CoE 164

Computing Platforms

Machine Exercise 01

Academic Period: 2nd Semester AY 2020-2021
Workload: 3 hours
Synopsis: Run-length message decoding
Submission Platform: Google Forms

Description

It was 1995 in the Philippines, and dial-up internet has just made its way into the common folk. With around 64 kbps internet, you can now access BBS and ICQ to read information and chat with people all over the world. You use a command line to access these platforms and you have to pay for your internet usage by the hour. On the other hand, an unlimited 128 kbps plan would cost you around ₱10k per month! Since you are not rich enough to pay for that monthly plan, you can only afford to pay for dial-up cards, which enable you to use a cumulative of 20 hours of internet connectivity. Additionally, you can only use up to 100 MB of data transfers every hour, with such a limit called the *data cap*. Once exceeded, the internet connection speed will be throttled down to be unusable until the next hour.

As a person with a lot of friends across the country and the world, you keep constant communication with them through ICQ. However, these constraints prove to be, well, too restrictive. You set out to maximize the data cap imposed on you every hour by formulating your own scheme to compress the messages sent through this platform to you.

You opted for a simple compression scheme named *run-length encoding*, which encodes the character and the amount of times it appears consecutively in a message. Each

character in the message is first read, and an accumulator counts the number of times a character is encountered consecutively. Once a new character, or no same character has been encountered, the amount in the accumulator and the characters themselves are printed in that order. Running the algorithm across the whole message results in a compressed message. For example, if there is a message “aabc”, then its compressed equivalent would be “2abc”. Notice that the number “2” is written before the character (a) that has that amount of repetitions. Since we use numbers to encode the occurrence of each character, we will naively assume that there will be no numbers in a decompressed message.

All looks well and easy under the simple world of ASCII, where characters can be encoded as an index of 7 bits, and the characters there consist only of the usual alphabet letters, numbers, and other symbols found in English. However, due to your friends being international, they for sure will use their own writing systems to send messages. Fortunately, the UTF-8 standard has just been introduced more than a year ago, in which characters can be encoded as an index between one and four bytes on a character plane. Also, to test whether your compression algorithm can truly save you some data, you also want to know the *compression ratio*, which can be interpreted as the number of times the data has been reduced. In terms of formulas, it would be $c = \frac{len(uncompressed)}{len(compressed)}$. To simplify things, you just wanted to know the rounded-off (round half up) integer value of the compression ratio for each message.

You have already finished the encoding algorithm and are now going to write a program that will decode the compressed messages. As these programs will be sent out to your friends, you would like to make sure that this program can decompress correctly and efficiently given that you need to be able to handle UTF-8 characters.

Input

The input to your program starts with the number T on a line indicating the number of messages to expect. Then, each T compressed messages m_i are written on each line i.

Output

The output should consist of T lines, with each line i consisting of the compression ratio c as a rounded-off (round half up) nonnegative integer, and the uncompressed message M_i.

Example

Input

```
5
10a6b5c
abcdefabcdef
2ĉa2ĉaĥ2ĵs
nña2ába3ß
```

そ8れな5あ

Output
3 aaaaaaaaaabbbbbbccccc
1 abcdefabcdef
1 ĉĉaĉĉaĥĵĵs
1 nñaáábaßßß
3 それれれれれれれれなあああああ

Additional Description/Requirements

Your program can only support the following specifications:
$T \leq 100000$
$1 \leq len(m) \leq 100000$
$1 \leq len(M) \leq 1000000$
$\forall x \in M \cup m, x \in \{UTF-8\}$

The UTF-8 set is the set of (less than) 1112064 codepoints that correspond to a single character. However, we limit ourselves to those characters commonly used by your friends, so it will be way smaller than that. Whitespaces are also included in UTF-8!

Writing a journal for this program hasn't crossed your mind the slightest. However, because you believe that you can easily forget things and there may be a chance in the future when you revisit the code, you try your best to write self-documenting code. This means that, among other things, you write your variables throughout your code to more clearly label their purposes and add comments to code that can be hard to understand.

Having been involved in the internet scene, you had connections with the Wayback Machine™, and was informed that the platform would be released in 1996. However, you appear to also have access to the Future Web Archive™, where you can see the future of the internet. Because of that, you can use any programming language of the future to develop your decompression program. Also, to ensure the quality of your program, you have decided to send your program (as a single source code file; in TXT if the system does not support the file extension) through this futuristic platform named Google Forms, in which a third-party from the future will check your code, and resend it with an assessment.

Grading Rubric

5% Input handling - able to read the input specifications and ASCII characters
5% Output handling - able to write ASCII characters
10% Input handling - able to read UTF-8 characters
10% Output handling - able to write UTF-8 characters
25% Algorithm to compute the integer compression ratio
45% Algorithm to decode the compressed message
5% Self-documenting code (optional)