Hashing 1 : Introduction

Radisson Hotel   5 rooms only

Register

1	Yes
2	No
3	Yes
4	No
5	No

no technology

5 room → 1000 rooms   [1, 1000]

bool room[1001]   ⇒ room[i] = true
if i^{th} room is occupied

⇒ room[i] = false   else

After Pandemic,

1000 room → $[1 - 10^9]$ room no.

bool room $[10^9+1]$ → Issues : huge space wastage

$SC = O(10^9)$ → Advantage : $TC = O(1)$

Hashmap : stores <key ,value > pair

<10015, occupied>

<3, free>

<1007, occupied>

} → check is 1007 ?

→ occupied in O(1) TC

TC : O(1) to search

SC : O(N) to store N room entries

↳ 1000 in our ex.

Hashmap<int, bool>

Note : Keys are unique

value can be anything

Quiz1 Store population of every country

Key : country name → string

value: population → int/long

Hashmap<String , Long> hm

Quiz No. of states of every country

Key : country name → String

value : # of states → int

Hashmap< String , int>

Quiz Name of all states of every country

Key : country name → String

value : all state names → list<String>

↳ C++ : vector

java : ArrayList

<India, UP>

<India, MP> ✗ keys have to unique

Hashmap< String , List< String > > hm

Quiz Population of each state in every country

Key : country name → String

value : population of each state } → Hashmap < String, long>

↑ state name ↑ population

Hashmap< String , Hashmap< String, long > > hm

we observe 2 things :

1. value can be anything

2. Key can only be primitive datatype

↙

int / long / float / double / string / char

Hashset <Key>

→ it only store keys

→ keys have to be unique

→ only be primitive datatype

Hashmap functionality

size : {# keys present}

insert(key , value)

search (key)

delete (key)

update(key , value)

Hashset functionality

size : {# keys present}

insert(key)

search(key) {present or not}

delete (key)

All operations are $O(1)$

→ Hashing libraries name in diff. languages

Pseudocode	Java	C++	Python	JS	C#
hashmap	hashmap	unordered_map	dict	map	dictionary
hashset	hashset	unordered-set	set	set	Hashset

Question 1

Given N array elements & Q queries.

for each query, find the frequency of given element in array.

$a[11] = \{2 \ 6 \ 3 \ 8 \ 2 \ 8 \ 2 \ 3 \ 8 \ 10 \ 6\}$

Q=4	freq
2	3
8	3
3	2

constraints

$N <= 10^5$ $Q <= 10^5$

$1 <= a[i] <= 10^9$

5 0

Bruteforce : for every query , iterate & get count

TC = O(Q×N) SC = O(1) not feasible

Optimize Store freq. in hashmap

Key : array element → int

value : freq. of element → int

Hashmap< int , int >

{ 2 6 3 8 2 8 2 3 8 10 6 }

<2,3> <3,2>

<6,2> <8,3>

<10,1>

Code

```
Hashmap <int, int> hm

for (i=0 to n-1) {          → N iterations
    if( hm.search(a[i]) == true) {
        //a[i] is present
        hm[a[i]]++      //update +1

    }
    else {
        hm.insert(a[i], 1)
    }
}

for( i=0 to Q-1) {          → Q iterations
    if (hm. search (query[i]) == true) {
        print( hm[query[i]])

    }
    else {
        print(0)
    }
}
```

TC = $O(N+Q)$

SC = $O(N)$

Question 2

Given N elements , find no. of distinct elements.

a[5] = { $\overset{\checkmark}{3}$ $\overset{\checkmark}{5}$ $\overset{\checkmark}{6}$ 5 $\overset{\checkmark}{4}$ }   ans = 4

Insert everything in hashset & print its size.

Hashset will not insert duplicate entries.

code

```
Hashset<int>  hs

for (i=0  to  n-1) {
      hs.insert (a[i])
}

print ( hs. size )
```

TC : O(N)

SC : O(N)

Question 3

Given a string s, find the length of the longest substring without repeating chars.

s = "abcabcbb"   ans=3

s = "bbbb"   ans = 1

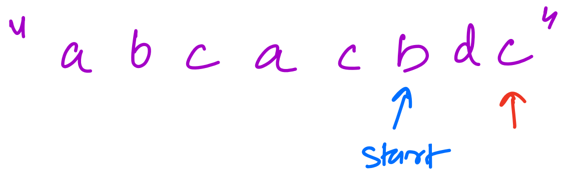
Bruteforce Check all substrings for uniqueness & keep track of max. length.

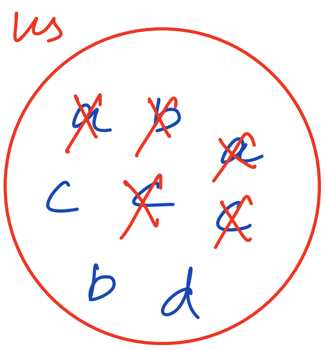
substrings → $O(N^2)$ # total substrings

for each substring → $O(N)$ (insert everything in hashset &
if hashset.size = substring size
then unique)

TC = $O(N^3)$

SC = $O(N)$





ans = ~~1~~ ~~2~~ ~~3~~ 4

Code

```
int  longest Unique Substring ( s) {

    int  start = 0

    int  maxlength = 0

    hashset < char>  hs

    for ( end = 0  to  n-1 ) {

        while ( hs . search ( a[end] ) == true ) {

            hs.delete ( a[start] )

            start++

        }

        hs.insert ( a[end] )

        maxlength = max( maxlength , end-start+1 )
                                        or hs.size()
}
```

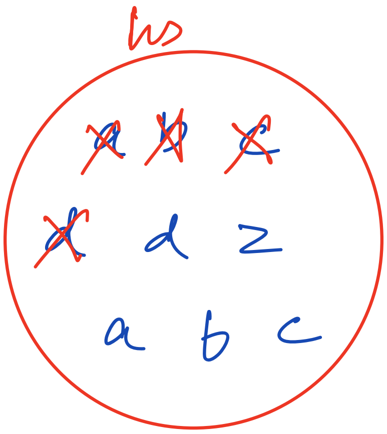
TC = O(N)

SC = O(N)

return max length

ex a b c d d z a b c

start ↑ ↑



ans = ~~1~~ ~~2~~ ~~3~~ 4 5

TC Analysis

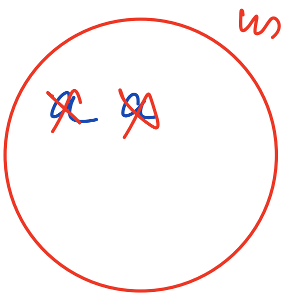
we are inserting every char max 1 time

⇒ total N insertions

⇒ we can only delete total N times

total TC = O(N)

eg "a a a a a a a"
↑



end = 0 while → 0 times

end = 1 while → 1 time

=2 → 1 times

.

⋮

n-1 → 1 time

total N times

eg a b c d e e
↑

ws

~~a~~ ~~b~~ ~~c~~
~~d~~ ~~e~~ e

end=0	while →	0 time
=1	→	0 tim
=2		0
=3		0
=4		0
=5		5 times

total N times

i	j
0	+0
1	+0
⋮	+
⋮	⋮
N-1	N

$\boxed{N}$