we put them in the order given above, the final result is (5 + 1 - 2) * 3 / (-2) = -6. Notice that the operations are performed in the order given by the cards, disregarding any operator precedence. On the other hand, if we choose the order `- 2`, `/ -2`, `+ 1`, `* 3`, the result is ((5 - 2) / (-2) + 1) * 3 = -3 / 2. That example turns out to be the maximum possible value for this set of cards.

Given a set of cards, can you figure out the maximum possible final value that can be obtained? Please give the result as an irreducible fraction with a positive denominator.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line with two integers **S** and **C**: the starting value for the game, and the number of cards. Then, **C** lines follow. The i-th of these lines represents one card, and contains one character $\mathbf{O_i}$ representing the operation (which is either +, -, *, or /) and one integer $\mathbf{V_i}$ representing the operand.

Output

For each test case, output one line containing `Case #x: y z`, where x is the test case number (starting from 1), and y and z are integers such that y/z is the maximum possible final value of the game, y and z do not have common divisors other than 1 and -1, and z is strictly greater than 0.

Limits

$1 \le \mathbf{T} \le 100$.
$-1{,}000 \le \mathbf{S} \le 1{,}000$.
$\mathbf{O_i}$ is one of +, -, *, or /, for all i.
$-1{,}000 \le \mathbf{V_i} \le 1{,}000$, for all i.
If $\mathbf{O_i}$ = /, then $\mathbf{V_i} \neq 0$, for all i.

Small dataset

$1 \le \mathbf{C} \le 15$.

Large dataset

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

Sample

Input	Output
5 1 2 - 3 * 2 5 4 + 1 - 2	Case #1: -1 1 Case #2: -3 2 Case #3: 100000000000000000000000 1 Case #4: 1 1 Case #5: 0 1

```
* 3
/ -2
1000 7
* -1000
* -1000
* 1000
* 1000
* 1000
* 1000
* 1000
-1 3
- -1
* 0
/ -1
0 1
+ 0
```

In Sample Case #1, the optimal strategy is to play the `* 2` card before the `- 3` card, which yields a result of -1. The unique rational expression of this as specified in the problem is -1 1.

Sample Case #2 is the one described in the third paragraph of the problem statement.

In Sample Case #3, we get the same answer regardless of the order in which we use the cards. Notice that the numerator of the answer is too large to fit in 64-bit integer.

In Sample Case #4, the largest result we can achieve is 1. One way is: `/ -1`, `* 0`, `- -1`.

In Sample Case #5, note that the only valid representation of the answer is `0 1`. `0 2` is invalid because it can be reduced. `0 -1` is invalid because the denominator must be positive.

Problem C. Spanning Planning

Small input
30 points

Solve C-small

Problem

A *spanning tree* of an undirected graph with N nodes is a tree with N-1 edges that uses only edges from N and includes all nodes in N.

Please construct a graph with at least 2 nodes, and no more than 22 nodes, such that the graph has *exactly* **K** different spanning trees. (Two spanning trees are considered different if and only if the sets of edges that they use are different.) The graph must have at most one edge per pair of nodes, and must not contain a loop (an edge from a node to itself).

It is guaranteed that at least one such graph exists for every **K** within the limits below.

Solving this problem

This problem has only 1 Small dataset and no Large dataset. You will be able to retry the dataset (with a time penalty).

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each consists of one line with an integer **K**: the desired number of spanning trees.

Output

For each test case, first output one line containing `Case #x: y`, where x is the test case number (starting from 1), and y is the number of nodes in your graph. (y must be between 2 and 22, inclusive.) Then, output y more lines. The i-th of these lines represents the i-th node in the graph, and must contain exactly y characters. The j-th character on the i-th line should be 1 if the i-th node and the j-th node are connected with an edge, and `0` otherwise. Note that this matrix will be symmetric and it will have all `0`s along its main diagonal.

If multiple answers are possible, you may output any of them. Note that we guarantee that at least one valid answer exists for every **K** within the limits below.

Limits

1 ≤ **T** ≤ 300.

Small dataset

3 ≤ **K** ≤ 10000.

Sample

Input	Output
2 3 8	Case #1: 3 011 101 110 Case #2: 4 0111 1001 1001 1110

In Case #1, the graph is a triangle, and removing any one edge creates a different spanning tree.

