Lu Chang & Zhang Mengjiao

# LeetCode Solutions

***First Edition***

# Preface

This project is aimed to accompany my girlfriend @MengjiaoZhang to learn git, programming skills and algorithms.

Here, I want to thank LeetCode providing these problems. It will be better if all problems can be accessed without subscribing. (ಡωಡ)hiahiahia

# Contents

# Chapter 1

# Solutions for Algorithms

"*Perhaps the most important principle for the good algorithm designer is to refuse to be content.*"

— Alfred V. Aho

# 535. Encode and Decode TinyURL

## Difficulty

Medium

## Tags

Cryptology

## Description

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.

## Analysis

This is an open problem where numerous solutions can be applied. We can even keep the original url as encode and decode, although it is meaningless.

We should to pay attention to these limitations:

- Correctness: We must make sure that the decoded url is the same as the original url.
- Uniqueness: Each url must have an unique encoded url, and an encoded url must be decoded to a single url.
- Simplicity: The aim to encode a url is to make it easy to share or write, so we need to make the encoded url as simple as possible.

We can use current popular encode/decode algorithms such as `AES`, `DES`, but in this problem, we just design a simpler algorithm to encode and

decode a url.

```
function encode(url)
    return hex(current number of urls in hash table)
end

function decode(encoded_url)
    return (hash table).find(encoded_url)
end
```

## Solution

### C++

```cpp
typedef unordered_map<string, string> Urlmap;

class Solution {
public:

    Urlmap urlmap;

    // Encodes a URL to a shortened URL.
    string encode(string longUrl) {
        size_t size = urlmap.size();
        stringstream encoded;
        encoded << hex << size;
        string encoded_url = encoded.str();
        urlmap.insert(make_pair(encoded_url, longUrl));
        return encoded_url;
    }

    // Decodes a shortened URL to its original URL.
    string decode(string shortUrl) {
    Urlmap::iterator it = urlmap.find(shortUrl);
    if (it == urlmap.end()) return NULL;
        return it->second;
    }
};
```

# 771. Jewels and Stones

## Difficulty

Easy

## Tags

Hash Table

## Description

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

**Example 1**

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

**Example 2**

```
Input: J = "z", S = "ZZ"
Output: 0
```

## Note

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

## Analysis

This is an easy problem. All we need to do is to verify wether each letter in `S` exists in `J`.

Therefore, we can use a hash set to store `J`. Specifically, `J` is composed of letters (lower or upper) only, so we can use an array of `char` to store each letter in . After that, we only need to iterate `S` to calculate the number of jewels.

We assume the length of is $m$, the length of `S` is $n$, then

- Time complexity: $\mathcal{O}(m+n)$
- Space complexity: $\mathcal{O}(1)$ (256 ASCII chars)

## Solution

### C

```c
int numJewelsInStones(char* J, char* S) {
    char j_letters[256] = { 0 };  // initialize an array to store letters in J
    char c;
    for (int i = 0; (c = J[i]) != '\0'; ++i) {
        j_letters[c] = 1;  // set j_letters[c] = 1 means c appears in J
    }
    int jewel_number = 0;  // number of jewels
    for (int i = 0; (c = S[i]) != '\0'; ++i) {
        jewel_number += j_letters[c];  // if c appears in J, then we add a
        ↪  number
    }
    return jewel_number;
}
```

# Indexes of Solutions for Algorithms

# Tags of Solutions for Algorithms

## Cryptology

535. Encode and Decode TinyURL

## Hash Table

771. Jewels and Stones

# Chapter 2

# Solutions for Databases