Encode and Decode TinyURL

> Note: This is a companion problem to the System Design problem: Design TinyURL.

TinyURL is a URL shortening service where you enter a URL such as `https://leetcode.com/problems/design-tinyurl` and it returns a short URL such as `http://tinyurl.com/4e9iAk`.

Design the `encode` and `decode` methods for the TinyURL service. There is no restriction on how your encode/decode algorithm should work. You just need to ensure that a URL can be encoded to a tiny URL and the tiny URL can be decoded to the original URL.

Solution 1

My first solution produces short URLs like `http://tinyurl.com/0`, `http://tinyurl.com/1`, etc, in that order.

```
class Codec:

    def __init__(self):
        self.urls = []

    def encode(self, longUrl):
        self.urls.append(longUrl)
        return 'http://tinyurl.com/' + str(len(self.urls) - 1)

    def decode(self, shortUrl):
        return self.urls[int(shortUrl.split('/')[-1])]
```

Using increasing numbers as codes like that is simple but has some disadvantages, which the below solution fixes:

- If I'm asked to encode the same long URL several times, it will get several entries. That wastes codes and memory.
- People can find out how many URLs have already been encoded. Not sure I want them to know.
- People might try to get special numbers by spamming me with repeated requests shortly before their desired number comes up.
- Only using digits means the codes can grow unnecessarily large. Only offers a million codes with length 6 (or smaller). Using six digits or lower or upper case letters would offer $(10+26*2)^6$ = 56,800,235,584 codes with length 6.

The following solution doesn't have these problems. It produces short URLs like `http://tinyurl.com/KtLa2U`, using a random code of six digits or letters. If a long URL is already known, the existing short URL is used and no new entry is generated.

```
class Codec:

    alphabet = string.ascii_letters + '0123456789'

    def __init__(self):
        self.url2code = {}
        self.code2url = {}

    def encode(self, longUrl):
        while longUrl not in self.url2code:
            code = ''.join(random.choice(Codec.alphabet) for _ in range(6))
            if code not in self.code2url:
                self.code2url[code] = longUrl
                self.url2code[longUrl] = code
        return 'http://tinyurl.com/' + self.url2code[longUrl]

    def decode(self, shortUrl):
        return self.code2url[shortUrl[-6:]]
```

It's possible that a randomly generated code has already been generated before. In that case, another random code is generated instead. Repeat until we have a code that's not already in use. How long can this take? Well, even if we get up to using half of the code space, which is a whopping $62^6/2$ = 28,400,117,792 entries, then each code has a 50% chance of not having appeared yet. So the expected/average number of attempts is 2, and for example only one in a billion URLs takes more than 30 attempts. And if we ever get to an even larger number of entries and this does become a problem, then we can just use length 7. We'd need to anyway, as we'd be running out of available codes.

written by StefanPochmann original link here

Solution 2

```
class Solution {
public:

    // Encodes a URL to a shortened URL.
    string encode(string longUrl) {
        return longUrl;
    }

    // Decodes a shortened URL to its original URL.
    string decode(string shortUrl) {
        return shortUrl;
    }
};
```

written by xplorld original link here

Solution 3

below is the tiny url solution in java, also this is the similar method in industry. In industry, most of shorten url service is by database, one auto increasing long number as primary key. whenever a long url need to be shorten, append to the database, and return the primary key number. (the database is very easy to distribute to multiple machine like HBase, or even you can use the raw file system to store data and improve performance by shard and replica).
Note, it's meaningless to promise the same long url to be shorten as the same short url. if you do the promise and use something like hash to check existing, the benefit is must less than the cost.
Note: if you want the shorted url contains '0-9a-zA-Z' instead of '0-9', then you need to use 62 number system, not 10 number system(decimal) to convert the primary key number. like 123->'123' in decimal, 123->'1Z' in 62 number system (or '0001Z' for align).

```
public class Codec {
    List<String> urls = new ArrayList<String>();
    // Encodes a URL to a shortened URL.
    public String encode(String longUrl) {
        urls.add(longUrl);
        return String.valueOf(urls.size()-1);
    }

    // Decodes a shortened URL to its original URL.
    public String decode(String shortUrl) {
        int index = Integer.valueOf(shortUrl);
        return (index<urls.size())?urls.get(index):"";
    }
}
```

written by qiu5 original link here