Power Arrangers (11pts, 21pts)

Competitive Submissions

You have not attempted this problem.

Last updated: May 4 2019, 11:03

Problem

Go, go, Power Arrangers! Everyone loves this team of five superhero high school students who wear the letters A, B, C, D, and E. When they stand side by side to confront evil monsters, they arrange their team in one of 120 possible different left-to-right orders, giving them various different tactical superpowers. They are even more popular than the Teenage Permutant Ninja Turtles!

Some critics of the show claim that the team only has its arrangment gimmick so that the owners of the show can sell 120 separate sets of 5 action figures, each of which features the team in a different left-to-right order, glued to a base so that the set cannot be rearranged. As an avid Power Arrangers fan, you have collected 119 of these sets, but you do not remember which set you are missing. Your 119 sets are lined up horizontally along a shelf, such that there are a total of 119 × 5 = 595 action figures in left-to-right order. You do not remember how the sets are arranged, but you know that the permutation of the sets is selected uniformly at random from all possible permutations, and independently for each case.

You do not want to waste any time figuring out which set you are missing, so you plan to look at the letters on at most **F** figures on the shelf. For instance, you might choose to look at the letter on the eighth figure from the left, which would be the third figure from the left in the second set from the left. When looking at a figure, you only get the letter from that one figure; the letters are hard to see, and the different team members look very similar otherwise!

After checking at most **F** figures, you must figure out which of the sets is missing, so you can complete your collection and be ready to face any possible evil threat!

Input and output

This is an interactive problem. You should make sure you have read the information in the [Interactive Problems section](https://codingcompetitions.withgoogle.com/codejam/faq#interactive-problems) of our FAQ.

Initially, your program should read a single line containing two integers **T**, the number of test cases, and **F**, the number of figures you are allowed to inspect per test case. Then, you need to process **T** test cases.

Within each test case, the missing set of figures is chosen uniformly at random from all possible sets, and the order of the remaining sets is chosen uniformly at random from all possible orders as well. Every choice is made independently of all other choices and of your inputs.

In each test case, your program will process up to **F** + 1 exchanges with our judge. You may make up to **F**exchanges of the following form:

* Your program outputs one line containing a single integer between 1 and 595, inclusive, indicating which figure (in left-to-right order along the shelf) you wish to look at. As a further example, 589 would represent the fourth figure from the left in the second set from the right.
* The judge responds with one line containing a single uppercase letter A, B, C, D, or E, indicating the letter on that figure. If you sent invalid data (e.g., a number out of range, or a malformed line), the judge will instead respond with one line containing the single uppercase letter N.

Then, after you have made as many of the **F** exchanges above as you want, you must make one more exchange of the following form:

* Your program outputs one line containing a single string of five uppercase letters: the permutation corresponding to the missing set (e.g., CADBE).
* The judge responds with one line containing a single uppercase letter: Y if your answer was correct, and N if it was not (or you provided a malformed line). If you receive Y, you should begin the next test case, or stop sending input if there are no more test cases.

After the judge sends N to your input stream (because of either invalid data or an incorrect answer), it will not send any other output. If your program continues to wait for the judge after receiving N, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

Limits

1 ≤ **T** ≤ 50.  
Time limit: 40 seconds per test set.  
Memory limit: 1GB.  
The missing set, and the order of the remaining sets, are chosen uniformly and independently at random.

Test set 1 (Visible)

**F** = 475.

Test set 2 (Hidden)

**F** = 150.

Testing Tool

You can use this testing tool to test locally or on our servers. To test locally, you will need to run the tool in parallel with your code; you can use our [interactive runner](https://storage.googleapis.com/coding-competitions.appspot.com/interactive_runner.py) for that. For more information, read the [Interactive Problems section](https://codingcompetitions.withgoogle.com/codejam/faq#interactive-problems) of the FAQ.

Local Testing Tool

To better facilitate local testing, we provide you the following script. Instructions are included inside. You are encouraged to add more test cases for better testing. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently.

If your code passes the testing tool but fails the real judge, please check the [Coding section](https://codingcompetitions.withgoogle.com/codejam/faq#coding) of our FAQ to make sure that you are using the same compiler as us.

