Design a data structure that supports insert, delete, search and getRandom in constant time

Design a data structure that supports following operations in Θ(1) time.

insert(x): Inserts an item x to the data structure if not already present.

remove(x): Removes an item x from the data structure if present.

search(x): Searches an item x in the data structure.

getRandom(): Returns a random element from current set of elements

We strongly recommend to minimize your browser and try this yourself first.

We can use hashing to support first 3 operations in Θ(1) time. How to do the 4th operation? The idea is to use a resizable array (ArrayList in Java, vector in C) together with hashing. Resizable arrays support insert in Θ(1) amortized time complexity. To implement getRandom(), we can simply pick a random number from 0 to size-1 (size is number of current elements) and return the element at that index. The hash map stores array values as keys and array indexes as values.

Following are detailed operations.

insert(x)
1) Check if x is already present by doing a hash map lookup.
2) If not present, then insert it at the end of the array.
3) Add in hash table also, x is added as key and last array index as index.

remove(x)
1) Check if x is present by doing a hash map lookup.
2) If present, then find its index and remove it from hash map.
3) Swap the last element with this element in array and remove the last element.
Swapping is done because the last element can be removed in O(1) time.
4) Update index of last element in hash map.

getRandom()
1) Generate a random number from 0 to last index.
2) Return the array element at the randomly generated index.

search(x)
Do a lookup for x in hash map.

Below is Java implementation of the data structure.

```
/* Java program to design a data structure that support folloiwng operations
   in Theta(n) time
   a) Insert
   b) Delete
   c) Search
   d) getRandom */
import java.util.*;

// class to represent the required data structure
class MyDS
{
   ArrayList<Integer> arr;   // A resizable array

   // A hash where keys are array elements and vlaues are
   // indexes in arr[]
   HashMap<Integer, Integer>  hash;

   // Constructor (creates arr[] and hash)
   public MyDS()
   {
       arr = new ArrayList<Integer>();
       hash = new HashMap<Integer, Integer>();
```

```
    }

    // A Theta(1) function to add an element to MyDS
    // data structure
    void add(int x)
    {
       // If ekement is already present, then noting to do
       if (hash.get(x) != null)
           return;

       // Else put element at the end of arr[]
       int s = arr.size();
       arr.add(x);

       // And put in hash also
       hash.put(x, s);
    }

    // A Theta(1) function to remove an element from MyDS
    // data structure
    void remove(int x)
    {
        // Check if element is present
        Integer index = hash.get(x);
        if (index == null)
           return;

        // If present, then remove element from hash
        hash.remove(x);

        // Swap element with last element so that remove from
        // arr[] can be done in O(1) time
        int size = arr.size();
        Integer last = arr.get(size-1);
        Collections.swap(arr, index,  size-1);

        // Remove last element (This is O(1))
        arr.remove(size-1);

        // Update hash table for new index of last element
        hash.put(last, index);
     }

     // Returns a random element from MyDS
     int getRandom()
     {
        // Find a random index from 0 to size - 1
        Random rand = new Random();  // Choose a different seed
        int index = rand.nextInt(arr.size());

        // Return element at randomly picked index
        return arr.get(index);
     }

     // Returns index of element if element is present, otherwise null
     Integer search(int x)
     {
        return hash.get(x);
     }
}

// Driver class
class Main
{
     public static void main (String[] args)
     {
         MyDS ds = new MyDS();
         ds.add(10);
         ds.add(20);
         ds.add(30);
         ds.add(40);
         System.out.println(ds.search(30));
         ds.remove(20);
        ds.add(50);
        System.out.println(ds.search(50));
        System.out.println(ds.getRandom());
    }
}
```

Output:

```
2
3
40
```