Shortening in the Real World

Problem Statement

You are part of a large gaming firm which is looking forward to offer online gaming competitions through live streaming. This means a lot of users will be looking to share their channels links through multiple social media networks (e.g: twitter, facebook, etc.)

Your company wants to implement URL redirection, where you will provide a URL on your website that when requested will redirect browsers to the links provided by users. Due to the burst of popularity of e-sport sites, your company did not plan for the large demand for URLs. The encoded URLs given by the system are very long and difficult to handle by regular users, reducing the usability of your system and service.

Your boss comes to you in a panic, hoping that you will be capable of solving this simple but critical task and help the company help users to handle URLs in an easy and comfortable manner. As a good software engineer with some basic understanding of Computer Science, you devise a way to encode the original encoded URL into a shorter form through a hash function. As you know, there is always the risk that your hash function will have a collision, but it is a good start.

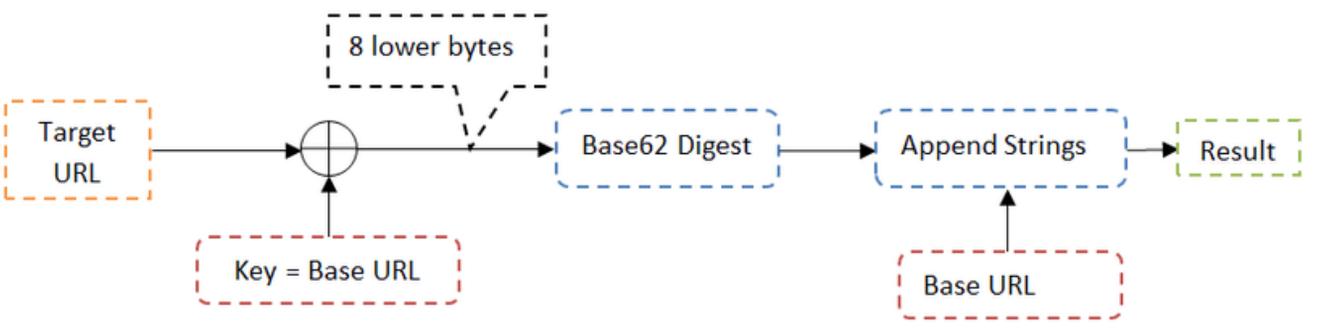
The hash function takes as input the base URL of your company and the target URL you wish to encode. (For an example of this process, see the *Explanation* portion of the sample input, below). As shown in the figure below, the algorithm proceeds in the following steps.

1) An xor cipher is applied to the target URL, using the base URL of your company as a repeating key. Here we perform a bitwise exclusive-or between each byte of the target URL and the base URL. If the target URL is shorter than the base URL of your company, you would truncate the base URL so that the lengths are equal. If the base URL of your company is shorter than the target URL, you would repeat the base URL as many times as needed to make the lengths equal.

2) Take the last 8 bytes of the output from step 1, and convert this to the corresponding unsigned integer. (See the example below for more details.)

3) Encode this unsigned integer using Base62 encoding. In this encoding, you convert the integer to a base 62 number, where the digits 0-9 represent values 0-9, lowercase letters a-z represent values 10-35, and capital letters A-Z represent values 36-61.

4) The encoded url consists of the base URL, a backslash, and the base 62 encoded value produced in the previous step.



Notes:

- You should process the URLs using UTF-8 encoding.

- The URL to be encoded will always be at least 8 characters long.
- The base 62 number should not be padded with extra zeros. For example, the number "62" should be represented as "10", not as "010" or "0010" or "00010". The number "0" would be represented as "0".
- The base URL will never end with a backslash.

Input Format

The first line of input will contain the base URL of your company.

The second line will contain a number *n* (1 <= *n* <= 1000) which will indicate the number of URLs you will need to encode.

The following *n* lines will contain target URLs to encode.

Output Format

The output should be *n* lines, where each line will correspond to an encoded URL.

Sample Input

```
http://www.ieee.com
2
http://www.ieee.org/xtreme
http://www.ieee.org/membership_services/membership/young_professionals/index.html
```

Sample Output

```
http://www.ieee.com/SHPQ4gzW1Y
http://www.ieee.com/Btazwa9mke
```

Explanation

Consider the first target URL to be encoded: `http://www.ieee.org/xtreme`.

We start by finding the UTF-8 encoding of the base URL `http://www.ieee.com` and the target URL.

```
target URL = [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69, 0x65, 0x65, 0x65, 0x2e, 0x6f, 0x72, 0x67, 0x2f,
0x78, 0x74, 0x72, 0x65, 0x6d, 0x65]
base URL   = [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69, 0x65, 0x65, 0x65, 0x2e, 0x63, 0x6f, 0x6d]
```

Now we apply the exclusive-or cipher, repeating the bytes in the base URL so that the two strings are the same length:

```
   [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69, 0x65, 0x65, 0x65, 0x2e, 0x6f, 0x72, 0x67, 0x2f, 0x78, 0x74,
0x72, 0x65, 0x6d, 0x65]
xor
   [0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x69, 0x65, 0x65, 0x65, 0x2e, 0x63, 0x6f, 0x6d, 0x68, 0x74,
0x74, 0x70, 0x3a, 0x2f, 0x2f]
=
   [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc, 0x1d, 0x0a, 0x47, 0x0c,
0x00, 0x02, 0x5f, 0x42, 0x4a]
```

Next we take the last 8 bytes of this number and convert it to an unsigned integer:

0x0a470c00025f424a = 740,573,857,905,066,570

Next we convert this number from base 10 to base 62 to get: `SHPQ4gzW1Y`

We then append this number to the base URL to produce the output: `http://www.ieee.com/SHPQ4gzW1Y`