[file\_downloadDownload testing\_tool.py](https://codingcompetitions.withgoogle.com/codejam/round/00000000000516b9/0000000000134e91)

Sample Interaction

This interaction corresponds to Test set 1.

t, f = readline\_int\_list() // Reads 50 into t and 475 into f

printline 10 to stdout // Looks at the last figure in the second set

// from the left

flush stdout

n = readline\_string() // Reads B into n. Ooh, team member B! They may

// not have the leadership ability of A, or the

// technical skill of C, but they entertain the

// team with clever quips!

printline 11 to stdout // Looks at the first figure in the third set

// from the left

flush stdout

n = readline\_string() // Reads B into n. Notice that B is at the start

// of the third set, whereas they were at the

// end of the second set.

printline 14 to stdout // Looks at the fourth figure in the third set

// from the left

flush stdout

n = readline\_string() // Reads D into n. Silent and brooding, team

// member D nonetheless fights fiercely to

// protect their friends... and the world!

printline ABCDE to stdout // We foolishly make a wild guess even though we

// could have looked at up to 472 more figures.

flush stdout

verdict = readline\_string() // Reads N into verdict (judge has decided our

// solution is incorrect)

exit // Exits to avoid an ambiguous TLE error

Robot Programming Strategy (10pts, 18pts)

#### Competitive Submissions

You have not attempted this problem.

Last updated: May 4 2019, 11:03

### Problem

After many sleepless nights, you have finally finished teaching a robotic arm to make the hand gestures required for the Rock-Paper-Scissors game. Now you just need to program it to compete in the upcoming robot tournament!

In this tournament, each robot uses a program that is a series of moves, each of which must be one of the following: R (for "Rock"), P (for "Paper"), or S (for "Scissors"). Paper beats Rock and loses to Scissors; Rock beats Scissors and loses to Paper; Scissors beats Paper and loses to Rock.

When two robots face off in a match, the first robot to play a winning move wins. To start, each robot plays the first move of its program. If the two moves are different, one of the moves beats the other and thus one of the robots wins the match. If the moves are the same, each robot plays the next move in its program, and so on.

Whenever a robot has reached the end of its program and needs its next move, it returns to the start of its program. So, for example, the fifth move of a robot with the program RSSP would be R. If a match goes on for over a googol (10100) of moves, the judges flip a fair coin to determine the winner.

Once a match is over, the winning robot resets, so it has no memory of the that match. In its next match, it starts by playing the first move of its program, and so on.

The tournament is played in K rounds and has a single-elimination "bracket" structure. There are N = 2K robots in total, numbered 0 through N - 1. In the first round, robot 0 plays a match against robot 1, robot 2 plays a match against robot 3, and so on, up to robots N - 2 and N - 1. The losers of those matches are eliminated from the tournament. In the second round, the winner of the 0-1 match faces off against the winner of the 2-3 match, and so on. Once we get to the K-th round, there is only one match, and it determines the overall winner of the tournament.

All of the other contestants are so confident that they have already publicly posted their robots' programs online. However, the robots have not yet been assigned numbers, so nobody knows in advance who their opponents will be. Knowing all of the other programs, is it possible for you to write a program that is *guaranteed* to win the tournament, no matter how the robot numbers are assigned?

### 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 containing an integer **A**: the number of adversaries (other robots) in the tournament. Then, there are **A** more lines; the i-th of these contains a string **Ci** of uppercase letters that represent the program of the i-th opponent's robot.

### Output

For each test case, output one line containing Case #x: y. If there is a string of between 1 and 500 characters that is guaranteed to win the tournament, as described above, then y should be the string of uppercase letters representing that program. Otherwise, y should be IMPOSSIBLE, in uppercase letters.

### Limits

1 ≤ **T** ≤ 100.  
Time limit: 20 seconds per test set.  
Memory limit: 1GB.  
Each character in **Ci** is uppercase R, P, or S, for all i.  
**A** = 2K - 1 for some integer K ≥ 1.

#### Test set 1 (Visible)

1 ≤ **A** ≤ 7.  
**Ci** is between 1 and 5 characters long, for all i.

#### Test set 2 (Hidden)

1 ≤ **A** ≤ 255.  
**Ci** is between 1 and 500 characters long, for all i.

### Sample

|  |  |
| --- | --- |
| Input | Output |
| 3  1  RS  3  R  P  S  7  RS  RS  RS  RS  RS  RS  RS | Case #1: RSRSRSP  Case #2: IMPOSSIBLE  Case #3: P |

In Sample Case #1, there is only one opponent, with the program RS. Our answer matches the opponent's moves for a while, and the opponent loops through its program several times. As is starts its fourth pass through its program, we beat it with P. Other valid solutions exist, like P, RR, and R.

