380. Insert Delete GetRandom O(1)

Nov. 1, 2019 | 53.5K views

Average Rating: 4.88 (107 votes)

Design a data structure that supports all following operations in *average* **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 (it's guaranteed that at least one element exists when this method is called). 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 2 is the only number in the set, getRandom always return 2.
randomSet.getRandom();
```

Solution

Overview

We're asked to implement the structure which provides the following operations in *average* $\mathcal{O}(1)$ time:

- Insert
- Delete
- GetRandom

First of all - why this weird combination? The structure looks quite theoretical, but it's widely used in popular statistical algorithms like Markov chain Monte Carlo and Metropolis–Hastings algorithm. These algorithms are for sampling from a probability distribution when it's difficult to compute the distribution itself.

Let's figure out how to implement such a structure. Starting from the Insert, we immediately have two good candidates with $\mathcal{O}(1)$ average insert time:

- Hashmap (or Hashset, the implementation is very similar): Java HashMap / Python dictionary
- Array List: Java ArrayList / Python list

Let's consider them one by one.

> Hashmap provides Insert and Delete in average constant time, although has problems with GetRandom.

The idea of GetRandom is to choose a random index and then to retrieve an element with that index. There is no indexes in hashmap, and hence to get true random value, one has first to convert hashmap keys in a list, that would take linear time. The solution here is to build a list of keys aside and to use this list to compute GetRandom in constant time.

