Code Jam 2019 - Qualification Round

Foregone Solution

Problem

Someone just won the Code Jam lottery, and we owe them **N** jamcoins! However, when we tried to print out an oversized check, we encountered a problem. The value of **N**, which is an integer, includes at least one digit that is a `4`... and the `4` key on the keyboard of our oversized check printer is broken.

Fortunately, we have a workaround: we will send our winner two checks for positive integer amounts A and B, such that neither A nor B contains any digit that is a `4`, and A + B = **N**. Please help us find any pair of values A and B that satisfy these conditions.

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 **N**.

Output

For each test case, output one line containing `Case #x: A B`, where `x` is the test case number (starting from 1), and `A` and `B` are positive integers as described above.

It is guaranteed that at least one solution exists. If there are multiple solutions, you may output any one of them. (See "What if a test case has multiple correct solutions?" in the Competing section of the FAQ. This information about multiple solutions will not be explicitly stated in the remainder of the 2019 contest.)

Limits

$1 \leq \mathbf{T} \leq 100$.
Time limit: 10 seconds per test set.
Memory limit: 1GB.
At least one of the digits of **N** is a 4.

Test set 1 (Visible)

$1 < \mathbf{N} < 10^5$.

Test set 2 (Visible)

$1 < \mathbf{N} < 10^9$.

Solving the first two test sets for this problem should get you a long way toward advancing. The third test set is worth only 1 extra point, for extra fun and bragging rights!

Test set 3 (Hidden)

$1 < \mathbf{N} < 10^{100}$.

Sample

Sample Input	Sample Output
3 4 940 4444	Case #1: 2 2 Case #2: 852 88 Case #3: 667 3777

In Sample Case #1, notice that A and B can be the same. The only other possible answers are `1 3` and `3 1`.

Code Jam 2019 - Qualification Round

You Can Go Your Own Way

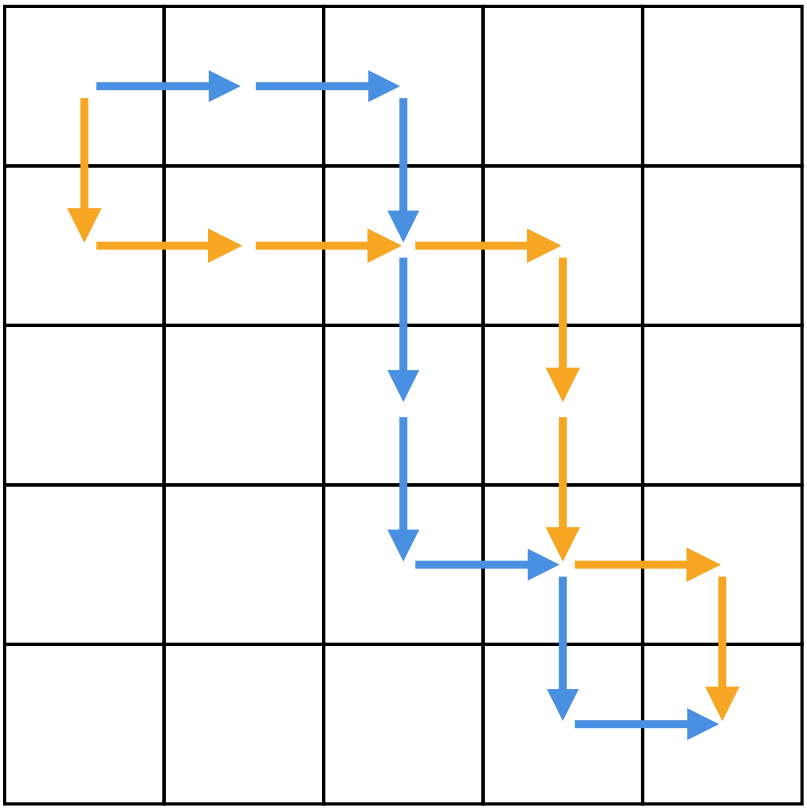
Problem

You have just entered the world's easiest maze. You start in the northwest cell of an **N** by **N** grid of unit cells, and you must reach the southeast cell. You have only two types of moves available: a unit move to the east, and a unit move to the south. You can move into any cell, but you may not make a move that would cause you to leave the grid.

You are excited to be the first in the world to solve the maze, but then you see footprints. Your rival, Labyrinth Lydia, has already solved the maze before you, using the same rules described above!

