Smile Often.
Think positively.
Give thanks.
Laugh Loudly.
Love others.
Dream Big.

Problem: Find if there is a subsequence, where gcd is 1

Note: if GCD(a, b) = 1, means a and b has only one common factor. Hence a, b are Co-prime could be prime numbers, which have only 2 factors

$A = \{6, 3, 9, 17\}$

3 3 1

17
1 17

Brute force:

1- going through each pairs and finding it.

TC: O(N)

Agenda!

1. Intro to Hash mapping
2. Given queries, Find frequency of an element
3. no. of Distinct Element.
4. Longest substring without repeat.

Note: we cannot declare array size greater than 10^6 or 10^7

Array:

Cons

Pros: access using indexing in constant time.

Hash mapping!

int boolean

Java: Hashmap <key, value>

any any

Python: { key : value }

access
insert
delete
check
update

} O(1) amortized

- for most of the time operation is constant.
- but in some time it can be Linear time

1. we can not have duplicate "keys"

Java:

```
HashMap<Integer, Integer> hm = new HashMap<>();
```

Python:

```
hashmap = { }
```

C++:

```
unordered_map<int, int> um;
```

Hash set:

1. when we need to store only keys.

Java:

Python:

```
my_set = Set()
my_set.add("Syed")
```

unordered map

Hash.map

hm.put(key, value)

hm.remove(key)

hm.get(key)

hm.contains(key)

hm.clear()

hm.size()

Python:

```
if key in hm:
    print hm[key]
```

remove?

```
del hm[key]
```

Accessing keys only:

```
hm.keys()
```

Accessing values:

```
hm.values()
```

merging dictionaries:

```
hm1 = {a:1, b:2}
hm2 = {c:3, d:4}
merged = {**hm1, **hm2}
```

HashSet

hs.add(key)

hs.remove(key)

hs.clear()

hs.contains(key)

hs.size() / len(hs)

un-ordered collection of unique elements

hs.add(key)

removing:

hs.remove(key) Throws error if not found

hs.discard(key) Does not throw error if not found

Copying dictionary:

```
new_dic = hm.copy()
```

Problem 1:

→ Given N elements and Q queries.
→ Find the frequency of elements provided in the query

A = [2 6 3 8 2 8 2 3 8 10 6] N = 11

Q = [2 8 3 5] Q = 4

Brute force: O(Q x N)

1. iterate through the array Q and checking
2. the frequency of the element in A.

Hashmap method: O(N+Q)

1. iterate A and store the element frequence in map
2. iterate array Q and chek the freq in map

```
freMap = { }  # create freMap will and upate
for a in A:
    if a in freMap:
        freMap[a] += 1
    else:
        freqMap[a] = 1
# iterate quires and check.
for i in range(N):
    if A[i] in freMap:
        ...
```

count = freMap.get(key, 0) + 1
freMap[key] = count

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

else :

Problem 2 : Count of distinct elements.

Given int[] A, Return Count of distinct elements.

A = [1, 1, 1, 2, 2] ans = 2

[1, 2]

Hashmap Approch :

1. iterate over array A and store freqency of elements.
2. return `len(hm.keys())`

using Set :

1. iterate over array A and hs.add(ith elem)
2. return `len(hs)`

Unique elements

Problem 3: Longest Substring without Repeat

A = "abcabcbb" ans = 3

A = "aaaa" ans = 2

hash map approch:

1. store each element in hash map
2. if element already in hashmap.
3. break loop and return length.

using hashset:

Same steps with set()

Brute force:

1. going through all substrings,
2. if no duplicates in substring.
3. mantain maximum length.

1. Sysde
2. Convert it into set.
3. Compare both length.

TC:

1. There are N^2 substrings.
2. going through substring again $O(N)$

$O(N^3)$

SC: $O(N)$, as we use hashset to check length.