> Array List has indexes and could provide Insert and GetRandom in average constant time, though has problems with Delete.

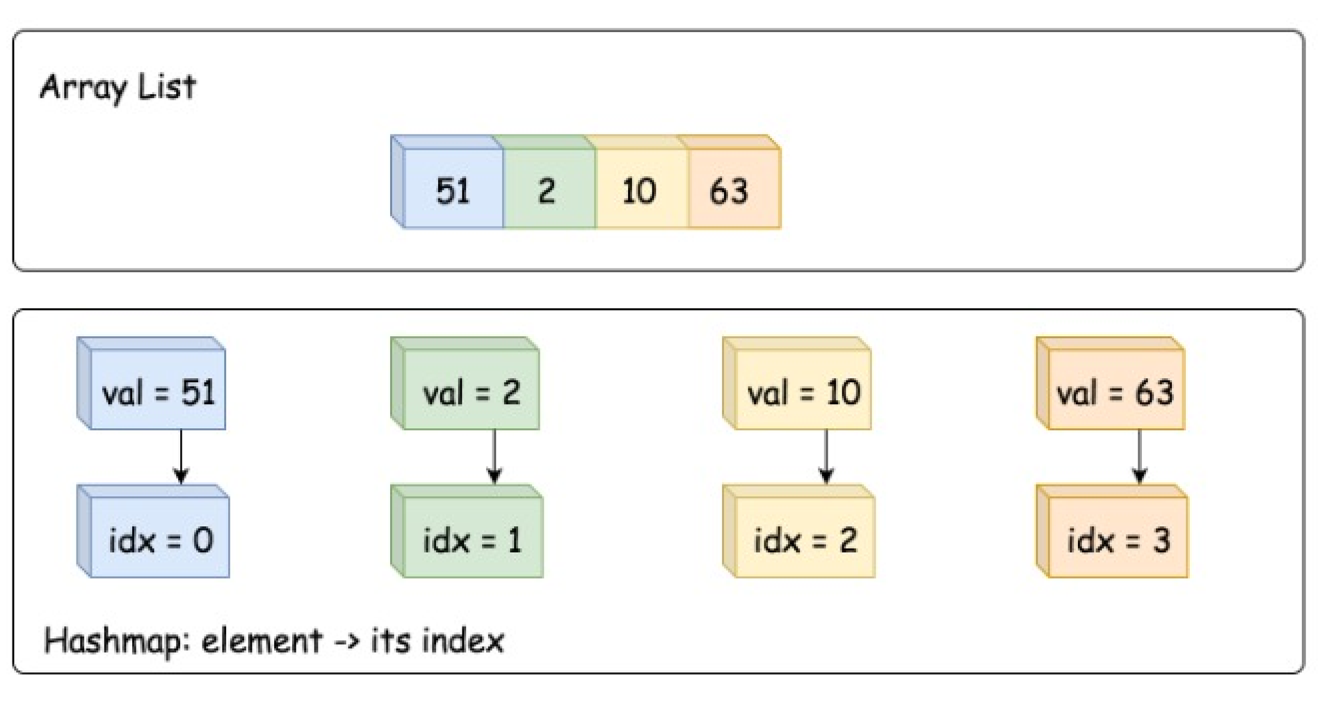
To delete a value at arbitrary index takes linear time. The solution here is to always delete the last value:

- Swap the element to delete with the last one.
- Pop the last element out.

For that, one has to compute an index of each element in constant time, and hence needs a hashmap which stores `element -> its index` dictionary.

Both ways converge into the same combination of data structures:

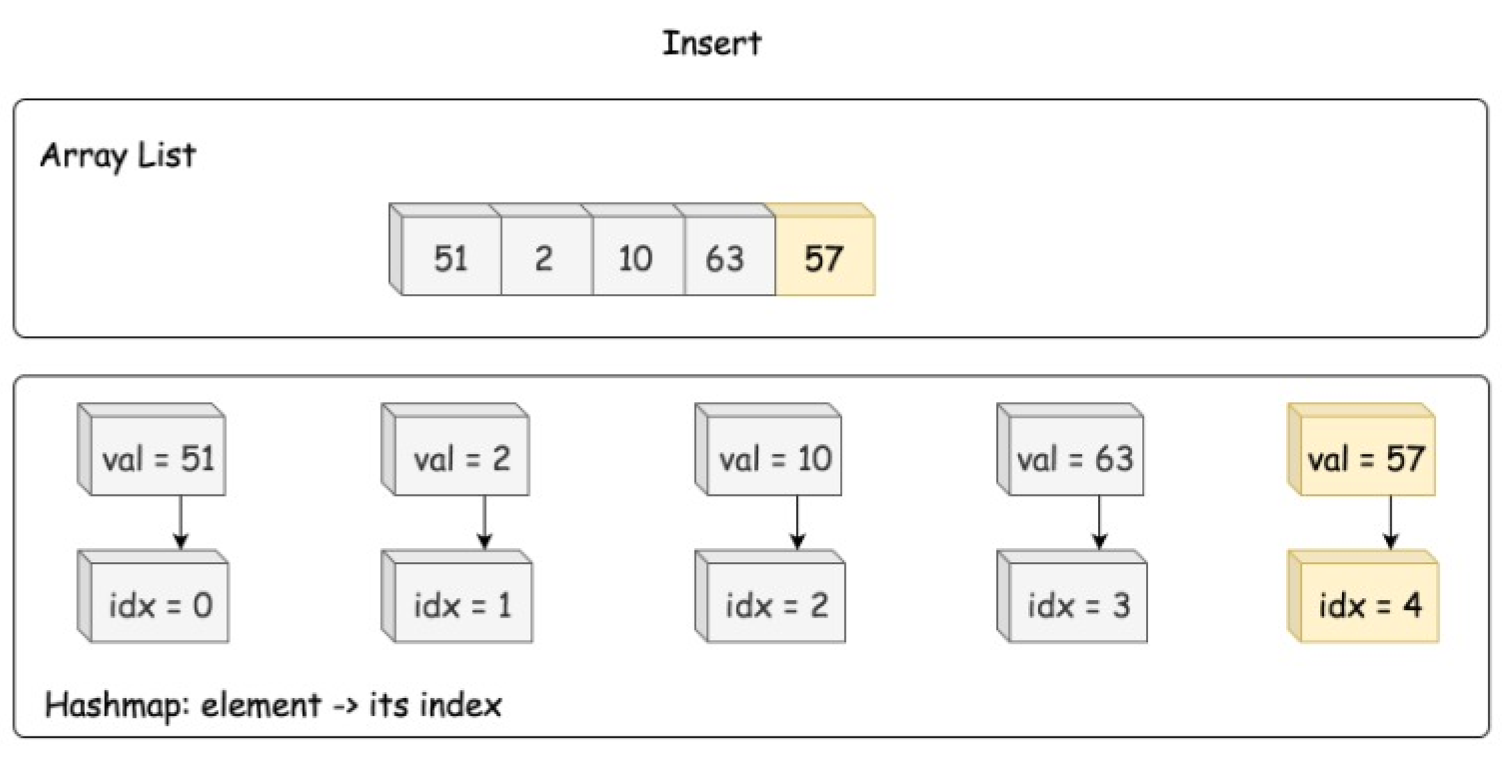
- Hashmap `element -> its index`.
- Array List of elements.



Approach 1: HashMap + ArrayList

Insert

- Add value -> its index into dictionary, average $\mathcal{O}(1)$ time.
- Append value to array list, average $\mathcal{O}(1)$ time as well.

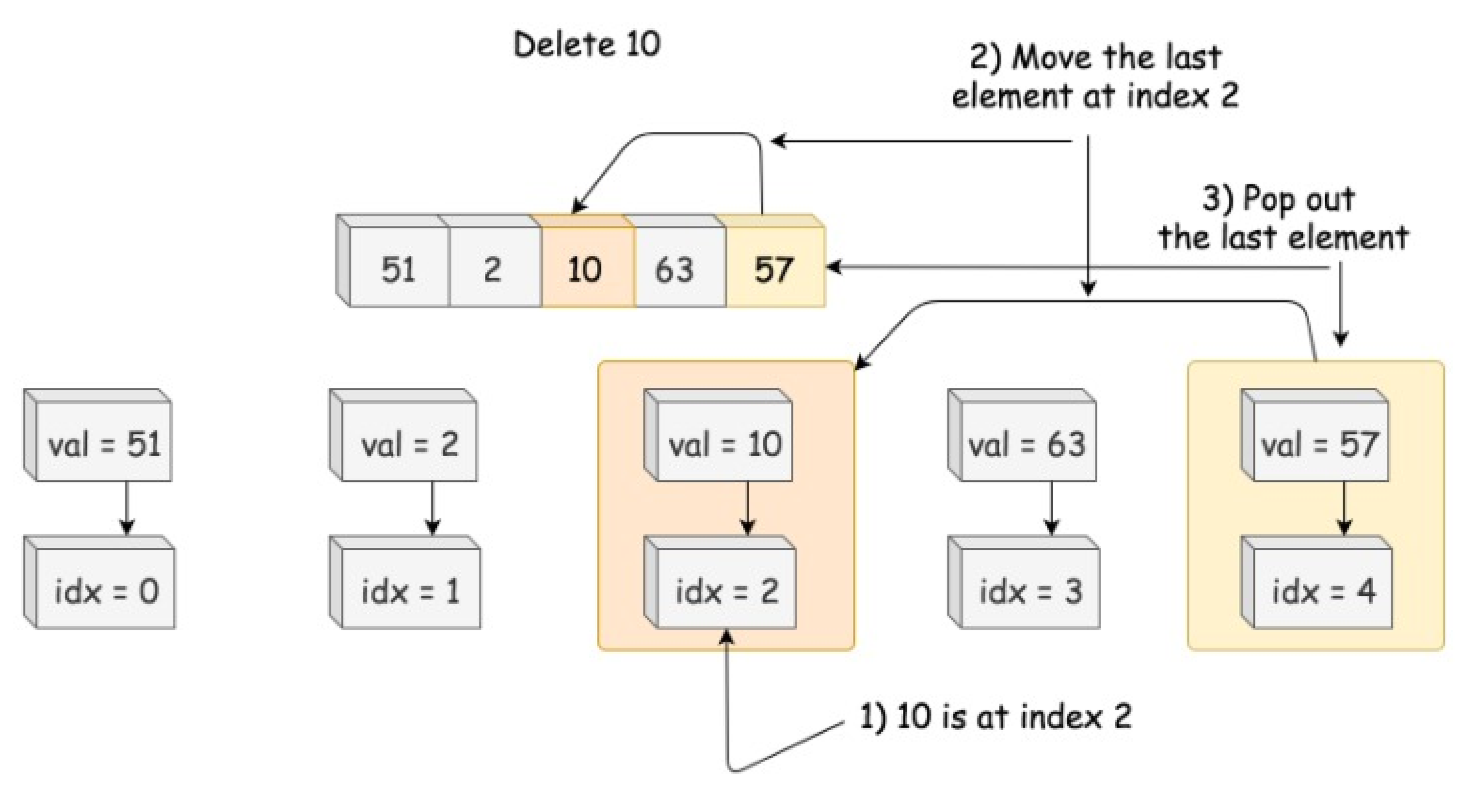


Java | Python | Copy

