☰ Articles > 380. Insert D

❮ Previous (/articles/remove-duplicates-from-sorted-array-ii/) Next ❯ (/articles/design-circular-queue/)

380. Insert Delete GetRandom O(1) (/problems/insert-delete-getrandom-o1/)

Nov. 1, 2019 | 48.9K views

Average Rating: 4.89 (102 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. 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 (https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo) and Metropolis–Hastings algorithm (https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_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 (https://wiki.python.org/moin/TimeComplexity):

- Hashmap (or Hashset, the implementation is very similar): Java HashMap (https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) / Python dictionary (https://docs.python.org/3/tutorial/datastructures.html#dictionaries)

- Array List: Java ArrayList (https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) / Python list (https://docs.python.org/3/tutorial/datastructures.html)

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.

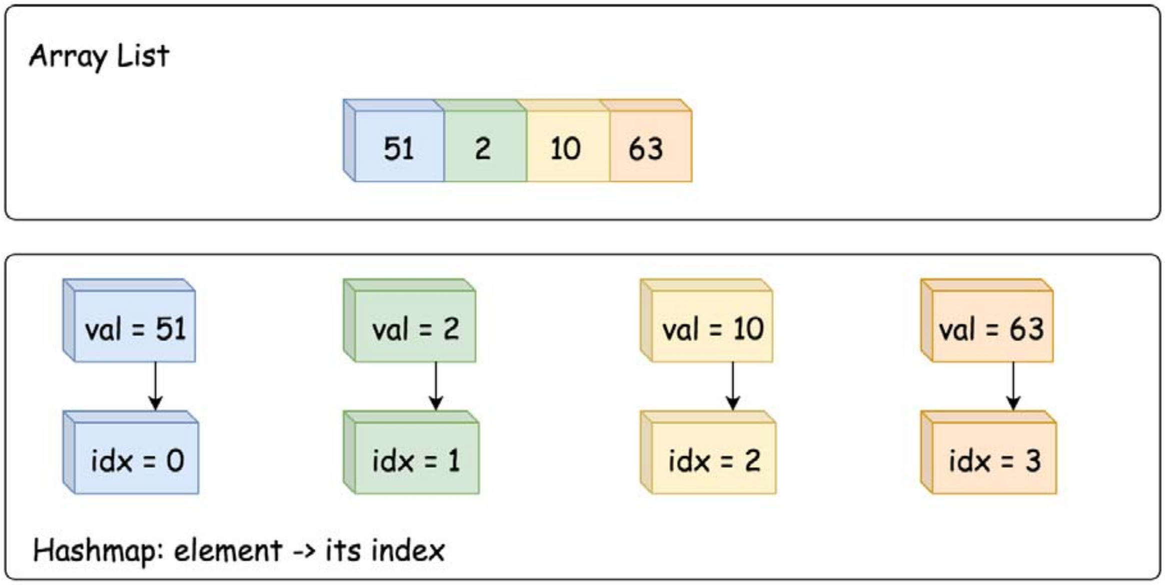
- Pop the last element out.

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`.