As an original thinker, you do not want to reuse any of Lydia's moves. Specifically, if her path includes a unit move from some cell A to some adjacent cell B, your path cannot also include a move from A to B. (However, in that case, it is OK for your path to visit A or visit B, as long as you do not go from A to B.) Please find such a path.

In the following picture, Lydia's path is indicated in blue, and one possible valid path for you is indicated in orange:



Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow; each case consists of two lines. The first line contains one integer **N**, giving the dimensions of the maze, as described above. The second line contains a string **P** of 2**N** - 2 characters, each of which is either uppercase `E` (for east) or uppercase `S` (for south), representing Lydia's valid path through the maze.

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 a string of 2**N** - 2 characters each of which is either uppercase `E` (for east) or uppercase `S` (for south), representing your valid path through the maze that does not conflict with Lydia's path, as described above. It is guaranteed that at least one answer exists.

Limits

1 ≤ **T** ≤ 100.
Time limit: 15 seconds per test set.
Memory limit: 1GB.
P contains exactly **N** - 1 `E` characters and exactly **N** - 1 `S` characters.

Test set 1 (Visible)

2 ≤ **N** ≤ 10.

Test set 2 (Visible)

2 ≤ **N** ≤ 1000.

Test set 3 (Hidden)

For at most 10 cases, 2 ≤ **N** ≤ 50000.
For all other cases, 2 ≤ **N** ≤ 10000.

Sample

Sample Input

```
2
2
SE
5
EESSSESE
```

Sample Output

```
Case #1: ES
Case #2: SEEESSES
```

In Sample Case #1, the maze is so small that there is only one valid solution left for us.

Sample Case #2 corresponds to the picture above. Notice that it is acceptable for the paths to cross.

Code Jam 2019 - Qualification Round

Cryptopangrams

Problem

On the Code Jam team, we enjoy sending each other *pangrams*, which are phrases that use each letter of the English alphabet at least once. One common example of a pangram is "the quick brown fox jumps over the lazy dog". Sometimes our pangrams contain confidential information — for example, `CJ QUIZ: KNOW BEVY OF DP FLUX ALGORITHMS` — so we need to keep them secure.

We looked through a cryptography textbook for a few minutes, and we learned that it is very hard to factor products of two large prime numbers, so we devised an encryption scheme based on that fact. First, we made some preparations:

- We chose 26 different prime numbers, none of which is larger than some integer **N**.
- We sorted those primes in increasing order. Then, we assigned the smallest prime to the letter `A`, the second smallest prime to the letter `B`, and so on.
- Everyone on the team memorized this list.

Now, whenever we want to send a pangram as a message, we first remove all spacing to form a plaintext message. Then we write down the product of the prime for the first letter of the plaintext and the prime for the second letter of the plaintext. Then we write down the product of the primes for the second and third plaintext letters, and so on, ending with the product of the primes for the next-to-last and last plaintext letters. This new list of values is our ciphertext. The number of values is one smaller than the number of characters in the plaintext message.

For example, suppose that **N** = 103 and we chose to use the first 26 odd prime numbers, because we worry that it is too easy to factor even numbers. Then `A` = 3, `B` = 5, `C` = 7, `D` = 11, and so on, up to `Z` = 103. Also suppose that we want to encrypt the `CJ QUIZ`... pangram above, so our plaintext is `CJQUIZKNOWBEVYOFDPFLUXALGORITHMS`. Then the first value in our ciphertext is 7 (the prime for `C`) times 31 (the prime for `J`) = `217`; the next value is `1891`, and so on, ending with `3053`.

We will give you a ciphertext message and the value of **N** that we used. We will not tell you which primes we used, or how to decrypt the ciphertext. Do you think you can recover the plaintext anyway?

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**, as described above, and **L**, the length of the list of values in the ciphertext. The second line contains **L** integers: the list of values in the ciphertext.

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 a string of **L** + 1 uppercase English alphabet letters: the plaintext.

Limits

$1 \leq \mathbf{T} \leq 100$.
Time limit: 20 seconds per test set.
Memory limit: 1 GB.
$25 \leq \mathbf{L} \leq 100$.
The plaintext contains each English alphabet letter at least once.

Test set 1 (Visible)

$101 \leq \mathbf{N} \leq 10000$.

Test set 2 (Hidden)

$101 \leq \mathbf{N} \leq 10^{100}$.

