哈希表hashmap

无隅

什么是哈希表

- 设计精妙、用途广泛的数据结构之一
- 拥有键值对元素的无序集合
- 键的值是唯一的，键对应的值可以通过键来获取、更新或移除
- 无论这个哈希表有多大，这些操作基本上通过常量时间的键比较就可完成

java, javascript, python,go中的HashMap

- Java: Map<A,B>

  ```
  Map<String, Integer> hm = new HashMap<>();
  ```

- javascript: Object

  ```
  var hm = {};
  ```

- python: dict

  ```
  hm = dict() or hm = {}
  ```

- go: map

  ```
  hm := make(map[string]int)
  ```

基础操作

- Insert - O(1)
- Delete - O(1)
- Find - O(1)

基础操作

Return Type	Method
Value	put(key, value)
Value	get(key)
Value	remove(key)
boolean	containsKey(key)
boolean	containsValue(value)
Set<Map.Entry<Key, Value>>	entrySet()
Set<Key>	keySet()
Collection<Value>	values()
...	size(); isEmpty(); putAll(Map<>);

遍历HashMap

```java
Iterator<Entry<String,String>> iterator = map.entrySet().iterator();
while (iterator.hasNext()) {
    Map.Entry<String,String> entry = (Map.Entry<String,String>) iterator.next();
    System.out.println("Key: " + entry.getKey() + " Value:" + entry.getValue());
}


for (Map.Entry<String, String> entry : map.entrySet()) {
    System.out.println("Key: " + entry.getKey() + " Value: " + entry.getValue());
}


for (Object key : map.keySet()) {
    System.out.println("Key: " + key.toString() + " Value: " + map.get(key));
}


map.entrySet().forEach(entry -> System.out.println("Key: " + entry.getKey()
                                    + " Value: " + entry.getValue()));
```

Hash Function哈希函数

- 哈希函数是用来将一个字符串（或任何其他类型）转化为小于哈希表大小且大于等于零的整数
- 一个好的哈希函数:

((1)) 可以尽可能少地产生冲突

((2)) 算得快

一种广泛使用的哈希函数算法是使用数值33，

假设任何字符串都是基于33的一个大整数，

比如：

hashcode("abcd") = (ascii(a) * 33 ^ 3 + ascii(b) * 33 ^ 2 + ascii(c) *33 + ascii(d)) % HASH_SIZE

= (97* 33 ^ 3 + 98 * 33 ^ 2 + 99 * 33 +100) % HASH_SIZE

= 3595978 % HASH_SIZE

其中HASH_SIZE表示哈希表的大小(可以假设一个哈希表就是一个索引0 ~ HASH_SIZE-1的数组)。

给出一个字符串作为key和一个哈希表的大小，返回这个字符串的哈希值。

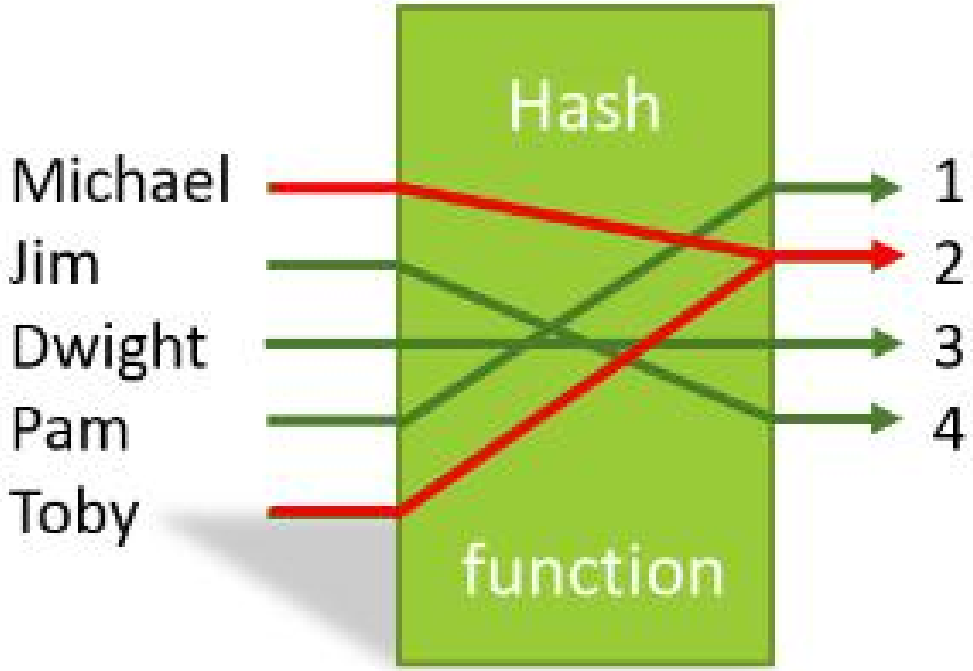
```java
public int hashCode(char[] key, int hashSize) {
    long result = 0;
    for(int i = 0; i < key.length;i++) {
        result = (result * 33 + (int)(key[i])) % hashSize;
    }
    return (int) result;
}
```