In Case #2, the available edges in our solution tree are 1-2, 1-3, 1-4, 2-4, and 3-4. The eight different spanning trees are defined by these sets of edges:

- 1-2, 1-3, 1-4
- 1-2, 1-3, 2-4
- 1-2, 1-3, 3-4
- 1-2, 1-4, 3-4

- 1-2, 2-4, 3-4
- 1-3, 1-4, 2-4
- 1-3, 2-4, 3-4
- 1-4, 2-4, 3-4

Submissions

Dice Straight

10pt	Not attempted **23/24 users** correct (96%)
15pt	Not attempted **18/21 users** correct (86%)

Operation

10pt	Not attempted **15/17 users** correct (88%)
20pt	Not attempted **12/12 users** correct (100%)

Spanning Planning

30pt	Not attempted **13/16 users** correct (81%)

Omnicircumnavigation

15pt	Not attempted **16/20 users** correct (80%)
20pt	Not attempted **6/12 users** correct (50%)

Stack Management

10pt	Not attempted **15/16 users** correct (94%)
30pt	Not attempted **0/1 users** correct (0%)

Teleporters

10pt	Not attempted **6/8 users** correct (75%)
30pt	Not attempted

Top Scores

Gennady.Korotkevich	120
zemen	110
vepifanov	110
SnapDragon	110
eatmore	100
apiapiapiad	95
simonlindholm	95
Zlobober	90
Endagorion	85
kevinsogo	80

Problem D. Omnicircumnavigation

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 15 points	Solve D-small
Large input 20 points	Solve D-large

Problem

Intrepid globetrotter K, who may or may not be the author of this problem, has been traveling a lot lately. On one of her recent trips, she traveled from San Francisco to Frankfurt to Johannesburg to Abu Dhabi to Singapore to Tokyo and back to San Francisco. On this trip, she circumnavigated the Earth by traveling along a closed path that touches every meridian. In other words, for every possible longitude, there is at least one point along this path at that longitude.

K is not sure that this trip qualifies as being *super awesome*, however, since it would also be possible to circumnavigate the Earth by flying to the North Pole and then walking around it, which does not seem to be particularly difficult (other than the part about flying to the North Pole). So she has decided to come up with a more generalized definition of circumnavigation. The new concept is called *omnicircumnavigation* — a closed path around the Earth (which we assume to be a sphere) that is a circumnavigation regardless of where one places the poles. In other words, an omnicircumnavigation is a closed path on the surface of a sphere that touches every possible hemisphere. (Touching the edge of a hemisphere is sufficient.) Equivalently, an omnicircumnavigation intersects every possible great circle — a circle of greatest possible diameter on the surface of a sphere.

You are given a sequence of **N** points on a sphere of radius 1. You need to check whether a path connecting those points in order is an omnicircumnavigation. The path is formed by connecting each pair of successive points along the shortest possible surface route, and connecting the last point to the first one in the same way. No two successive points (including the pair of the last point and the first point) are collinear with the origin. (That is, they are not antipodes — polar opposites — and they do not represent the same point on the surface of the sphere.)

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line containing **N**, the number of cities visited by K. The next **N** lines contain three integers $\mathbf{X_i}$, $\mathbf{Y_i}$ and $\mathbf{Z_i}$ each. The i-th point in the list is given by the coordinates ($\mathbf{X_i}$ / sqrt($\mathbf{X_i}^2 + \mathbf{Y_i}^2 + \mathbf{Z_i}^2$), $\mathbf{Y_i}$ / sqrt($\mathbf{X_i}^2 + \mathbf{Y_i}^2 + \mathbf{Z_i}^2$), $\mathbf{Z_i}$ / sqrt($\mathbf{X_i}^2 + \mathbf{Y_i}^2 + \mathbf{Z_i}^2$)).

Output

For each test case, output one line containing `Case #x: y`, where x is the case number and y is either `YES` or `NO` depending on whether the route is an omnicircumnavigation or not.

Limits

$1 \le \mathbf{T} \le 200$.
$-10^6 \le \mathbf{X_i} \le 10^6$, for all i.
$-10^6 \le \mathbf{Y_i} \le 10^6$, for all i.
$-10^6 \le \mathbf{Z_i} \le 10^6$, for all i.
At least one of the values in ($\mathbf{X_i}$, $\mathbf{Y_i}$, $\mathbf{Z_i}$) $\neq 0$, for all i. For all i, j such that (i + 1 = j) or (i = **N** - 1 and j = 0), neither of ($\mathbf{X_i}$, $\mathbf{Y_i}$, $\mathbf{Z_i}$) and ($\mathbf{X_j}$, $\mathbf{Y_j}$, $\mathbf{Z_j}$) is a multiple of the other. (No two successive points, including the last and first, are antipodes or represent the same point on the sphere.)

Small dataset

$3 \le \mathbf{N} \le 50$.

Large dataset

$3 \le \mathbf{N} \le 5000$.

