一.概述

哈希表是一种使用哈希函数组织数据，支持快速插入和搜索的数据结构。

两种类型的哈希表：

- 哈希集合：存储非重复值
- 哈希映射：存储(key,value)键值对

二.设计哈希表

1.哈希表的原理

https://leetcode-cn.com/leetbook/read/hash-table/xht3is/

2.设计哈希表的关键

- 哈希函数：取决于键值数目和桶的数量，一般来说，桶的数量越多，桶的容量越小，需要在数量和容量之间进行权衡。

- 冲突解决

  冲突解决算法应该解决以下几个问题：

  1。如何组织在同一个桶中的值?
  2。如果为同一个桶分配了太多的值，该怎么办?
  3。如何在特定的桶中搜索目标值?
  根据我们的哈希函数，这些问题与桶的容量和可能映射到同一个桶的键的数目有关。

3.设计哈希集合

- https://leetcode-cn.com/problems/design-hashset/

```
// 采用拉链法的哈希表
struct List{
    int val;
    List* next;
};
class MyHashSet {
public:
    List* hashSet = nullptr;
    MyHashSet() {
        hashSet = new List[1111];
        for(int i=0;i<1111;i++){
            hashSet[i].next = nullptr;
        }
    }

    void add(int key) {
        int index = key%1111;
        List* tempList = new List;
        tempList->val = key;
```

```
        tempList->next = hashSet[index].next;
        hashSet[index].next = tempList;
    }

    void remove(int key) {
        int index = key%1111;
        List* pre = &hashSet[index];
        List* ptr = hashSet[index].next;
        while(ptr!=nullptr){
            if(ptr->val==key){
                List* temp = ptr;
                ptr = ptr->next;
                pre->next = temp->next;
                delete temp;
            }else{
                pre = ptr;
                ptr = ptr->next;
            }
        }
    }

    bool contains(int key) {
        int index = key%1111;
        for(List* ptr=hashSet[index].next;ptr!=nullptr;ptr=ptr->next){
            if(ptr->val==key) return true;
        }
        return false;
    }
};

/**
 * Your MyHashSet object will be instantiated and called as such:
 * MyHashSet* obj = new MyHashSet();
 * obj->add(key);
 * obj->remove(key);
 * bool param_3 = obj->contains(key);
 */
```

4.设计哈希映射

- https://leetcode-cn.com/problems/design-hashmap/

```
struct List{
    int key;
    int value;
    List* next;
};
class MyHashMap {
public:
```

```
    List* hashMap = nullptr;
    MyHashMap() {
        hashMap = new List[1111];
        for(int i=0;i<1111;i++){
            hashMap[i].next = nullptr;
        }
    }

    void put(int key, int value) {
        // 尾插法
        int index = key%1111;
        List* pre = &hashMap[index];
        List* ptr = hashMap[index].next;
        while(ptr!=nullptr){
            if(ptr->key==key){
                ptr->value=value;
                return;
            }
            ptr = ptr->next;
            pre = pre->next;
        }
        List* temp = new List;
        temp->key = key;
        temp->value = value;
        temp->next = nullptr;
        pre->next = temp;
    }

    int get(int key) {
        int index = key%1111;
        for(List* ptr=hashMap[index].next;ptr!=nullptr;ptr=ptr->next){
            if(ptr->key==key) return ptr->value;
        }
        return -1;
    }

    void remove(int key) {
        int index = key%1111;
        List* pre = &hashMap[index];
        List* ptr = hashMap[index].next;
        while(ptr!=nullptr){
            if(ptr->key==key){
                pre->next = ptr->next;
                delete ptr;
                return;
            }
            ptr = ptr->next;
            pre = pre->next;
        }
```

```
    }
};

/**
 * Your MyHashMap object will be instantiated and called as such:
 * MyHashMap* obj = new MyHashMap();
 * obj->put(key,value);
 * int param_2 = obj->get(key);
 * obj->remove(key);
 */
```

三.哈希集合

本节主要讨论如何利用标准模板库使用哈希集合以及哈希集合的使用场景。

1.用法

哈希集是集合的实现之一，是一种存储不重复值的数据结构。

```
#include <unordered_set>                // 0. include the library

int main() {
    // 1. initialize a hash set
    unordered_set<int> hashset;
    // 2. insert a new key
    hashset.insert(3);
    hashset.insert(2);
    hashset.insert(1);
    // 3. delete a key
    hashset.erase(2);
    // 4. check if the key is in the hash set
    if (hashset.count(2) <= 0) {
        cout << "Key 2 is not in the hash set." << endl;
    }
    // 5. get the size of the hash set
    cout << "The size of hash set is: " << hashset.size() << endl;
    // 6. iterate the hash set
    for (auto it = hashset.begin(); it != hashset.end(); ++it) {
        cout << (*it) << " ";
    }
    cout << "are in the hash set." << endl;
    // 7. clear the hash set
    hashset.clear();
    // 8. check if the hash set is empty
    if (hashset.empty()) {
        cout << "hash set is empty now!" << endl;
    }
}
```

2.存在重复元素

- https://leetcode-cn.com/problems/contains-duplicate/
- 基本思路：哈希表经常用来解决重复元素的问题

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_set<int> hashset;
        for(int i=0;i<nums.size();i++){
            if(hashset.count(nums[i])>0) return true;
            hashset.insert(nums[i]);
        }
        return false;
    }
};
```

四.哈希映射

哈希映射是映射的一种实现，能够存储(key,value)键值对