Hashing 1: Introduction

HashMap

Let's take an example:-

- Imagine we have a hotel called Reddison, which has 5 rooms.
- How can we maintain information on the status of rooms provided the hotel is old and hasn't adapted to technology yet?

Solution: The hotel may maintain a manual register for five rooms like:-

Room no	occupied
1	Yes
2	No
3	Yes
4	No
5	No

- After a few years, the hotel became a success, and the rooms increased to 1000.
- Provided the hotel decided to adapt to technology, what is the programmatically most straightforward approach to maintain the status of rooms?
    - An array can be maintained where the index can denote the room number.
    - If there are N rooms, we'll create an array of size + 1 where true denotes that room is occupied, and false denotes unoccupied.
- Pandemic hit, due to which footfall reduced significantly. Owner visits Numerologist who asks them to change room numbers to some random lucky numbers from [1-10^9]. How can we maintain the status of the rooms now?
    - Maintain boolean array of length $10^9 + 1$ `bool arr[10^9 + 1]` .
    - **ISSUE:** Status can be checked in O(1), but just for 1000 rooms, we require an array of size 10^9.

- **Solution:** HashMaps
  - HashMap is a data structure that stores <key, value> pair.

Key	value
100003	occupied
3	occupied
10007	occupied

- **In HashMap, T.C of search is O(1) time and S.C is O(N)**
- Key must be unique
- Value can be anything
- **Note:** We'll discuss the internal working of Map in Advanced classes.

In hashmap approach we can search in O(1) time and can have a space complexity of O(N)

Let's see some questions based on Hashmap.

Question

Which of the following HashMap will you use to store the population of every country?

Choices

☐ HashMap<String, Float>
☐ HashMap<String, Double>
☐ HashMap<String, String>
☑ HashMap<String, Long>

- Key must be unique in Hashmap, so for that reason :
  - We use the country name as the key.
  - Since the country name is a `string` , the key would be of type `string` .
- In this case, value is a population that can be stored in `int` or `long` datatype.
- Solution:-

  `hashmap<String,long> populationByCountry` .

Question

Which of the following HashMap will you use to store the no of states of every country?

Choices

☐ HashMap<String, Float>
☐ HashMap<String, Double>
☑ HashMap<String, int>
☐ HashMap<String, String>

- Key must be unique in Hashmap, so for that reason :
  - We use the country name as the key.
  - Since the country name is a `string` , the key would be of type `string` .
- We know that value can be anything. In this case :
  - Value is the number of states stored in `int` or `long` datatype.
- Solution:-

```
hashmap<String,int> numberOfStatesByCountry
```

Question

Which of the following HashMap will you use to store the name of all states of every country?

Choices

☐ HashMap<String, List < Float > >
☑ HashMap<String, List < String > >
☐ HashMap<String, String>
☐ HashMap<String, Long>

- Key must be unique in Hashmap, so for that reason :
  - We use the country name as the key.
  - Since the country name is a `string` , the key would be of type `string` .
- Value can be anything. In this case:
  - Value is the name of states.
  - To store them, we would require a list of strings, i.e., `vector<string>` in C++ or `ArrayList<Sting>` in Java, etc., to store the name of states.
- Solution:-

```
hashmap<String,list<String>> nameOfStatesByCountry
```

Question

Which of the following HashMap will you use to store the population of each state in every country?

Choices

□ HashMap<String, Int>
□ HashMap<String, List < Int > >
☑ HashMap<String,HM < String , Int > >
□ HashMap<String, Long>

- Key must be unique in Hashmap, so for that reason :
  - We use the country name as the key.
  - Since the country name is a `string` , the key would be of type `string` .
- Value can be anything. In this case :
  - We need to store the name of states with its population.
  - We will create another hashmap with the state name as key and population as value.
- Solution:-

```
hashmap<String,hashmap<String,Long>> populationOfStatesByCountry
```

We can observe that:-

- **Value can be anything.**
- **Key can only be a primitive datatype.**

HashSet

Sometimes we only want to store keys and do not want to associate any values with them, in such a case; we use HashSet.

```
Hashset<Key Type>
```

- **Key must be unique**
- **Like HashMap, we can search in O(1) time in Set**

HashMap and HashSet Functionalities

HashMap

- **INSERT(Key,Value):** new key-value pair is inserted. If the key already exists, it does no change.
- **SIZE:** returns the number of keys.
- **DELETE(Key):** delete the key-value pair for given key.
- **UPDATE(Key,Value):** previous value associated with the key is **overridden** by the new value.
- **SEARCH(Key):** searches for the specified key.