Sample

Input	Output
4	Case #1: NO

```
3           Case #2: YES
1 0 0       Case #3: YES
0 1 0       Case #4: YES
0 0 1
8
5 5 5
5 -5 5
-5 -5 5
-5 5 5
-5 5 -5
-5 -5 -5
5 -5 -5
5 5 -5
3
1 0 0
-1 1 0
-1 -1 0
5
1 0 0
-1 1 0
2 0 0
-2 2 0
-1 -1 0
```

In Sample Case #1, the three points are the surface points of one octant of the sphere, and the path traces out that octant. There are many hemispheres that do not overlap that path at all.

In Sample Case #2, the eight points are the corners of a cube inscribed in the sphere; any hemisphere will contain at least some parts of that path. Note that dividing all values by 5 would produce an equivalent case (with the same set of points).

In Sample Case #3, the path is itself a great circle, and so every other great circle must intersect it somewhere.

Sample Case #4 uses the same three points as in Sample Case #3, except that the first two points are visited twice each. Note that a case may include multiple representations of the same point, and that a path may include the same points or connections more than once.

Problem E. Stack Management

Small input 10 points	Solve E-small
Large input 30 points	Solve E-large

Problem

You are playing a solitaire game in which there are **N** stacks of face-up cards, each of which initially has **C** cards. Each card has a *value* and a *suit*, and no two cards in the game have the same value/suit combination.

In one move, you can do one of the following things:

1. If there are two or more cards with the same suit that are on top of different stacks, you may remove the one of those cards with the smallest value from the game. (Whenever you remove the last card from a stack, the stack is still there — it just becomes empty.)
2. If there is an empty stack, you may take a card from the top of any one of the non-empty stacks and place it on top of (i.e., as the only card in) that empty stack.

You win the game if you can make a sequence of moves such that eventually, each stack contains at most one card. Given a starting arrangement, determine whether it is possible to win the game.

Input

The first line of the input gives the number **P** of premade stacks that will be used in the test cases. Then, **P** lines follow. The i-th of those lines begins with an integer $\mathbf{C_i}$, the number of cards in the i-th of those premade stacks, and continues with $\mathbf{C_i}$ ordered pairs of integers. The j-th of these ordered pairs has two integers $\mathbf{V_{ij}}$ and $\mathbf{S_{ij}}$, representing the value and suit of the j-th card from the top in the i-th premade stack.

Then, there is another line with one integer **T**, the number of test cases. **T** test cases follow. Each case begins with one line with two integers **N** and **C**: the number of stacks, and the number of cards in each of those stacks. Then, there is one line with **N** integers $\mathbf{P_i}$, representing the indexes (starting from 0) of the test case's set of premade stacks.

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 `POSSIBLE` if it is possible to win the game, or `IMPOSSIBLE` otherwise.

Limits

$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{P} \le 60000$.
$0 \le \mathbf{P_i} < \mathbf{P}$, for all i.
The $\mathbf{P_i}$-th premade stack has exactly **C** cards.
No two cards in a test case have the same value/suit combination.

Small dataset

$2 \le \mathbf{N} \le 4$.
$2 \le \mathbf{C_i} \le 13$, for all i.
$2 \le \mathbf{C} \le 13$.
$1 \le \mathbf{V_{ij}} \le 13$, for all i and j.
$1 \le \mathbf{S_{ij}} \le 4$, for all i and j.

Large dataset

$2 \le \mathbf{N} \le 50000$.
$2 \le \mathbf{C_i} \le 50000$, for all i.
$2 \le \mathbf{C} \le 50000$.
$4 \le \mathbf{N} \times \mathbf{C} \le 10^5$.
$1 \le \mathbf{V_{ij}} \le 50000$, for all i and j.
$1 \le \mathbf{S_{ij}} \le 50000$, for all i and j.

Sample

Input	Output

```
5            Case #1: POSSIBLE
2 7 2 7 1    Case #2: IMPOSSIBLE
2 6 4 7 4
2 3 2 6 2
2 4 2 10 2
2 5 4 7 3
2
2 2
0 2
3 2
4 1 3
```

In sample case #1, there are two stacks, each of which has two cards. The first stack has a 7 of suit 2 on top and a 7 of suit 1 below that. The second stack has a 3 of suit 2 on top and a 6 of suit 2 below that.

It is possible to win the game as follows:

- Remove the 3 of suit 2 from the second stack.
- Remove the 6 of suit 2 from the second stack. This makes the second stack empty.
- Move the 7 of suit 2 to the second stack. Then the win condition is satisfied: all stacks have at most one card.

In sample case #2, there are three stacks, each of which has two cards. It is not possible to win the game in this case; the only possible move is to remove the 5 of suit 4 on top of the third stack, and this does not open up any new moves.

