Tiny URL

http://stackoverflow.com/questions/742013/how-to-code-a-url-shortener
http://n00tc0d3r.blogspot.com/

What is tinyurl?

tinyurl is a URL service that users enter a long URL and then the service return a shorter and _unique_ url such as "http://tiny.me/5ie0V2". **The highlight part can be any string with 6 letters containing [*0-9, a-z, A-Z*]. That is, *62^6 ~= 56.8* billions unique strings.**

How it works?

On Single Machine

Suppose we have a database which contains three columns:
- id (auto increment)
- actual url
- shorten url

Intuitively, we can design a hash function that maps the actual url to shorten url. But string to string mapping is not easy to compute.

Notice that in the database, each record has a unique id associated with it. What if we **convert the id to a shorten url**?

Basically, we need a **Bijective function (双射函数)** f(x) = y such that

- Each x must be associated with one and only one y;
- Each y must be associated with one and only one x.

In our case, the set of x's are integers while the set of y's are 6-letter-long strings. Actually, each 6-letter-long string can be considered as a number too, a 62-base numeric, if we map each distinct character to a number,
e.g. *0-0,*
...,
9-9,
10-a,
11-b,
...,
35-z,
36-A,
...,
61-Z.

Then, the problem becomes Base Conversion (基地转换) problem which is bijection (if not overflowed :).

A short photo id is a base58 conversion of the photo id. Base58 is like base62 [0-9a-zA-Z] with some characters removed to make it less confusing when printed. (namely 0, O, I, and l).
http://www.flickr.com/groups/api/discuss/72157616713786392/

```
import java.util.*;
import java.lang.*;

class Solution{
    public String toShortURL(long id, int base, HashMap<Integer, Character> map){
        StringBuilder sb = new StringBuilder();
        while(id > 0){
            int digit = (int)id % base;
            System.out.println(digit);
            sb.append(map.get(digit));
            id /= base;
        }
        /*
        while(sb.length() < 6){
            sb.append('0');
        }
        */
        return sb.reverse().toString();
    }

    public long toID(String shortUrl, int base, HashMap<Character, Integer> map){
        StringBuilder sb = new StringBuilder(shortUrl);
        long id = 0L;
        for(int i = 0; i < sb.length(); i ++){
            System.out.println(map.get(sb.charAt(i)) + " " + (sb.length() - 1 - i));
            id += map.get(sb.charAt(i)) * Math.pow(base, sb.length() - 1 - i);
        }

        return id;
    }
}

class ConversionMap{
    private HashMap<Integer, Character> hm = new HashMap<Integer, Character>();
    private HashMap<Character, Integer> hm_back = new HashMap<Character, Integer>();

    public ConversionMap(){
            int index = 0;
            for(int i = '0'; i <= 'z'; i ++){
                    if(Character.isDigit(i) ||    // Character.isAlphabetic(i) only exist in Java 7
                            (i >= 'A' && i <= 'Z') ||
                            (i >= 'a' && i <= 'z') ){
                            hm.put(index, (char)i);
                            hm_back.put((char)i, index);
                            index ++;
                    }
            }
    }

    public HashMap getConversionMap(){
            return hm;
    }

    public HashMap getReversionMap(){
            return hm_back;
    }
}
public class TinyURL{
    public static void main(String[] args){
            // Long.MAX_VALUE  = 2^64 - 1
            ConversionMap map = new ConversionMap();
            Solution s = new Solution();

            System.out.println(s.toShortURL(10009999L, 62, map.getConversionMap()));
            /*
                    在定义数字常量的时候， 默认的都是INT型，LONG开的要这样定义

                    long l = 78541258611L;  <-   在最后加上'L'
            */
            System.out.println(s.toID("g03b", 62, map.getReversionMap()));
    }
}
```

For each input long url, the corresponding id is auto generated (in O(1) time). The base conversion algorithm runs in O(*k*) time where *k* is the number of digits (i.e. *k=6*).

On Multiple Machine

Suppose the service gets more and more traffic and thus we need to distributed data onto multiple servers.

We can use Distributed Database. But maintenance for such a db would be much more complicated (replicate data across servers, sync among servers to get a unique id, etc.).

Alternatively, we can use Distributed Key-Value Datastore.
Some distributed datastore (e.g. Amazon's Dynamo) uses Consistent Hashing to hash servers and inputs into integers and locate the corresponding server using the hash value of the input. We can apply base conversion algorithm on the hash value of the input.

The basic process can be:
<u>Insert</u>

1. Hash an input long url into a single integer;
2. Locate a server on the ring and store the key--longUrl on the server;
3. Compute the shorten url using base conversion (from 10-base to 62-base) and return it to the user.

<u>Retrieve</u>

1. Convert the shorten url back to the key using base conversion (from 62-base to 10-base);
2. Locate the server containing that key and return the longUrl.

Further Readings

- StackOverflow: How to code a url shortener