Faketorial Hashing

Are you familiar with polynomial hashing? If you are not, all the better! You don't need to know what polynomial hashing is, the world is better off without it. I hate polynomial hashing so much that I found a new way to hash strings. It is called the *Faketorial Hashing*.

First, let's define a function, *ord(ch)* = the position of *ch* in the alphabet + 1, where *ch* can be any lowercase letter. So, *ord(a) = 2, ord(b) = 3, ord(c) = 4, … ord(z) = 27*.

Let *fact(x)* be *x!* or the factorial of x. A few examples, *fact(1) = 1, fact(2) = 2, fact(3) = 6, fact(4) = 24, fact(5) = 120*, etc. Given a string **S** of length **N**, consisting of lowercase letters only, the *Faketorial Hashing* of S, is defined as below:

fake_hash(S) = fact(ord(S[0])) × fact(ord(S[1])) × fact(ord(S[2])) × …… × fact(ord(S[N - 1]))

In other words, it is the product of the factorial of the **ord()** value of all the characters in **S** (That's right, no modulus! Unlike the lame polynomial hashing). Not only that we have a new hashing mechanism in place, but we would also like to crack this now. Given a string **S_1** consisting of lowercase letters only, your task is to find a different string **S_2** consisting of lowercase letters, such that, **fake_hash(S_1) = fake_hash(S_2)** and **$S_1 \neq S_2$**.

If there are multiple possible choices for **S_2**, you need to find the **lexicographically smallest** one, or output the word *"Impossible"* without quotes, if it is not possible to find such a string.

Input

The first line contains an integer **T**, denoting the number of test cases. Each test case contains the string **S_1** consisting of lowercase letters (a-z) only.

Constraints

- **$1 \leq T \leq 3000$**
- **$1 \leq |S_1| \leq 30$**

Except for the sample, the following constraints will hold:

- **$1 \leq |S_1| \leq 5$, for 90% of the test cases**
- **$1 \leq |S_1| \leq 15$, for 99% of the test cases**

Output

For each test case, output the case number followed by the required output. Please refer to the sample input/output section for the precise format.

Example

Input:

```
10
tourist
petr
mnbvmar
bmerry
xellos
sevenkplus
dragoon
zzz
snapdragon
zosovoghisktwnopqrstuvwxyzoos
```

Output:

```
Case 1: aaaaabbdnstttu
Case 2: aqst
Case 3: abmmnrv
Case 4: aaabbnrry
Case 5: aaaaaaadddlnuz
Case 6: aaaaaaabbddddnquuuz
Case 7: aaaaaaaaaaaaabdnnnt
Case 8: Impossible
Case 9: aaaaaaaaaabdnnnpst
Case 10: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaffffjnnnnqqttttuuuuxxzzzzzz
```