Problem F. Teleporters

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 F-small
Large input 30 points	Solve F-large

Problem

A short, short time into the future, in a nearby galaxy, you find yourself wanting to take a little trip and get away from the responsibilities of being Planet Thundera's only manufacturer of yarn. You decide to travel to Planet Care-a-Lot, the most relaxing planet there is. To travel, you are going to use the network of interstellar teleporters.

A teleporter is a small machine floating around somewhere in space. You can use it remotely from any point in space, but, due to the conservation of teleportation distance principle, it can teleport you to any other point in space at exactly the same L1 distance from the teleporter as your L1 distance to it before the teleportation. The L1 distance between two points at coordinates (x_0, y_0, z_0) and (x_1, y_1, z_1) is given by $|x_0 - x_1| + |y_0 - y_1| + |z_0 - z_1|$. Unfortunately, your space jetpack is broken, so you cannot move around on your own; to travel, you can only use the teleporters. You start at Planet Thundera. You can use a teleporter to travel from Planet Thundera to a point p_1, then use another to get from p_1 to p_2, and so on. The last teleportation must take you exactly to Planet Care-a-Lot.

Given the locations in 3-dimensional space of both planets and all the available teleporters, find out if it is possible for you to make the trip using only teleporters. If the trip can be made, what is the minimum number of teleportations needed to get to your destination? (Even if two teleportations use the same teleporter, they still count as separate teleportations.)

The input is given as points with coordinates that are all integers that fall within a certain range. However, you are allowed to teleport to intermediate points with integer or non-integer coordinates, and there are no range restrictions on the points you can visit.

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each test case starts with a single line with a single integer **N**, the number of teleporters available. Then, **N**+2 lines follow, each containing three integers $\mathbf{X_i}$, $\mathbf{Y_i}$, and $\mathbf{Z_i}$. The first of these lines represents the coordinates of your home planet, Thundera. The second of these lines represents the coordinates of your destination planet, Care-A-Lot. Each of the remaining **N** lines represents the coordinates of one of the teleporters.

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` if it is not possible to get from Thundera to Care-A-Lot using only the available teleporters, or, if it is possible, an integer representing the minimum number of teleportations needed.

Limits

$1 \le \mathbf{T} \le 100$.
$(\mathbf{X_i}, \mathbf{Y_i}, \mathbf{Z_i}) \ne (\mathbf{X_j}, \mathbf{Y_j}, \mathbf{Z_j})$ for all $i \ne j$. (No two described objects have the same coordinates.)

Small dataset

$1 \le \mathbf{N} \le 100$.
$-10^3 \le \mathbf{X_i} \le 10^3$, for all i.
$-10^3 \le \mathbf{Y_i} \le 10^3$, for all i.
$-10^3 \le \mathbf{Z_i} \le 10^3$, for all i.

Large dataset

$1 \le \mathbf{N} \le 150$.
$-10^{12} \le \mathbf{X_i} \le 10^{12}$, for all i.
$-10^{12} \le \mathbf{Y_i} \le 10^{12}$, for all i.
$-10^{12} \le \mathbf{Z_i} \le 10^{12}$, for all i.

Sample

Input	Output
3 1 0 0 0 0 4 0 0 3 0 2 0 0 1 0 0 11 0 0 3 0 0 0 3 0 0 0 6 2 0 6 0 0 3 0 0 6 1 0	Case #1: IMPOSSIBLE Case #2: 3 Case #3: 2

In Sample Case #1, the only teleporter is exactly 3 units away from Thundera, and we can only use it to go to another position that is *exactly* 3 units away from the teleporter. From that position, we can still only reach other positions that are exactly 3 units away from the teleporter. Since Care-a-Lot is 1 unit away from the teleporter, we can never reach it.

In Sample Case #2, the optimal strategy is to first use the teleporter at (0, 0, 3) to travel to (0, 0, 5). Then, from there, use the teleporter at (0, 0, 0) to travel to (0, 0, -5). Finally, from there, use the teleporter at (0, 0, 3) again to travel to (0, 0, 11). Note that the two uses of the teleporter at (0, 0, 3) cause us to travel different distances, because we are at different distances from the teleporter each time. Also note that the two uses of that teleporter count as two separate teleportations.

In Sample Case #3, the optimal strategy is to first use the teleporter at (3, 0, 0) to travel to (6, 0, 0). Then, from there, use the teleporter at (6, 1, 0) to travel to (6, 2, 0). Note that even though there was a teleporter at (6, 0, 0), merely occupying the same point as a teleporter does not count as using it.