Sample

Sample Input

```
2
103 31
217 1891 4819 2291 2987 3811
1739 2491 4717 445 65 1079 8383
5353 901 187 649 1003 697 3239
7663 291 123 779 1007 3551 1943
2117 1679 989 3053
10000 25
3292937 175597 18779 50429
375469 1651121 2102 3722
2376497 611683 489059 2328901
3150061 829981 421301 76409
38477 291931 730241 959821
1664197 3057407 4267589 4729181
5335543
```

Sample Output

```
Case #1:
CJQUIZKNOWBEVYOFDPFLUXALGORITHMS
Case #2:
SUBDERMATOGLYPHICFJKNQVWXZ
```

Code Jam 2019 - Qualification Round

Dat Bae

Problem

A research consortium has built a new database system for their new data center. The database is made up of one master computer and **N** worker computers, which are given IDs from 0 to **N**-1. Each worker stores exactly one bit of information... which seems rather wasteful, but this is very important data!

You have been hired to evaluate the following instruction for the database:

- `TEST_STORE` <bits>: The master reads in <bits>, which is a string of **N** bits, and sends the i-th bit to the i-th worker for storage. The master will then read the bits back from the workers and return them to the user, in the same order in which they were read in.

During normal operation, `TEST_STORE` should return the same string of bits that it read in, but unfortunately, **B** of the workers are broken!

The broken workers are correctly able to store the bits given to them, but whenever the master tries to read from a broken worker, no bit is returned. This causes the `TEST_STORE` operation to return only **N**-**B** bits, which are the bits stored on the non-broken workers (in ascending order of their IDs). For example, suppose **N** = 5 and the 0th and 3rd workers are broken (so **B** = 2). Then:

- `TEST_STORE 01101` returns `111`.
- `TEST_STORE 00110` returns `010`.
- `TEST_STORE 01010` returns `100`.
- `TEST_STORE 11010` also returns `100`.

For security reasons, the database is hidden in an underground mountain vault, so calls to `TEST_STORE` take a very long time. You have been tasked with working out which workers are broken using at most **F** calls to `TEST_STORE`.

Input and output

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

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

For each test case, your program will first read a single line containing three integers **N**, **B**, and **F**, indicating the number of workers, the number of broken workers, and the number of lines you may send (as described below).

Then you may send the judge up to **F** lines, each containing a string of exactly **N** characters, each either `0` or `1`. Each time you send a line, the judge will check that you have not made more than **F** calls. If you have, the judge will send you a single line containing a single `-1`, and then finish all communication and wait for your program to finish. Otherwise, the judge will send a string of length **N**-**B**: the string returned by `TEST_STORE`, as described above.

Once your program knows the index of the **B** broken workers, it can finish the test case by sending **B** space-separated integers: the IDs of the broken workers, in sorted order. This does not count as one of your **F** calls.

If the **B** integers are not exactly the IDs of the **B** broken workers, you will receive a Wrong Answer verdict, and the judge will send a single line containing `-1`, and then no additional communication. If

your answer was correct, the judge will send a single line with `1`, followed by the line that begins the next test case (or exit, if that was the last test case).

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 1024$.
$1 \le \mathbf{B} \le \min(15, \mathbf{N}-1)$.

Test set 1 (Visible)

F = 10.

Test set 2 (Hidden)

F = 5.

Testing Tool

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

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

Download testing tool

Sample Interaction

The following interaction meets the limits for Test set 1.

```
t = readline_int()            // Reads 2 into t
n, b, f = readline_int_list()  // Reads 5, 2, 10 into n, b, f
printline 01101 to stdout     // The next four outputs match the example in
                              // the problem statement.
flush stdout
response = readline_str()     // Reads 111 into response. (At this point, we
                              // could determine the answer; the remaining
                              // queries are just examples!)
printline 00110 to stdout
flush stdout
response = readline_str()     // Reads 010 into response
printline 01010 to stdout
flush stdout
response = readline_str()     // Reads 100 into response
printline 11010 to stdout
flush stdout
response = readline_str()     // Reads 100 into response
printline 0 3 to stdout       // Guesses the answer. Notice that we were
                              // not required to use all 10 of our allowed
                              // queries.
flush stdout
```

```
verdict = readline_int()     // Reads 1 into verdict. We got that test case
                             // right!
n, b, f = readline_int_list()  // Reads 2, 1, 10 into n, b, f.
printline 01 to stdout       // 01 is a query, not a guess at the final
                             // answer (if we wanted to guess that just
                             // worker 1 were broken, we would have to
                             // send 1 as we do below)
flush stdout
response = readline_str()    // Reads 1 into response.
printline 1 to stdout        // Makes a (bad) wild guess.
verdict = readline_str()     // Reads -1 into verdict.
exit                         // exits to avoid an ambiguous TLE error
```