Dynamic Sliding window approch

1. from i->(j-1) are unique.
2. reset i if duplicat found. and update length.

i = 0 ansCount = 1
j = 1
current = (A[0],) #set

while (j<N):

    if A[j] in current:

        length = ((j-1) - i + 1)
        ansCount = max(ansCount, length)
        i = j-1 , current_set.clear()
                  To start new

    else:
        current.add(A[j])

    j++

ansCount = max(ansCount, (j-i+1))

return ansCount

i j k e m p k m a b c r k

length = ((j-1) - i + 1) = 6
                          = 7

Problem 4:

First non-repeating Element

=> Given N elements.

=> Find First non repeating Element.

A = [1, 2, 3, 1, 2, 5] ans = 3

Brute force:

1. iterate over A
2. for every element.
3. Count the freqency in array using nested loop
4. if it is 1 return it.

hash map:

1. create freMap
2. iterate over freMap keys.
3. if value is 1 return it.

What are different types of maps?

1. Tree map
2. Linked Hash map

=> Given int[] A, int[] B

=> Find freq of each element from arr B in arr A.

=> return a list containing all fre

$1 \leq \|A\| \leq 10^5$ $1 \leq \|B\| \leq 10^5$	$1 \leq A[i] \leq 10^5$ $1 \leq B[i] \leq 10^5$

ex: A = [1, 2, 1, 1] B = [1, 2]

1 → 3

2 → 1

ans [3, 2]

```
freqMap = { }
for i in rang (len(A)):
    value = freqmap.get(A[i], 0)
    freqMap[A[i]] = value + 1
ans = [ ]
for i in range (len(B))
    ans.append(freqMap[B[i]])
return ans
```

Count distinct element:

⇒ Given array A

⇒ return the number of unique elements in array.

$1 \leq N \leq 10^5$

$1 \leq A[i] \leq 10^9$

A = [3, 4, 3, 6, 6] ans = 3

```
ans = set()
for a in A:
    ans.add(a)
return len(ans)
```

③ Common Elements

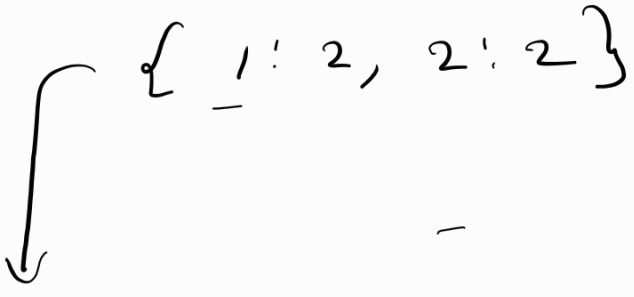
⇒ Given two int[] A and B
Find all common elements

set A
set B

A = [1 2 2 1]
B = [2 3 1 2]

ans = [2 2 1]

Count freq in A and
" in B



Take and iterate on keys.
check in B
if value is > 0.
↓ add to ans
reduce value by 1

get freqMap of A

```
for a in A:
    value = freqmap.get(a, 0)
    freqMap[a] = value + 1
```

#iterate over array B

```
ans = [ ]
for b in B:
    value = freqmap.get(b, 0)
    if value > 0:
        value -= 1
        ans.append(b)
        freqMap[b] = value
return ans
```

④ Longest Substring without Repeat

String A = "AcaA" ans = "Ac"

using sliding window with set()

i j

"bbb" "Abc"

i=0

j=2

ansCount=1

substring = b

while j<n)

if A[j] in substring

length = (j-1)-i+1

j-1

i j

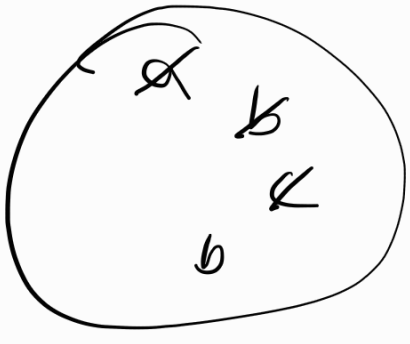
"dadbc" ans = 4

Bruteforce!

i

Abcbc

j



Count Unique element.

