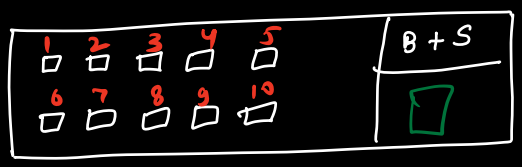
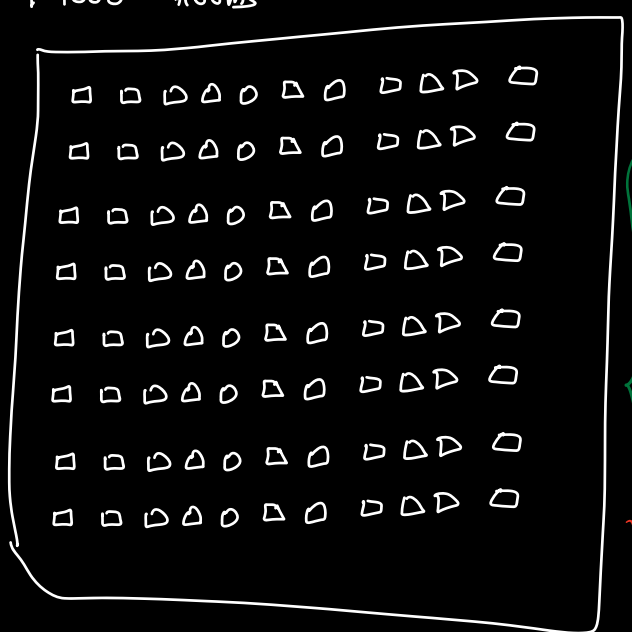
Shreyansh
&
Bhupinder



A register to know availability of rooms.

1-1000 Rooms



bool ch[1000]

52 occupy ch[52] = T

73 occupy ch[73] = T

52 or ×

52 left ch[52] = F

TC: O(1)

Jyotish 1000 lucky number $1 - 10^9$

{ 2, 7, 93, 10^5, $10^5 + 8$, $10^8 + 465$, ... }

ch[10^9]

ch[x] = T

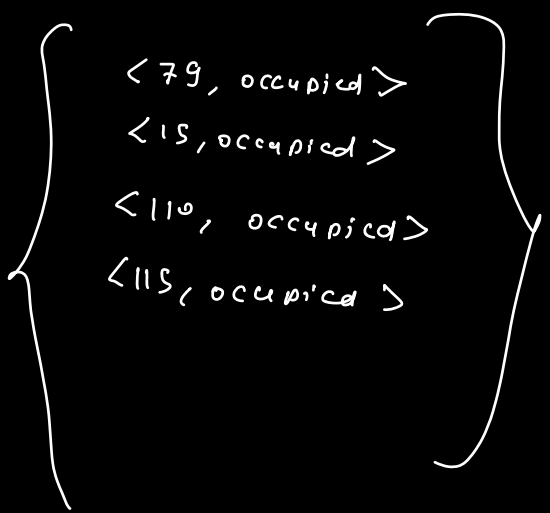
ch[x] = F

Creating an array 10^9 size

Actually using 10^3 size.

→ Huge space wastage

Hashmap.



TC: To insert
To search O(1)
To change

Hashmap Key, value → have to unique

Hashmap < key, value > hm

Hashmap < int, bool > hm

Q. Store population of every country.

Key: Country_name {unique}

Value: Population

Hashmap < Key, value >

hashmap < String, long >

Q. No of states in each country.

Key : Country_name (unique)

No. of states : Value

Hashmap < String, int >

Q. For every country, store all the states names.

Key : Country_name

Value: Names of all the states.

hashmap < String, arraylist/array of String >

Q. For every country, store population of each state.

Key : Country_name

Value: Population of each state

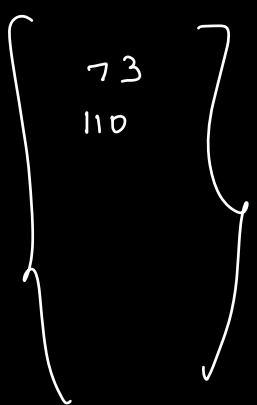
Hashmap < String, hashmap < String, long > >

Value : It can be anything

Key : { All primitive datatypy + string }

bool, char, int, long, float, double

string



Only Keys

Hashset → unique

Hashset<key> hs

To search : O(1)

100 students

Hashset

{ scalerid1, scalerid2, scalerid3 }

	Java	C++	Python	JS	C#
Hashmap	hashmap	unordered_map	dict	map	2
hashset	hashset	unordered_set	set	set	:

hashmap <key, value> name

- insert <key, value>
- search (key)
- remove <key, value>
- update <key, new_value>
- size()

→ TC: O(1) time

insert (key, v1)
insert (key, v2)

hashset

- insert (key)
- remove (key)
- search (key)
- update (key) → remove (key), insert (new key)
- size

→ O(1) time

Q Find frequency in Array

Given N array, Q queries.

