Hashing | Maps | Time Complexity | Collisions | Division Rule of Hashing

Hashing

1	2	1	3	2

How many times 1 appears = 2
" " 2 " = 1
" " 4 " = 0

```
int f(number, a[]) {
    cnt = 0;
    for(i=0; i<n; i++){
        if (arr[i] == number) {
            cnt++;
        }
    }
    return cnt;
}
```

Time complexity : O(N)

f(1, a[])
f(3, a[])
:
:
Q

Time complexity O(Q × N)

(10^8 operations take 1 sec to execute)

pre-storing / fetching

0	2	2	1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

hash array

pre-calculation → in index 1 & 2, it appears thrice

so as we traverse the array, the count (pre-calculation) is done / updated.

hash[1] = 2
hash[2] = 2
hash[3] = 1
hash[10] = 0

```
#include<iostream>
using namespace std;
int main(){
    int n;   ← 5
    cin>>n;
    int arr[n];   [1,3,2,1,3]
    for(i=0; i<n; i++) cin>>arr[i];

    // precompute
    int hash[13];   // say
        ={0}
    for(int i=0; i<n; i++){
      hash[arr[i]] += 1;
    }

    int q;   ← 5
    cin>>q;
    while(q--){
      int number;
      cin>>number;
      // fetch
      cout<<hash[number]<<endl;
    }
    return 0;
}
```

hash size = 12 (12 elements)

[0,0,0,0,0,0,0,0,0,0,0,0]

→ hash[1] += 1

[0,1,0,0,0,0,0,0,0,0,0,0]

hash[3] += 1

hash[2] += 1

hash[1] += 1

hash[3] += 1

input	output
1 →	2
4 →	0
2 →	1
3 →	2
12 →	0

Note:- max array size is 10^7 ✓

arr[10^9+1] ✗ segmentation fault

(size)

Inside main, max array size is 10^6 because if size is greater than that it cannot allocate that much memory size.

Note: If array is declared globally i.e. outside above main function, the size can be max 10^7 → arr[10^7] i.e. (for int)

★ CHARACTER HASHING

Using arrays

S = " a b c d a b e f c "

(only lowercase letters say)

a appears 2

a → 2
c → 2
z → 0

Q queries can be asked.

Method 1 - using arrays

```
f(char c, s){
    cnt=0;
    for(int i=0; i<n; i++){
        if(s[i]==c)
            cnt++;
    }
    return cnt;
}
```

T.C = O(Q × N)

Method 2 - Using hashing

```
int n = 'a';
```

a's ASCII value is 97 and this value is stored in int n.

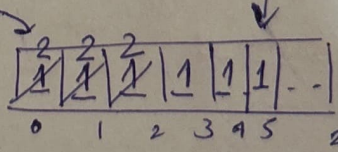
'a' → 97
'z' → 122

f → 'f' - 'a' = 102 - 97 = 5 → ch - 'a'

a → 'a' - 'a' = 0

no. of lowee case letters = 26

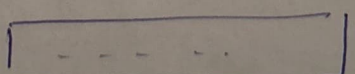
arr[26]



(hashing method)

★ In case string had other characters as well - just take full ASCII table i.e 256 characters arr[256]

'a' → 97
'b' → 98



256 size array

★ Drawback in Number Hashing

In number hashing, hashing into arrays can only be done to at max 10^7. Thus for numbers having size $> 10^7$ i.e 10^9, 10^{12} etc, we cannot hash it using arrays.

Thus to solve this, in C++, STL comes in & in Java, Collection comes in.

- STL → 1) map
  2) unordered map → pair<int, int> cannot be used here.
- Collection → HashMap.

★ MAPS

arr = 1, 2, 3, 1, 3, 2, 12 unordered-map

map < int key, int value >

key → number, value → frequency

```
mpp[1]++;
mpp{arr[i]}++
mpp[4] = 0.
```
?

(12→1)
(3→1)2
(2→1)2
(1→1)2

Map will just store the elements that are req. Thus a full sized array is not required to be declared.

Thus it is a bit faster & efficient. ~~than~~

Note: The map stores all the values in sorted order.

```
#include<iostream>
using namespace std;
int main(){
    int n; int arr[n];
    cin>>n;
    for(int i=0; i<n; i++){
        cin>>arr[i];
    }

// pre-computation of frequency of each element
    map<int,int> mpp;
    for(int i=0; i<n; i++){
        mpp[arr[i]];
    }
// print OR iterate in the map
    int q;
    cin>>q;
    while(q--){
        int num;
        cin>>num;
        cout<<mpp[num]<<endl;
    }
    return 0;
}
```

initializes array of size 'n'

e.g. (5) [1, 2, 2, 3, 3]

a map 'mpp' used to store freq of each element in array

1 → 1
2 → 2
3 → 2

(key, value pair)

→ q → no. of queries to be inputted by user. (3)

user inputs an integer 'num' and for each num, outputs a freq of 'num' by fetching it from map mpp[num]

2, 2, 0

```
for(auto it : mpp){
    cout<<it.first<<" -> "<<it.second<<endl;
}
```

↑ same method used in character hashing question. (see code)

Time Complexity → map

Storing & fetching in a map → $\log N$ (N = no. of elements in a map) for all cases

best / worst / avg

Unordered Map

3 → 2
12 → 1
2 → 2
4 → 1
} not in order.

In unordered map, for storing & fetching →

average case → O(1)
best case → O(1)

worst case → O(N) (N = no. of elements in map)

very rare case

↓

happens because of collisions

★ HASHING METHODS

- Division Method (LINEAR CHAINING)
- Folding Method
- Mid Square Method

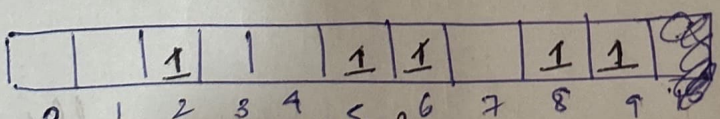
} not required.

{ 2, 5, 16, 28, 139 }

say array > size 10 not allowed

arr[i] % 10

{ 2, 5, 6, 8, 9 }



{2, 5, 16, 28, 139, 38, 48, 28, 18}

139 % 10 = 9

0
1
2 → 2
3
4
5 → 5
6 → 16
7
8 → 28 → 38 → 48 ~~→ 28~~
9 → 139

If someone asks how many times 28 appears, apply search algo on 1/8 line and thus the task would be performed in minimal time.

Collision

18, 28, 38, 48 ..... , 1008

0
1
2
.
.
.
(8 → 18 → 28 → ..... ∞) very lengthy & takes long time

↓

COLLISION takes place

(all keys end up at the same hash index)