⇒ given array A.

⇒ return count of element with frequency 1

```
      0  1  2  3  4
A = [ 3, 4, 3, 6, 6 ]
```

ans

freqArr = { }

freqArr [a] =

ans: 4

index: 1

if index > _ [-1] ?

↓ ans = ...

create freqMap

```
freqMap = { }
for i in range(n)
    value = freqMap.get(A[i], [])
    if len(value) == 0:
        temp = [1, i]
        freqMap[A[i]] = temp
    else:
        value[0] = value[0] + 1
        value.append(i)
        freqMap[A[i]] = value
```

{ 3 : [2, 0, 2]
  4 : [1, 1]
  6 : [2, 3, 3]
  1 : [1, 0]
}

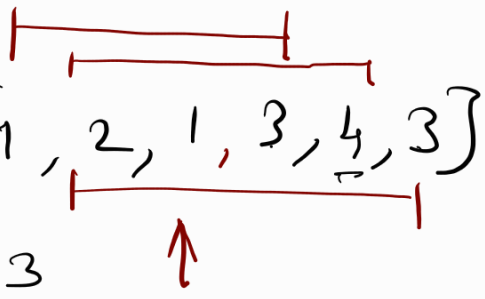
~~[1, 0]~~

```
# element = ?, index = -1
for key in freqMap.keys():
    if freqMap[key][0] == 1:
        idx = freqMap[key][-1]
        if index == -1:
            index = idx
            element = key
        else:
            if index > idx:
                index = idx
                element = key
return element.
```

Distinct number in window.

1. Given int[] A, int B.
2. return distinct elements count in all window size B.
3. return int[] of size (N - B + 1) where ith element in this array contains distinct element count.

> if B > N return empty array



A = [1, 2, 1, 3, 4, 3] N = 6

B = 3

=> N - B + 1

=> 6 - 3 + 1

=> 4 => Num of window of size 6

0 1 2 3

[2]

freqMap = {

1 : 2

2 : 1

3 : 1

}

d = N - B + 1 i = 1

while (j < d) : i++ j++

remove(i - 1)

value = freqMap.get(A[i-1], 0) - 1

if value < 0 : value = 0

Brute_force : $O(N^3)$

=> going through all the substring of size B $O(N^2)$

=> check if substring has distinct element $O(N)$ using set.

```
for i in range(N)                    # start
    for j in range(N)                # end
        stringLength = j - i + 1
        if stringLength == B:
            temp = { }
            for k in range(i, j+1)
                value = temp.get(A[k], 0)
                temp[A[k]] = value + 1
            for k in temp.keys():
                if temp[key] == 1
                ans.append
```

optimzed using sliding windows :

```
   0   1   2   3
[  2                ]        freqmap = {
```

d = N - B + 1 i = 1

while (j < d) : i++ j++

remove (i - 1)

value = freq map. get (A[i-1], 0) - 1

if value < 0 : value = 0

1 : 2
2 : 1
3 : 1
}

get the freq first substring of window

freq map = { }

result = [ ]

for i in range (B)

value = freqmap. set (A[i], 0)

freqmap [A[i]] = value + 1

result. append (len(freqmap))

i, j, d = 1, B, (N-B+1)

while (i < d): i++, j++

remove, del if count == 0.

freq map. get (A[i-1], 0)

if ivalue <= 1 :

↓ del freqmap[A[i-1]]

else:

freqmap[A[i-1]] = value - 1

Decrement j

freq[A[j]] = freqmap.get(A[j], 0) + 1

result.append(len(freqmap))

TC: O(N)

SC: O(B)

0	1	2	3	4	5
1	2	1	3	4	3
	i				

B = 3

i = 1

j =

d = 4 - 1 = 3

$\frac{6-3+1}{4}$

d = 4