HashSet

- **INSERT(Key):** inserts a new key. If key already exists, it does no change.
- **SIZE:** returns number of keys.
- **DELETE(Key):** deletes the given key.
- **SEARCH(Key):** searches for the specified key.

Time Complexity of **all the operations** in both Hashmap and Hashset is **O(1)**.

Therefore, if we insert N key-value pairs in HashMap, then time complexity would be O(N) and space complexity would be O(N).

Hashing Library Names in Different Languages

	Java	**C++**	**Python**	**Js**	**C#**
Hashmap	Hashmap	unordered_map	dictionary	map	dictionary
Hashset	Hashset	unordered_set	set	set	Hashset

Problem 1 Frequency of given elements

Problem Statement

Given N elements and Q queries, find the frequency of the elements provided in a query.

Example

N = 10

2	**6**	**3**	**8**	**2**	**8**	**2**	**3**	**8**	**10**	**6**

Q = 4

2	**8**	**3**	**5**

Solution

Element	**Frequency**
2	3
8	3
3	2
5	0

Idea 1

- For each query, find the frequency of the element in the Array.
- TC - **O(Q*N)** and SC - **O(1)**.

> How can we improve TC?

:::warning
Please take some time to think about the solution approach on your own before reading further.....
:::

Approach

- First, search for the element in the Hashmap.
    - If the element does not exist, then insert the element as key and value as 1.
    - If an element already exists, then increase its value by one.

Pseudeocode

```
Function frequencyQuery(Q[], A[])
{
   Hashmap<int,int> mp;
   q = Q.length
   n = A.length

   for(i = 0 ; i < n ; i ++ )
   {
      if(mp.Search(A[i]) == true)
      {
         mp[Array[i]] ++
      }
      else{
        mp.Insert(A[i],1)
      }
   }

   for(i = 0 ; i < q; i ++ )
   {
        if(mp.Search(Q[i]) == true)
      {
         print(mp[Q[i]])
      }
      else{
        print("0")
      }
   }
}
```

Complexity

Time Complexity: O(N)

Space Complexity: O(N)

Problem 2 First non repeating element

Problem Statement

Given N elements, find the first non-repeating element.

Example

Input 1:

```
N = 6
```

1	**2**	**3**	**1**	**2**	**5**

Output1 :

```
ans = 3
```

1	**2**	**3**	**1**	**2**	**5**

Input 2:

```
N = 8
```

4	**3**	**3**	**2**	**5**	**6**	**4**	**5**

Output 2:

```
ans = 2
```

Input 3:

```
N = 7
```

2	**6**	**8**	**4**	**7**	**2**	**9**

Output 3:

```
ans = 6
```

:::warning
Please take some time to think about the solution approach on your own before reading further.....
:::

Solution

Idea 1

- Use Hashmap to store the frequency of each element. Store <**key**:element, **value**:frequency>.
- Iterate over the Hashmap and find the element with frequency 1.

Flaw in Idea 1

- When we store in Hashmap, the order of elements is lost; therefore, we cannot decide if the element with frequency 1 is first non-repeating in the order described in the Array.

Idea 2

- Use Hashmap to store the frequency of each element. Store `<key:element, value:frequency>`.
- Instead of Hashmap, iterate over the Array from the start. If some element has a frequency equal to one, then return that element as answer.

Pseudeocode

```
Function firstNonRepeating(A[]) {
    Hashmap < int, int > mp;
    n = A.length

    for (i = 0; i < n; i++) {
        if (mp.Search(A[i]) == true) {
            mp[A[i]]++
        } else {
            mp.Insert(A[i], 1)
        }
    }
    for (i = 0; i < n; i++) {
        if (mp[A[i]] == 1) {
            return A[i]
        }
    }
    return -1
}
```

Time Complexity : **O(N)**

Space Complexity : **O(N)**

Problem 3 Count of Distinct Elements

Problem Statement

Given an array of N elements, find the count of distinct elements.

