# 【第十四课】哈希表与布隆过滤器

**彩蛋题目：开课吧OJ118**

## 705. 设计哈希集合

思路： 1.设哈希表的大小为 base，则可以设计一个简单的哈希函数：hash(x) = x mod base。

2.我们开辟一个大小为 base的数组，数组的每个位置是一个链表。当计算出哈希值之后，就插入到对应位置的链表当中。

3.由于我们使用整数除法作为哈希函数，为了尽可能避免冲突，应当将 base 取为一个质数。在这里，我们取 base = 769。

```
var MyHashSet = function() {
    this.BASE = 100;
    this.data = new Array(this.BASE).fill(0).map(() => new Array());
};

MyHashSet.prototype.add = function(key) {
    const h = this.hash(key);
    for (const element of this.data[h]) {
        if (element === key) {
            return;
        }
    }
    this.data[h].push(key);
};

MyHashSet.prototype.remove = function(key) {
    const h = this.hash(key);
    const it = this.data[h];
    for (let i = 0; i < it.length; ++i) {
        if (it[i] === key) {
            it.splice(i, 1);
            return;
        }
    }
};

MyHashSet.prototype.contains = function(key) {
    const h = this.hash(key);
    for (const element of this.data[h]) {
        if (element === key) {
            return true;
        }
    }
    return false;
};

MyHashSet.prototype.hash = function(key) {
    return key % this.BASE;
}
```

# 706. 设计哈希映射

思路： 「设计哈希映射」与「设计哈希集合」解法接近，唯一的区别在于我们存储的不是 key 而是{key,value}对，除此之外，代码都是类似的。

```
var MyHashMap = function() {
    this.BASE = 100;
    this.data = new Array(this.BASE).fill(0).map(() => new Array());
};

MyHashMap.prototype.put = function(key, value) {
    const h = this.hash(key);
    for (const it of this.data[h]) {
        if (it[0] === key) {
            it[1] = value;
            return;
        }
    }
    this.data[h].push([key, value]);
};

MyHashMap.prototype.get = function(key) {
    const h = this.hash(key);
    for (const it of this.data[h]) {
        if (it[0] === key) {
            return it[1];
        }
    }
    return -1;
};
// 思路是一样的 就是这次多传入的是一个value
MyHashMap.prototype.remove = function(key) {
    const h = this.hash(key);
    for (const it of this.data[h]) {
        if (it[0] === key) {
            const idx = this.data[h].indexOf(it);
            this.data[h].splice(idx, 1);
            return;
        }
    }
};

MyHashMap.prototype.hash = function(key) {
    // 计算得到当前哈希表的，当前待插入值的数据的对应的下标
    return key % this.BASE;
}
```

# 面试题 16.25. LRU 缓存

思路：1.LRU的缓存策略就像是维护一个队列，每次把访问的节点移动过到队列的末尾，当队列的长度大于缓存的长度，就把队列首部的元素给删除。

2.在调用get和put的时候，访问或者修改了的数据要被更新到最新访问的位置，所以会在这里处理数据的顺序。

3.其次存取的数据要能记录访问顺序，或者说要有序。

```
const LRUCache = function(capacity) {
    // 最大缓存容量
    this.capacity = parseInt(capacity, 10);
    // 数据缓存对象
    this.cache = {};
    // 键名缓存数组,还可提供key的访问时间顺序
    this.keys = [];
};

/**
 * @param {number} key
 * @return {number}
 */
LRUCache.prototype.get = function(key) {
    const idx = this.keys.indexOf(key);
    if(idx === -1) return -1;

    // 更新keys的顺序
    this.keys.push(this.keys.splice(idx, 1)[0]);
    return this.cache[key];
};

/**
 * @param {number} key
 * @param {number} value
 * @return {void}
 */
LRUCache.prototype.put = function(key, value) {
    const idx = this.keys.indexOf(key);
    if(idx !== -1){
        // 更新原有数据和访问时间顺序
        this.keys.push(this.keys.splice(idx, 1)[0]);
    }else{
        if(this.keys.length === this.capacity){ // 超出缓存容量
            // 删除缓存
            this.cache[this.keys.shift()] = null;
        }
        this.keys.push(key);
    }
    this.cache[key] = value;
};
```

# 535. TinyURL 的加密与解密

思路： 1.给了一个网址，让最后输出一个短网址；这个叫加密。给了一个短网址，让最后输出它原始网址，这个叫解密。

2.一个url生成一个6位的随机数作为key值存储在哈希表里，然后把这个随机数作为后缀放入返回的简化url

3.拿到简化url后提取出后6位随机数，去哈希表找到对应的长url即可

```
let map={};
let str="1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

function getRandom(str){
    let res="";
    for(let i=0;i<6;i++){
        res+=str[Math.floor(Math.random()*61)];
    }
    return res;
}
var encode = function(longUrl) {
    let result=getRandom(str);
    while(map[result]){
        result=getRandom(str)
    }

    map[result]=longUrl;
    return "http://tinyurl.com/"+result;
};

/**
 * Decodes a shortened URL to its original URL.
 *
 * @param {string} shortUrl
 * @return {string}
 */
var decode = function(shortUrl) {
    let key=shortUrl.slice(-6);
    return map[key];
};

/**
 * Your functions will be called as such:
 * decode(encode(url));
 */
```

# 187. 重复的DNA序列

思路： 1.用长度为 10 的窗口去截取出 子串，把它出现的次数统计在 map 中。

2.在移动窗口截取子串的过程中，一旦有子串的出现次数达到 2,就将该子串推入 ret 数组。

3.最后返回 ret 数组。

```
  var findRepeatedDnaSequences = function(s) {
    if (s.length < 11) return [];
    let n = s.length, map = new Map(), left = 0, right = 10, res= [];

    while (right <= n) {
        let cur = s.substring(left, right);

        map.set(cur, map.has(cur) ? map.get(cur) + 1 : 1);
        left++;
        right++;
    }
```

```
    for (let [k, v] of map) {
        if (v > 1) res.push(k);
    }
    return res;
};
```