↓

one element ⇒ frequency of elem in the array.

N = 10
Q = 4

arr = { 2, 6, 3, 8, 2, 8, 8, 8, 5, 10 }

2 : 2
8 : 4
3 : 1
12 : 0

Idea 1: For every query, iterate & get the count.

TC : O(NQ)
SC : O(1)

idea 2: Hashmap < key, value >

key → array element
value → frequency of element

hashmap < int, int >

N = 10 arr = { 2, 6, 3, 8, 2, 8, 8, 8, 5, 10 }

Hm

{ <~~3,1~~> <2,2>
<6,1>
<3,1>
<8,~~1~~ ~~2~~ ~~3~~ 4>
<5,1>
<10,1> }

2 : 2
8 : 4
3 : 1
12 :

```
Hashmap <int, int> hm = new Hashmap<>();

// Build the hm

for( i=0 ; i<N ; i++ )          // N: array size
{
    If ( arr[i] is present in hm )      // hm.containsKey(arr[i])
    {
        update freq by +1               // key = arr[i]; hm[key] -> value; hm.get(key) -> value
    }
    else
    {
        insert new key, value {arr[i], 1}      // hm.put(arr[i], 1)
    }
}

for( i=1 ; i<=Q ; i++ )
{
    int x <- input

    If ( x is present in hm )           // hm.containsKey(x)
    {
        print( hm.get(x) )
    }
    else
    {
        print(0)
    }
}
```

Breaks

10:27 10:35

TC : O(N+Q)

SC : O(N)

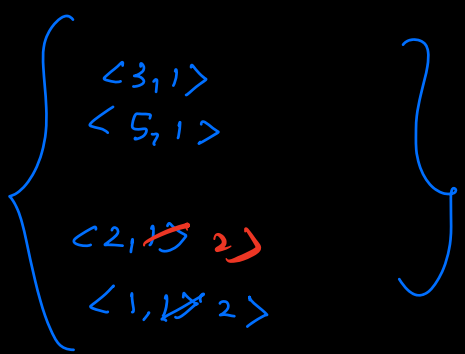
Q. Find the first non-repeating elements. (Element from start)

Ex arr[6] = { 1, 2, 3, 1, 2, 5 }

ans = 3.

1. Create hm with <element, frequency>
2. Iterate array until freq == 1 comes.

Ex arr[6] = { 1, 2, 3, 1, 2, 5 }



```
Hashmap <int,int> hm

for( i=0; i<N; i++)
{
    if ( hm.containskey ( arr[i]) )
    {
        hm.put ( arr[i], hm.get( arr[i]) + 1 )
    }
    else
    {
        hm.put ( arr[i], 1)
    }
}

for (i=0; i<N; i++)
{
    if ( hm.get ( arr[i]) == 1)
    {
        return arr[i];
    }
}
```

Q. Given an array. Find no of distinct numbers.

arr[5] = { 3, 5, 6, 5, 4 }

3, 5, 6, 4

ans = 4

arr[7] = { 6, 3, 7, 3, 8, 6, 9 }

6 3 7 8 9

ans = 5

arr[5] = { 1, 1, 1, 1, 2 }

1, 2

ans = 2

Create hashmap <value, freq>

hm.size()

```
Hashmap <int,int> hm

for(i=0; i<N; i++)
{
    if( hm.containskey( arr[i] ) )
    {
        hm.put ( arr[i], hm.get( arr[i] ) + 1 )
    }
    else
    {
        hm.put ( arr[i], 1)
    }

return hm.size()
```

2. Create hashset, ans: sz of hashset.

```
Hashset <int> hs.

for( i=0; i<N; i++)
{
    hs.insert ( arr[i] )
}

return hs.size()
```

TC : O(N)
SC : O(N)

Q. Given an array. Check if all elements are non-repeating or not. → true / false

If hs.size / hm.size = distinct elements.

hs.size() == N.

```
Hashset<int> hs.
for(i=0; i<N; i++)
{
    hs.insert(arr[i])
}
return hs.size() == N   → array size.
```

```
Hashset <int> hs.

for( i=0 ; i<N; i++)
{
    if ( hs.contains ( arr[i] )    return false
    hs. insert ( arr[i])
}
return True.
```

Syntax

insert	hm.put (Key, Value)
Search	hm. containsKey (Key)
Access	hm.get (Key)
Update	Put again hm.put (Key, newvalue)
Size	hm.size()

n <String, Object>