Daily Coding Problem #55

Problem

This problem was asked by Microsoft.

Implement a URL shortener with the following methods:

- `shorten(url)`, which shortens the url into a six-character alphanumeric string, such as `zLg6wl`.
- `restore(short)`, which expands the shortened string into the original url. If no such shortened string exists, return null.

Hint: What if we enter the same URL twice?

Solution

Clearly, we need a random string generator for this problem. If you're in an interview and you don't know how to generate a random string by heart, that's fine -- you can just assume you have access to a function that generate N random characters. In this case, we'll create a helper function called `_generate_short` that does it for us.

The idea for this problem is to generate a shortened url and store it in a dictionary where the shortened url is the key and the actual url is the value. Then, when retrieving the actual url we can just look it up in the dictionary.

However, we need to be careful in that we don't accidentally overwrite an existing entry when shortening a url. So what we'll do is continuously generate urls until we find one that doesn't already exist, and then use that one. We do that in the helper function `generate_unused_short`.

```
import random
import string

class URLShortener:
    def __init__(self):
        self.short_to_url = {}

    def _generate_short(self):
        return ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(6))

    def _generate_unused_short(self):
        t = self._generate_short()
        while t in self.short_to_url:
            t = self._generate_short()
        return t

    def shorten(self, url):
        short = self._generate_unused_short()
        self.short_to_url[short] = url
        return short
```

```
    def restore(self, short):
        return self.short_to_url.get(short, None)
```

We can improve this a bit. What if we shorten the same url twice? We could potentially re-use the existing shortened url, but we don't know how to access it without querying all values in our dict!

So we can create a second dict that maps urls to shortened urls and update that appropriately. When we see a url we've seen before, we can just then just re-use that shortened url.

```
import random
import string

class URLShortener:
    def __init__(self):
        self.short_to_url = {}
        self.url_to_short = {}

    def _generate_short(self):
        return ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(6))

    def _generate_unused_short(self):
        t = self._generate_short()
        while t in self.short_to_url:
            t = self._generate_short()
        return t

    def shorten(self, url):
        short = self._generate_unused_short()
        if url in self.url_to_short:
            return self.url_to_short[url]
```

```
        self.short_to_url[short] = url
        self.url_to_short[url] = short
        return short

    def restore(self, short):
        return self.short_to_url.get(short, None)
```