# 题目

设计一个支持在平均 时间复杂度 O(1) 下，执行以下操作的数据结构：

insert(val)：当元素 val 不存在时返回 true ，并向集合中插入该项，否则返回 false 。

remove(val)：当元素 val 存在时返回 true ，并从集合中移除该项，否则返回 false 。

getRandom：随机返回现有集合中的一项。每个元素应该有 相同的概率 被返回。

示例 :

输入: inputs = ["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"]

[[], [1], [2], [2], [], [1], [2], []]

输出: [null, true, false, true, 2, true, false, 2]

解释:

RandomizedSet randomSet = new RandomizedSet(); // 初始化一个空的集合

randomSet.insert(1); // 向集合中插入 1 ， 返回 true 表示 1 被成功地插入

randomSet.remove(2); // 返回 false，表示集合中不存在 2

randomSet.insert(2); // 向集合中插入 2 返回 true ，集合现在包含 [1,2]

randomSet.getRandom(); // getRandom 应随机返回 1 或 2

randomSet.remove(1); // 从集合中移除 1 返回 true 。集合现在包含 [2]

randomSet.insert(2); // 2 已在集合中，所以返回 false

randomSet.getRandom(); // 由于 2 是集合中唯一的数字，getRandom 总是返回 2

提示：

-231 <= val <= 231 - 1

最多进行2 \* 105次insert ，remove和getRandom方法调用

当调用getRandom方法时，集合中至少有一个元素

注意：本题与主站 380 题相同：https://leetcode-cn.com/problems/insert-delete-getrandom-o1/

# 分析

## 方法一：变长数组 + 哈希表

思路：

这道题要求实现一个类，满足插入、删除和获取随机元素操作的平均时间复杂度为O(1)。

变长数组可以在O(1)的时间内完成获取随机元素操作，但是由于无法在O(1)的时间内判断元素是否存在，因此不能在O(1)的时间内完成插入和删除操作。哈希表可以在O(1)的时间内完成插入和删除操作，但是由于无法根据下标定位到特定元素，因此不能在O(1)的时间内完成获取随机元素操作。为了满足插入、删除和获取随机元素操作的时间复杂度都是O(1)，需要将变长数组和哈希表结合，变长数组中存储元素，哈希表中存储每个元素在变长数组中的下标。

插入操作时，首先判断val是否在哈希表中，如果已经存在则返回false，如果不存在则插入val，操作如下：

1、在变长数组的末尾添加val；

2、在添加val之前的变长数组长度为val所在下标index，将val和下标index存入哈希表；

3、返回true。

删除操作时，首先判断val是否在哈希表中，如果不存在则返回false，如果存在则删除val，操作如下：

1、从哈希表中获得val的下标index；

2、将变长数组的最后一个元素last移动到下标index处，在哈希表中将last的下标更新为index；

3、在变长数组中删除最后一个元素，在哈希表中删除val；

4、返回true。

删除操作的重点在于将变长数组的最后一个元素移动到待删除元素的下标处，然后删除变长数组的最后一个元素。该操作的时间复杂度是O(1)，且可以保证在删除操作之后变长数组中的所有元素的下标都连续，方便插入操作和获取随机元素操作。

获取随机元素操作时，由于变长数组中的所有元素的下标都连续，因此随机选取一个下标，返回变长数组中该下标处的元素。

代码：

class RandomizedSet {

private:

vector<int> nums;

unordered\_map<int, int> indices;

public:

/\*\* Initialize your data structure here. \*/

RandomizedSet() {

srand((unsigned)time(NULL));

}

/\*\* Inserts a value to the set. Returns true if the set did not already contain the specified element. \*/

bool insert(int val) {

if (indices.count(val)) {

return false;

}

int idx = nums.size();

nums.emplace\_back(val);

indices[val] = idx;

return true;

}

/\*\* Removes a value from the set. Returns true if the set contained the specified element. \*/

bool remove(int val) {

if (!indices.count(val)) {

return false;

}

int idx = indices[val];

int last = nums.back();

nums[idx] = last;

indices[last] = idx;

nums.pop\_back();

indices.erase(val);

return true;

}

/\*\* Get a random element from the set. \*/

int getRandom() {

int randomIdx = rand() % nums.size();

return nums[randomIdx];

}

};

/\*\*

\* Your RandomizedSet object will be instantiated and called as such:

\* RandomizedSet\* obj = new RandomizedSet();

\* bool param\_1 = obj->insert(val);

\* bool param\_2 = obj->remove(val);

\* int param\_3 = obj->getRandom();

\*/