Example

Input:

N = 5

3	**5**	**6**	**5**	**4**

Output:

```
ans = 4
```

Explanation: We have to return different elements present. If some element repeats, we will count it only once.

Input:
N = 3

3	**3**	**3**

Output:

```
ans = 1
```

Input:

N = 5

1	**1**	**1**	**2**	**2**

Output:

```
ans = 2
```

Solution

- Insert element in Hashset and return the size of HashSet.

> In Hashset, if a single key is inserted multiple times, still, its occurrence remains one.

Pseudocode

```
Function distinctCount(Array[]) {
    hashset < int > set;
    for (i = 0; i < Array.length; i++) {
        set.insert(Array[i])
    }
    return set.size
}
```

Complexity

Time Complexity: O(N)

Space Complexity: O(N)

Problem 4 Subarray sum 0

Problem Statement

Given an array of N elements, check if there exists a subarray with a sum equal to 0.

Example

Input:

N = 10

2	**2**	**1**	**-3**	**4**	**3**	**1**	**-2**	**-3**	**2**

Output:

if we add elements from index 1 to 3, we get 0; therefore, the answer is **true**.

:::warning

Please take some time to think about the solution approach on your own before reading further.....

:::

Solution

- Traverse for each subarray check if sum == 0.
  - Brute Force: Create all Subarrays, Time complexity: **O(n^3)**.
  - We can optimize further by using **Prefix Sum** or **Carry Forward** method and can do it in Time Complexity: **O(n^2)**.
  - How can we further optimize it?

Observations

- Since we have to find sum of a subarrays(range), we shall think towards **Prefix Sum**.

Initial Array: -

0	1	2	3	4	5	6	7	8	9
2	2	1	-3	4	3	1	-2	-3	2

Prefix sum array: -

0	1	2	3	4	5	6	7	8	9
2	4	5	2	6	9	10	8	5	7

We need a subarray with **sum(i to j) = 0**
Using Prefix Sum Array,
PrefixSum[j] - PrefixSum[i-1] = 0
PrefixSum[j] = PrefixSum[i-1]

It implies, if there exist duplicate values in Prefix Sum Array, then the sum of a subarray is 0.

Example,

```
PrefixSum[2] = 5
PrefixSum[8] = 5
sum of elements in intial array from index 3 to 8 = 0
```

Summary

- If numbers are repeating in Prefix Sum Array, then there exists a subarray with sum 0.
- Also, if the Prefix Sum Array element is 0, then there exists a subarray with sum 0.
    - Example:
        - A[] = {2, -1, 3, 5}
        - PrefixSum[] = {2, -1, 0, 5}
        - Here, 0 in PrefixSum Array implies that there exist a subarray with sum 0 starting at index 0.

Approach

- Calculate prefix sum array.
- Traverse over elements of prefix sum array.
    - If the element is equal to 0, return true.
    - Else, insert it to HashSet.
- If the size of the prefix array is not equal to the size of the hash set, return true.
- Else return false.

Pseudeocode

```
// 1. todo calculate prefix sum array

// 2.
Function checkSubArraySumZero(PrefixSumArray[]) {
    Hashset < int > s
    for (i = 0; i < PrefixSumArray.length; i++) {
        if (PrefixSumArray[i] == 0) {
            return true
        }
        s.insert(PrefixSumArray[i])
    }
    if (s.size != PrefixSumArray.size)
        return true
    return false
}
```

Time Complexity : **O(N)**

Space Complexity : **O(N)**

HINT for Count Subarrays having sum 0

Given an array A of N integers.

Find the count of the subarrays in the array which sums to zero. Since the answer can be very large, return the remainder on dividing the result with 109+7

Input 1

A = [1, -1, -2, 2]

Output 1

3

Explanation

The subarrays with zero sum are [1, -1], [-2, 2] and [1, -1, -2, 2].

Input 2

A = [-1, 2, -1]

Output 2

1

Explanation

The subarray with zero sum is [-1, 2, -1].