

MAP → are associative container / and
it's non-sequential data, it store the
element corresponding to the its key

A key value has it map value.
and map value has to be unique
according to keys

for example	key value	map value
	[1]	= "abc"
	[2]	= "FGH"
	[1]	= "abc"

key value has to be unique ~~who~~ we
can't use same key value if we
then it will replace the previous
map value to new one

- ⇒ begin()
- end()
- size()
- max_size()
- empty()
- insert (key value, map value)
- erase (iterator position)
- clear()

Time complexity of insertion → $O(\log N)$
Find/traverse → $O(\log N)$

void learn-map()

{

Syntax ->

map<int, string> A;

A[1] = {"ABC"};

A[2] = {"DEF"};

A[4] = {"GHI"};

A[3] = {"JKL"};

for (auto it : A)

{

cout << it.first << " " << it.second;

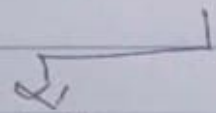
}

output -> A[1] = {"ABC"}

A[2] = {"DEF"}

A[3] = {"JKL"}

A[4] = {"GHI"}



keys ~~map~~ ^{has} to be always unique
and it will be in sorted
order after execution in map

Input
ABC
ABC
ED
GH
ED

Q1

Give N strings, Print unique string in lexicography order with their frequency.

void solution ()

```

{
    int N;
    cin >> N;

    map <string, int> A;

    for (int i = 1; i <= N; i++)
    {
        string s;
        cin >> s;
        A[s]++;
    }

    for (auto it : A)
    {
        cout << it->first << " " << it->second;
    }
}

```

Input

ABC

ABC

EDF

GHI

EDF

Output

ABC 2

EDF 2

GHI 1

we store that in
map A[s]

A[s] → key
++ → value

Q2

Leetcode 169

find majority of element in given array
if element is appear more than
 $n/2$ ($\text{length}/2$) then it is majority
element

Input

↓

1
1
2
3
1
1

Output

1

length is 6 = $6/2 = 3$

1 is appear more than
3 times

void solution ()

{

map <int, int> A;

int N;

cin >> N;

for (int i=0; i<N; i++)

{

int x;

cin >> x;

A[x];

A[x] = A[x] + 1;

if (A[x] > N/2)

{

cout << x;

}

}

}

*** **

Key Point's

- ① This Ordered map has time complexity of $O(\log(n))$ of ~~every~~ ^{insertion/access} operation

If we deal with string so it's time complexity will affect because string can be of any size

Insertion operation

- ② It's internally work on the base of Red-black-Tree (advance data structure)

- ③ You can take any data type/utility class in it like pair, List etc.

- ④ Its key is always unique and sorted

- unordered_map \rightarrow It is ^{some} like unordered map all the function are same almost except few

Difference: ~~just~~ are \rightarrow

InBuilt implementation \rightarrow It work on ^{hashable} ~~hash~~

Time complexity $\rightarrow O(1)$ linear time

valid key - dat types \rightarrow you can't use

complex data structure in it like vector, pair, list etc.

- \rightarrow key is not in sorted form (unordered)
- \rightarrow used unordered map if you don't want sorted key because it's faster than map

Q. Give N strings and Q queries.
In each query you are given a string
Print frequency of that string.

void solution ()

{

unordered_map<string, int> A;

int N;

cin >> N;

for (int i = 0; i < N; i++)

{

int S;

cin >> S;

A[S]++;

}

int Q;

cin >> Q;

while (Q--)

{

int S;

cin >> S;

cout << A[S];

cout << endl;

}

Input

8

abc

DEF

abc

GHI

abc

GHI

LMN

abc

Query

abc

GHI

Output

4

2

- Multi-map
 multi-map is same like map
 but difference is in map key
 are unique but in multi-map key
 is not unique we can store
 same multi value / key