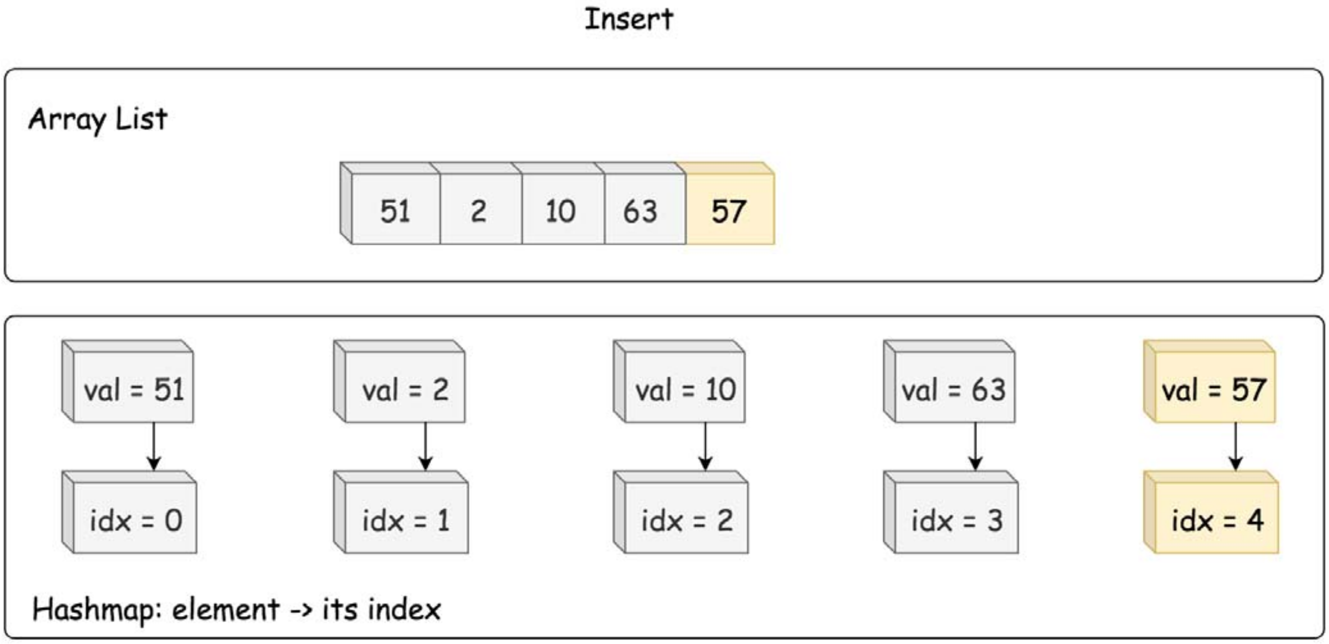
- Array List of elements.



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.



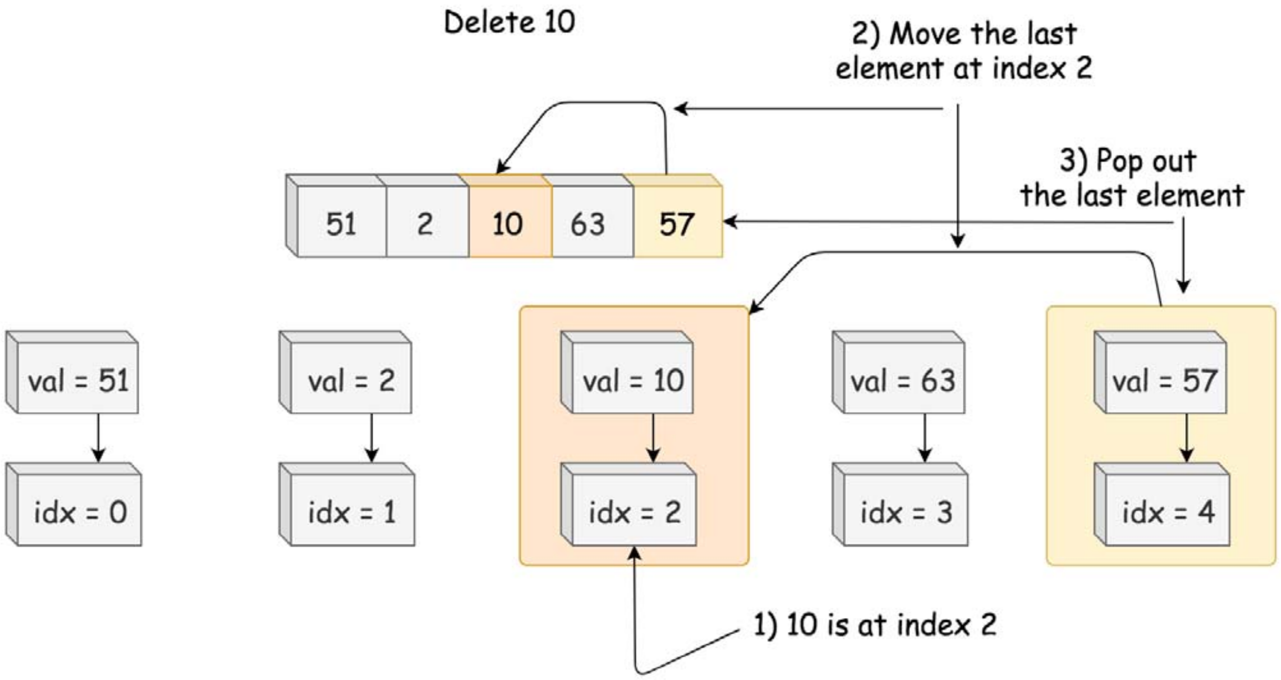
Java Python Copy

```
/** Inserts a value to the set. Returns true if the set did not already contain the specified element. */
public boolean insert(int val) {
  if (dict.containsKey(val)) return false;

  dict.put(val, list.size());
  list.add(list.size(), 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