冲突

- 无论使用什么hash function，都需要考虑冲突问题
- 为啥会有冲突

  （1）有一些key会map到相同的index上

  （2）无限空间往有限空间映射



如何解决冲突

- Change index

  Open hashing

  Closed hashing

- 扩容

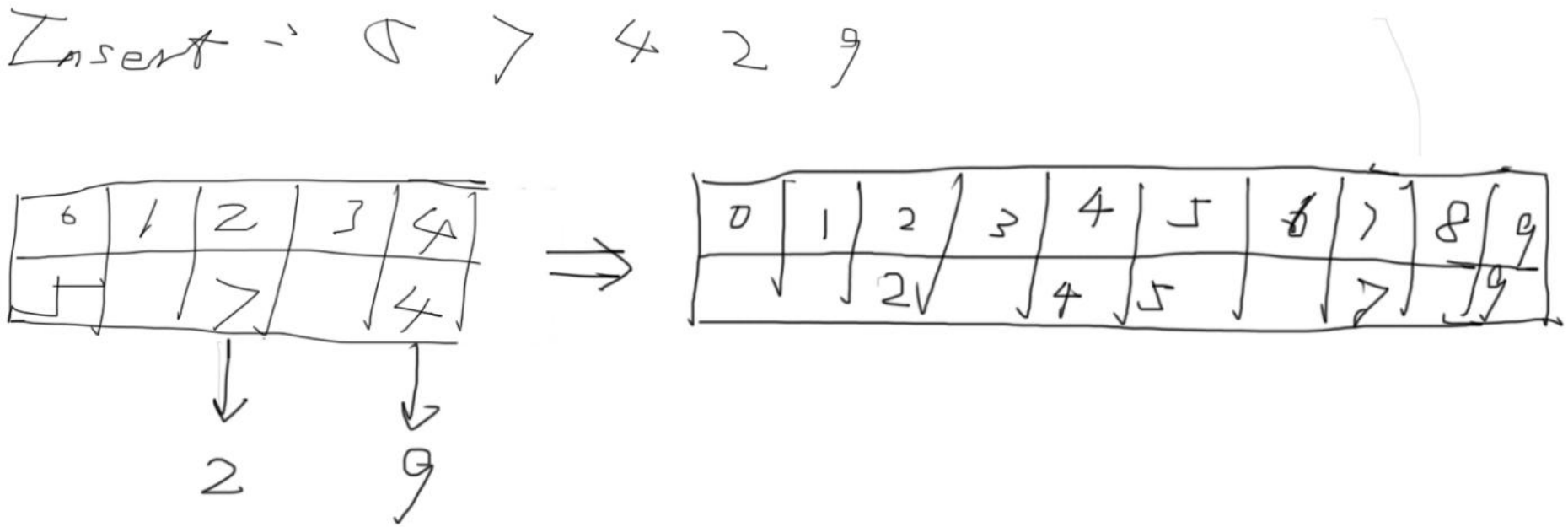
  装填因子Load factor: size/capacity

  Java: LF > 0.75, resize()

