Basic Hashing

Date
Page

* Suppose you have an array

1	2	3	2	7

check how many times "2" appears!

```
function(number arr)
{  count = 0
   for (i=1 to size of array-1)
   {   if(i==number)
       { count++; }
   }
   return count;
}
```

TC : O(n)

* So to find all the element's appearance below & count

~~TC : O(n^2)~~ for ex. Q = 0, 1, 2, 3, 4, 5, 6 ...

so then TC. would be [Q × O(n)]

Num. of elements

for example [Q = 10^5] & [Num. of array elements = 10^5]

TC = Q × O(n)

TC = $10^5 \times 10^5$

TC = 10^{20}

$10^8 \approx$ 1sec.

$10^9 \approx$ 10sec.

$10^{20} \approx$ 200sec.

one operation will take almost 2 minutes which is very time taking.

(Not Recommended)

So here comes **Hashing**

* Hashing :- restoring & fetching

question

for example [problem statement] array will have elements at most number 12.

restoring

Hash array :-

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

n = 13.

precalculation :-

array

1	2	1	3	2

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

[Number of Appearance]

frequencies

* check How many times 1 appears?

Hash[1] = (2) ✓ fetching

TC : O(1) SC : O(n)

Maxsize :- Of Hash array

Inside of Int main() 10^6 → 1000000

outside of Int main() (Globally) 10^7 → 10000000

* Character Hashing

question

s = "abcdabefc"

frequency of

a →	2
c →	2
z →	0

again same logic

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

~~If do for n characters~~
~~(int main()~~
~~{ for(...)~~

again TC would be $Q \times O(n)$

So

s = "abcdabefc"

lowercase :- a, b, c, ..., z → 26 letters

How to store

Hash Array	0	0	0 ...	0	0	0	0
	0	1	2 3 ...	23	24	25	
	a	b	c d ...	x	y	z	

ASCII value :- 97 to 122 (a to z)

to store

f = ASCII of f - ASCII of a
= 102 - 97
f = 5

a = ASCII of a - ASCII of a
= 97 - 97
a = 0

hash[str[i] - 'a'];

prestoring

fetching is easy

hash[c - 'a']

If you have smaller & capital letters both you have to use Hash array of 256 length and then you can directly do as you have

maxlength → Globally 10^7,
(size) for $10^8, 10^9, 10^{10}$... → we can't use array (X)
you have to use STL

Basic Understanding

Map / Unordered map

working of unordered map

map (key, value) — num. freq.

arr = | 1 | 2 | 3 | 1 | 3 | 2 | 12 |

arr (bucket):
(12 → 1)
(3 → 2)
(2 → 2)
(1 → 2)

⟹ It doesn't stores whole array of elements.
It saves memory.
only numbers appears atleast once are only get stored.
If i check arr(13) → It will directly give me 0.

Task :- Do Number Hashing using maps.

TC :- (of maps) O(logn) in all cases {avg, best, worst}

Unordered map

TC :- O(1) {avg, Best} worst O(n)

Recommended :- Unordered map (1st preference)
map (2nd preference)

working of unordered map also

hash[str[i]]++; instead of hash[str[i] - 'a'];

Summary * count freq. using [Naive Nested for loops] TC: n x O(n)

(1) Number Hashing using Hash array } better

(2) character Hashing using Hash array }

↳ limitations :- 10^7 (globally) Maximum Input size

so [better option] ↘

(3) Number & character Hashing using C++ STL → maps → O(log n)

→ unordered maps

Best avg: O(1) | worst: O(n)

अब आगे...

maps	multimaps	unordered maps
• sorted & ordered • each key contains unique elements • log n • Implementation: Balanced BST (Red black tree)	• each key can contain duplicate values • sorted/ordered	• each key contains unique values • unordered Best & Avg. O(1) worst O(n) • Implementation: Hash tables.

Inner working of unordered maps

Before that let's learn one of the Hash methods (Division method)

array = [11, 29, 88, 76, 54, 53, 91]

h(x) = x % 10

Bucket	
0	0 →
1	1 → 11 → 91
2	2 →
3	3 → 53
4	4 → 54
5	5 →
6	6 → 76
7	7 →
8	8 → 88
9	9 → 29

← collision

load factor = n/b

number of elements / num of keys

Handle collision
2 methods → separate chaining
→ open addressing
↓
Probing

unordered maps uses this

separate chaining → creates chain values using ll (linkedlist)
means avg case O(1)
worst case O(n)

load factor here is (0.7) means each bucket is on an avg 70% full, typically good balance. it is.
If it is Higher than 1 → you have to do rehashing.

Hashing In unordered maps