```
/** Removes a value from the set. Returns true if the set contained the specified element. */
public boolean remove(int val) {
  if (! dict.containsKey(val)) return false;

  // move the last element to the place idx of the element to delete
  int lastElement = list.get(list.size() - 1);
  int idx = dict.get(val);
  list.set(idx, lastElement);
  dict.put(lastElement, idx);
  // delete the last element
  list.remove(list.size() - 1);
  dict.remove(val);
  return true;
}
```

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

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

Implementation

Java Python Copy

```
class RandomizedSet {
  Map<Integer, Integer> dict;
  List<Integer> list;
  Random rand = new Random();

  /** Initialize your data structure here. */
  public RandomizedSet() {
    dict = new HashMap();
    list = new ArrayList();
  }

  /** Inserts a value to the set. Returns true if the set did not already contain the specifie
element. */
  public boolean insert(int val) {
    if (dict.containsKey(val)) return false;

    dict.put(val, list.size());
    list.add(list.size(), val);
    return true;
  }

  /** Removes a value from the set. Returns true if the set contained the specified element. *
  public boolean remove(int val) {
    if (! dict.containsKey(val)) return false;

    // move the last element to the place idx of the element to delete
    int lastElement = list.get(list.size() - 1);
    int idx = dict.get(val);
    list.set(idx, lastElement);
    dict.put(lastElement, idx);
    // delete the last element
    list.remove(list.size() - 1);
    dict.remove(val);
    return true;
  }

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

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:

☰ Articles › 380. Insert D

❮ Previous (/articles/remove-duplicates-from-sorted-array-ii/)

Next ❯ (/articles/design-circular-queue/)

Comments: 25

Sort By ▾

Type comment here... (Markdown is supported)

Preview Post

(/typedef82)

typedef82 (/typedef82) ★ 55 November 4, 2019 12:13 AM

Great article

46 ^ ˅ Share Reply

(/swapnilkamat23)

swapnilkamat23 (/swapnilkamat23) ★ 7 December 22, 2019 1:04 AM

Best explanation ! Thank you

6 ^ ˅ Share Reply

(/mission_2020)

mission_2020 (/mission_2020) ★ 87 December 15, 2019 12:03 AM

We should add that the data wont be duplicate as the hashmap stores the data as the key

2 ^ ˅ Share Reply

SHOW 1 REPLY

(/coder0710)

coder0710 (/coder0710) ★ 2 June 6, 2020 3:56 PM

choice and random are log(n) operations so how we are saying getRandom() is O(1)

1 ^ ˅ Share Reply

(/raykansa)

rayKansa (/raykansa) ★ 2 April 5, 2020 2:39 PM

i get why remove("value") in java is O(n), why can't remove(index) use this trick and be O(1)

1 ^ ˅ Share Reply

SHOW 3 REPLIES

(/poojank)

poojank (/poojank) ★ 1 March 29, 2020 10:38 PM

Good explanation !Thanks

1 ^ ˅ Share Reply

(/parambole)

parambole (/parambole) ★ 66 ◷ January 5, 2020 5:42 PM

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

1 ▲ ▼ Share Reply

SHOW 3 REPLIES

(/peter_pen)

Peter_Pen (/peter_pen) ★ 25 ◷ January 7, 2020 9:51 AM

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

(/133c7)

133c7 (/133c7) ★ 10 ◷ 2 days ago

How are our solutions supposed to conform to the autograder's expected output? The very nature of the solution is random and cannot be predicted deterministically.

0 ▲ ▼ Share Reply

SHOW 1 REPLY

(/lingqingxu)

lingqingxu (/lingqingxu) ★ 9 ◷ June 14, 2020 6:11 PM

厉害厉害

0 ▲ ▼ Share Reply

‹ 1 2 3 ›