In Sample Case #2, there are three opponents, with the programs R, P, and S. It is up to you to figure out why this case is IMPOSSIBLE!

In Sample Case #3, all seven opponents use the same program. Using the program P, for example, guarantees that you will win. Remember that each robot begins at the start of its program at the start of each match against a new opponent.

Bacterial Tactics (15pts, 25pts)

Competitive Submissions

You have not attempted this problem.

Last updated: May 4 2019, 11:03

Problem

Becca and Terry are microbiologists who have a friendly rivalry. When they need a break from their research, they like to play a game together. The game is played on a matrix of unit cells with **R** rows and **C** columns. Initially, each cell is either empty, or contains radioactive material.

On each player's turn, if there are no empty cells in the matrix, that player loses the game. Otherwise, they choose an empty cell and place a colony of bacteria there. Bacteria colonies come in two types: H (for "horizontal") and V (for "vertical").

* When a type H colony is placed into an empty cell, it occupies that cell (making it non-empty), and also tries to spread into the cell immediately to the west (if there is one) and the cell immediately to the east (if there is one).
* When a type V colony is placed into an empty cell, it occupies that cell (making it non-empty), and also tries to spread into the cell immediately to the south (if there is one) and the cell immediately to the north (if there is one).

Whenever a colony (of either type) tries to spread into a cell:

* If the cell contains radioactive material, the colony mutates and the player who placed the colony loses the game.
* If that cell is empty, the colony occupies that cell (making it non-empty), and then the rule above is triggered again (i.e. the colony will try to spread further).
* If the cell already contains bacteria (of any type), the colony does not spread into that cell.

Notice that it may be possible that all of a player's available moves would cause them to lose the game, and so they are doomed. See the sample case explanations below for examples of how the game works.

Becca makes the first move, and then the two players alternate moves until one of them loses the game. If both players play optimally, who will win? And, if Becca will win, how many distinct winning opening moves does she have? (Two opening moves are distinct if and only if they either use different cells, or different kinds of colony, or both.)

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each case begins with one line containing two integers **R** and **C**: the number of rows and columns, respectively, in the matrix. Then, there are **R**more rows of **C** characters each. The j-th character on the i-th of these lines represents the j-th column of the i-th row of the matrix. Each character is either . (an empty cell) or # (a cell with radioactive material).

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 an integer: either 0 if Becca will not win, or, if Becca will win, the number of distinct winning opening moves she can make, as described above.

Limits

Time limit: 30 seconds per test set.  
Memory limit: 1GB.  
1 ≤ **T** ≤ 100.

Test set 1 (Visible)

1 ≤ **R** ≤ 4.  
1 ≤ **C** ≤ 4.

Test set 2 (Hidden)

1 ≤ **R** ≤ 15.  
1 ≤ **C** ≤ 15.

Sample

|  |  |
| --- | --- |
| Input | Output |
| 5  2 2  ..  .#  4 4  .#..  ..#.  #...  ...#  3 4  #.##  ....  #.##  1 1  .  1 2  ## | Case #1: 0  Case #2: 0  Case #3: 7  Case #4: 2  Case #5: 0 |

In Sample Case #1, Becca cannot place an H colony in the southwest empty cell or a V colony in the northeast empty cell, because those would spread onto a radioactive cell and Becca would lose. She has only two possible strategies that do not cause her to lose immediately:

1. Place an H colony in the northwest or northeast empty cells. The colony will also spread to the other of those two cells.
2. Place a V colony in the northwest or southwest empty cell. The colony will also spread to the other of those two cells.

If Becca chooses strategy 1, Terry can place a V colony in the southwest empty cell. If Becca chooses strategy 2, Terry can place an H colony in the northeast empty cell. Either way, Becca has no empty cells to choose from on her next turn, so she loses and Terry wins.

In Sample Case #2, any of Becca's opening moves would cause a mutation.

In Sample Case #3, five of Becca's possible opening moves would cause a mutation, but the other seven are winning. She can place an H colony in any of the cells of the second row, or she can place a V colony in any of the cells of the second column. In either case, she leaves two disconnected sets of 1 or 2 cells each. In each of those sets, only one type of colony can be played, and playing it consumes all of the empty cells in that set. So, whichever of those sets Terry chooses to consume, Becca can consume the other, leaving Terry with no moves.

In Sample Case #4, both of Becca's two distinct possible opening moves are winning.

In Sample Case #5, Becca has no possible opening moves.