Open hashing VS Closed hashing

扩容：重哈希rehashing

- 哈希表容量的大小在一开始是不确定的。如果哈希表存储的元素太多，我们应该将哈希表容量扩大一倍，并将所有的哈希值重新安排。



```java
public ListNode[] rehashing(ListNode[] hashTable) {
    if (hashTable == null || hashTable.length == 0) {
        return null;
    }
    int capacity = hashTable.length;
    int newCapacity = capacity * 2;
    ListNode[] newHashTable = new ListNode[newCapacity];
    for (ListNode head : hashTable) {
        while (head != null) {
            int key = head.val;
            int hashcode = (key % newCapacity + newCapacity) % newCapacity;
            if (newHashTable[hashcode] != null) {
                ListNode node = newHashTable[hashcode];
                ListNode keyNode = new ListNode(key);
                while (node != null && node.next != null) {
                    node = node.next;
                }
                node.next = keyNode;
            }
            else {
                newHashTable[hashcode] = new ListNode(key);
            }
            head = head.next;
        }
    }
    return newHashTable;
}
```

Two Sum

给定一个整数数组和一个目标值，找出数组中和为目标值的两个数。

你可以假设每个输入只对应一种答案，且同样的元素不能被重复利用。

示例:

给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9 所以返回 [0, 1]

https://leetcode-cn.com/problems/two-sum/description/

```java
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        map.put(nums[i], i);
    }
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement) && map.get(complement) != i) {
            return new int[] { i, map.get(complement) };
        }
    }
    return new int[];
```

单词模式

给定一种 pattern(模式) 和一个字符串 str ，判断 str 是否遵循相同的模式。

这里的遵循指完全匹配，例如， pattern 里的每个字母和字符串 str 中的每个非空单词之间存在着双向连接的对应模式。

示例1: 输入: pattern = "abba", str = "dog cat cat dog" 输出: true

示例 2: 输入:pattern = "abba", str = "dog cat cat fish" 输出: false

示例 3: 输入: pattern = "aaaa", str = "dog cat cat dog" 输出: false

示例 4: 输入: pattern = "abba", str = "dog dog dog dog" 输出: false

说明:

```java
public boolean wordPattern(String pattern, String str) {
    String[] words = str.split(" ");
    if (words.length != pattern.length()) {
        return false;
    }
    Map<Character, String> map = new HashMap<Character, String>();
    for(int i = 0; i < words.length; i++) {
        char c = pattern.charAt(i);
        if (map.containsKey(c)) {
            if (!map.get(c).equals(words[i])) {
                return false;
            }
        }
        else {
            if (map.containsValue(words[i])) {
                return false;
            }
            map.put(c, words[i]);
        }
    }
    return true;

}
```

字母异位词分组

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

示例:

输入: ["eat", "tea", "tan", "ate", "nat", "bat"], 输出: [["ate","eat","tea"], ["nat","tan"], ["bat"]]说明:

所有输入均为小写字母。

不考虑答案输出的顺序。

https://leetcode-cn.com/problems/group-anagrams/description/

```java
public List<List<String>> groupAnagrams(String[] strs) {
    List<List<String>> result = new ArrayList<List<String>>();
    HashMap<String, List<String>> map = new HashMap<String, List<String>>();
    if (strs == null || strs.length == 0) {
        return result;
    }
    for (String str : strs) {
        char[] chars = str.toCharArray();
        Arrays.sort(chars);
        String newStr = new String(chars);
        if (!map.containsKey(newStr)) {
            List<String> list = new ArrayList<String>();
            list.add(str);
            map.put(newStr, list);
        }
        else {
            List<String> list = map.get(newStr);
            list.add(str);
            map.put(newStr, list);
        }
    }
    for (Map.Entry<String, List<String>> entry: map.entrySet()) {
        List<String> stringresult = entry.getValue();
        Collections.sort(stringresult);
        result.add(stringresult);
    }
    return result;
}
```

作业

- 同构字符串

https://leetcode-cn.com/problems/isomorphic-strings/description/

- 分数到小数

https://leetcode-cn.com/problems/fraction-to-recurring-decimal/description/