Code Jam 2017 - Round 1A

Alphabet Cake

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

Time limit: 20 seconds per test set.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
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 (Test Set 1 - Visible)

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

Large dataset (Test Set 2 - Hidden)

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

Sample

Sample Input

```
3
3 3
G??
?C?
??J
3 4
CODE
????
?JAM
2 2
CA
KE
```

Sample Output

```
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.

Code Jam 2017 - Round 1A

Ratatouille

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

Memory limit: 1 GB.
$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{R_i} \le 10^6$, for all i.
$1 \le \mathbf{Q_{ij}} \le 10^6$, for all i and j.

Small dataset (Test Set 1 - Visible)

Time limit: 60 seconds.
$1 \le \mathbf{N} \le 2$.
$1 \le \mathbf{P} \le 8$.

Large dataset (Test Set 2 - Hidden)

Time limit: 120 seconds.
$1 \le \mathbf{N} \le 50$.
$1 \le \mathbf{P} \le 50$.
$\mathbf{N} \times \mathbf{P} \le 1000$.

Sample

Sample Input

```
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
```

Sample Output

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

```
800 1440 1600
1700 1620 900
```

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.

Code Jam 2017 - Round 1A

Play the Dragon

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

Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.

Small dataset (Test Set 1 - Visible)

Time limit: 60 seconds.
$1 \leq \mathbf{H_d} \leq 100$.
$1 \leq \mathbf{A_d} \leq 100$.

$1 \le \mathbf{H_k} \le 100$.
$1 \le \mathbf{A_k} \le 100$.
$0 \le \mathbf{B} \le 100$.
$0 \le \mathbf{D} \le 100$.

Large dataset (Test Set 2 - Hidden)

Time limit: 240 seconds.
$1 \le \mathbf{H_d} \le 10^9$.
$1 \le \mathbf{A_d} \le 10^9$.
$1 \le \mathbf{H_k} \le 10^9$.
$1 \le \mathbf{A_k} \le 10^9$.
$0 \le \mathbf{B} \le 10^9$.
$0 \le \mathbf{D} \le 10^9$.

Sample

Sample Input

```
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
```

Sample Output

```
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.