```python
def insert(self, val: int) -> bool:
    """
    Inserts a value to the set. Returns true if the set did not already contain the specified element.
    """
    if val in self.dict:
        return False
    self.dict[val] = len(self.list)
    self.list.append(val)
    return True
```

Delete

- Retrieve an index of element to delete from the hashmap.
- Move the last element to the place of the element to delete, $\mathcal{O}(1)$ time.
- Pop the last element out, $\mathcal{O}(1)$ time.



Java | Python | Copy

```python
def remove(self, val: int) -> bool:
    """
    Removes a value from the set. Returns true if the set contained the specified element.
    """
    if val in self.dict:
        # move the last element to the place idx of the element to delete
        last_element, idx = self.list[-1], self.dict[val]
        self.list[idx], self.dict[last_element] = last_element, idx
        # delete the last element
        self.list.pop()
        del self.dict[val]
        return True
    return False
```

GetRandom

GetRandom could be implemented in $\mathcal{O}(1)$ time with the help of standard `random.choice` in Python and `Random` object in Java.

Java | Python | Copy

```python
def getRandom(self) -> int:
    """
    Get a random element from the set.
    """
    return choice(self.list)
```

Implementation

Java | Python | Copy

```python
from random import choice
class RandomizedSet():
    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.dict = {}
        self.list = []


    def insert(self, val: int) -> bool:
        """
        Inserts a value to the set. Returns true if the set did not already contain the specified element.
        """
        if val in self.dict:
            return False
        self.dict[val] = len(self.list)
        self.list.append(val)
        return True


    def remove(self, val: int) -> bool:
        """
        Removes a value from the set. Returns true if the set contained the specified element.
        """
        if val in self.dict:
            # move the last element to the place idx of the element to delete
```

Complexity Analysis

- Time complexity. GetRandom is always $\mathcal{O}(1)$. Insert and Delete both have $\mathcal{O}(1)$ average time complexity, and $\mathcal{O}(N)$ in the worst-case scenario when the operation exceeds the capacity of currently allocated array/hashmap and invokes space reallocation.
- Space complexity: $\mathcal{O}(N)$, to store N elements.

Rate this article: ★★★★★

Previous Next

Comments: 26 Sort By

Type comment here... (Markdown is supported)

Preview Post

typedef82 ★58 November 4, 2019 10:43 AM
Great article
49 Share Reply

swapnilkamat23 ★8 December 22, 2019 11:34 AM
Best explanation ! Thank you
8 Share Reply

mission_2020 ★104 December 15, 2019 10:33 AM
We should add that the data wont be duplicate as the hashmap stores the data as the key
3 Share Reply
SHOW 1 REPLY

lingqingxu ★11 June 15, 2020 3:41 AM
厉害厉害
2 Share Reply

rayKansa ★3 April 6, 2020 12:09 AM
i get why remove("value") in java is O(n), why can't remove(index) use this trick and be O(1)
2 Share Reply
SHOW 4 REPLIES

poojank ★2 March 30, 2020 8:08 AM
Good explanation !Thanks
2 Share Reply

fangdream ★3 June 7, 2020 1:26 AM
choice and random are log(n) operations so how we are saying getRandom() is O(1)
2 Share Reply

parambole ★78 January 6, 2020 4:12 AM
How is list.get() an 0(1) operation? Since it is a LinkedList we will have to traverse it to get the value of the element
2 Share Reply
SHOW 3 REPLIES

Peter_Pen ★32 January 7, 2020 8:21 PM
Sorry, but the solution works only if we put in the structure no more than Integer.MAX_VALUE elements (java)
Random (Java) works only with 0-Integer.MAX_VALUE either.
Thus - this is not the right solution at all. (Unless you add conditions to the task)
0 Share Reply
SHOW 1 REPLY

sohammehta ★1156 July 13, 2020 3:16 AM
You might wanna use only `list.add(val);` [O(1)] instead of `list.add(list.size(), val);` [O(n)] in #insert method
0 Share Reply

‹ 1 2 3 ›