Insert Delete GetRandom O(1)

Design a data structure that supports all following operations in **O(1)** time.

1. `insert(val)`: Inserts an item val to the set if not already present.
2. `remove(val)`: Removes an item val from the set if present.
3. `getRandom`: Returns a random element from current set of elements. Each element must have the **same probability** of being returned.

Example:

```
// Init an empty set.
RandomizedSet randomSet = new RandomizedSet();

// Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomSet.insert(1);

// Returns false as 2 does not exist in the set.
randomSet.remove(2);

// Inserts 2 to the set, returns true. Set now contains [1,2].
randomSet.insert(2);

// getRandom should return either 1 or 2 randomly.
randomSet.getRandom();

// Removes 1 from the set, returns true. Set now contains [2].
randomSet.remove(1);

// 2 was already in the set, so return false.
randomSet.insert(2);

// Since 1 is the only number in the set, getRandom always return 1.
randomSet.getRandom();
```

Solution 1

I got a similar question for my phone interview. The difference is that the duplicated number is allowed. So, think that is a follow-up of this question.
How do you modify your code to allow duplicated number?

```
public class RandomizedSet {
    ArrayList<Integer> nums;
    HashMap<Integer, Integer> locs;
    java.util.Random rand = new java.util.Random();
    /** Initialize your data structure here. */
    public RandomizedSet() {
        nums = new ArrayList<Integer>();
        locs = new HashMap<Integer, Integer>();
    }

    /** Inserts a value to the set. Returns true if the set did not already conta
in the specified element. */
    public boolean insert(int val) {
        boolean contain = locs.containsKey(val);
        if ( contain ) return false;
        locs.put( val, nums.size());
        nums.add(val);
        return true;
    }

    /** Removes a value from the set. Returns true if the set contained the speci
fied element. */
    public boolean remove(int val) {
        boolean contain = locs.containsKey(val);
        if ( ! contain ) return false;
        int loc = locs.get(val);
        if (loc < nums.size() - 1 ) { // not the last one than swap the last one
with this val
            int lastone = nums.get(nums.size() - 1 );
            nums.set( loc , lastone );
            locs.put(lastone, loc);
        }
        locs.remove(val);
        nums.remove(nums.size() - 1);
        return true;
    }

    /** Get a random element from the set. */
    public int getRandom() {
        return nums.get( rand.nextInt(nums.size()) );
    }
}
```

written by yubad2000 original link here

Solution 2

```
class RandomizedSet {
public:
    /** Initialize your data structure here. */
    RandomizedSet() {

    }

    /** Inserts a value to the set. Returns true if the set did not already conta
in the specified element. */
    bool insert(int val) {
        if (m.find(val) != m.end()) return false;
        nums.emplace_back(val);
        m[val] = nums.size() - 1;
        return true;
    }

    /** Removes a value from the set. Returns true if the set contained the speci
fied element. */
    bool remove(int val) {
        if (m.find(val) == m.end()) return false;
        int last = nums.back();
        m[last] = m[val];
        nums[m[val]] = last;
        nums.pop_back();
        m.erase(val);
        return true;
    }

    /** Get a random element from the set. */
    int getRandom() {
        return nums[rand() % nums.size()];
    }
private:
    vector<int> nums;
    unordered_map<int, int> m;
};
```

written by Ren.W original link here

Solution 3

```
public class RandomizedSet {
    HashMap<Integer, Integer> map;
    ArrayList<Integer> list;

    /** Initialize your data structure here. */
    public RandomizedSet() {
        map = new HashMap<Integer, Integer>();
        list = new ArrayList<Integer>();
    }

    /** Inserts a value to the set. Returns true if the set did not already conta
in the specified element. */
    public boolean insert(int val) {
        if(map.containsKey(val)) {
            return false;
        }else {
            map.put(val, list.size());
            list.add(val);
            return true;
        }
    }

    /** Removes a value from the set. Returns true if the set contained the speci
fied element. */
    public boolean remove(int val) {
        if(!map.containsKey(val)) {
            return false;
        }else {
            int key = map.get(val);
            int lastElement = list.get(list.size() - 1);
            map.put(lastElement, key);
            list.set(key, lastElement);
            map.remove(val);
            list.remove(list.size() - 1);
            return true;
        }
    }

    /** Get a random element from the set. */
    public int getRandom() {
        Random random = new Random();
        return list.get( random.nextInt(list.size()) );
    }
}

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * RandomizedSet obj = new RandomizedSet();
 * boolean param_1 = obj.insert(val);
 * boolean param_2 = obj.remove(val);
 * int param_3 = obj.getRandom();
 */
```

written by vivian34721 original link here

From Leetcoder.