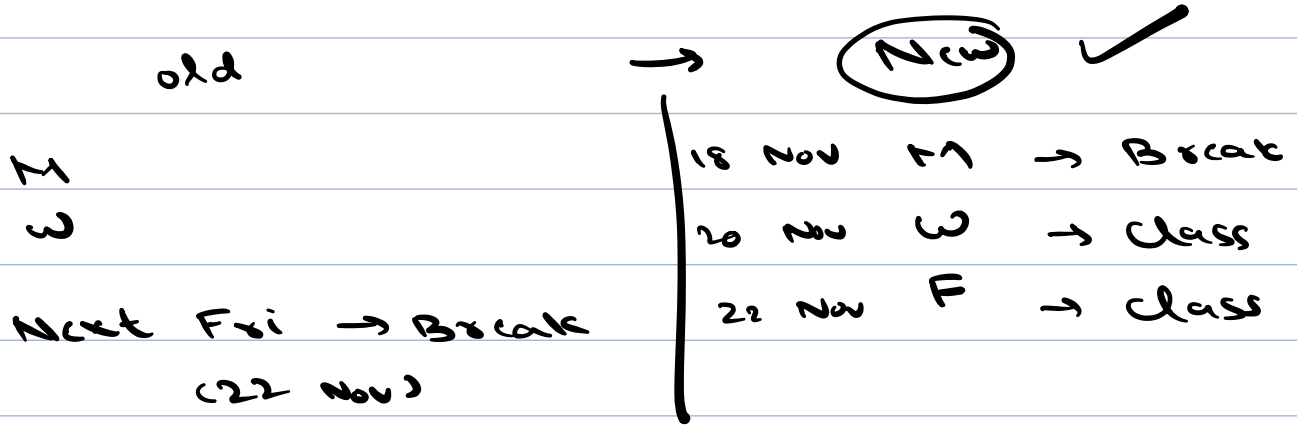
Agenda

1. Longest substring without Repeating Characters
2. First non repeating Element
3. Subarray with sum 0
4. Subarray with sum k



1. Given a String s, find length of the longest substring without repeating characters.

str : "abcabcbb" ans : 3

str : "bbbbb" ans : 1

str : "pwwkew" ans : 3

Brute Force : Go to all substrings, check if substr is valid (without duplicates), compare its length with ans and keep max in ans.

= N^2 substr

```
int ans = 0
for (s ——————— ) {
    for (e ——————— ) {
        // s e
        for (i = s ; i <= e ; i++)
            put all chars in hs
        if (hs.size() == e-s+1)
            ans = max(ans, e-s+1)
    }
}
```

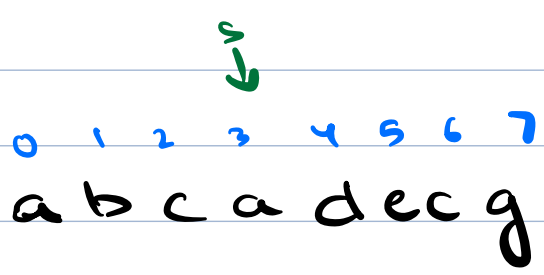
[a b] = b - a + 1

TC : $O(N^3)$

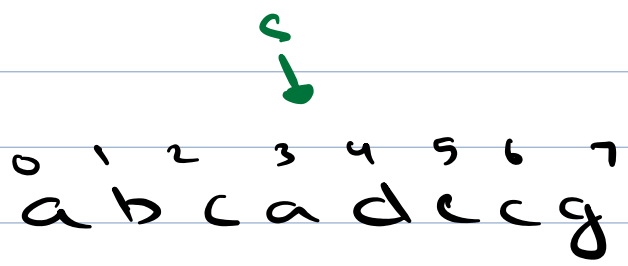
SC : $O(N)$

Optimized :

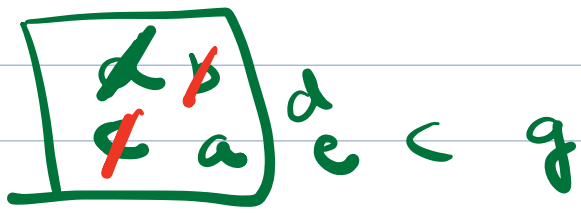
ans = 5 bcade



ans = ~~1 2 3 4~~ 5



ans = ~~1 2 3 4~~ 5




```
// Str, int n
int ans = 0
Hashset <char> hs
int s = 0
for (e = 0 ; e < n ; e++) {
    while (hs.contains(str[e] == true){
        hs.remove(str[s]
        s++
    }
    hs.add(str[e])
    ans = max (ans, hs.size())
}
return ans
```

substr size

TC : O(N)

Every char can be processed twice, added once and removed once from hashset

SC: O(min(N, M))

Str → a to z

M → size of character set
(ASCII → 128 chars)

2. Find the first unique non-repeating element. from start

Ex 1 ar[6] = < 1 2 3 1 2 5 > ans = 3

Ex 2 ar[8] = < 4 3 3 2 5 6 4 5 > ans = 2

Ex 3 ar[7] = < 2 6 8 4 7 2 9 > ans = 6

Idea:

1. Insert all elements in hm
2. Iterate on hm and get 1st key with freq = 1

[Note - in hashmap, insertion order is not maintained; when we print hashmap we'll get any order)

Idea:

1. Insert all elements in hm
2. Iterate on array and get elem with freq = 1

// int arr[], int N

```
Hashmap <int, int> hm
for (i=0 ; i<n ; i++) {
    if (hm.contains(arr[i]) == true)
        hm[arr[i]]++
    else {
        hm.insert(arr[i], 1)
    }
}

for (i=0 ; i < N ; i++) {
    int freq = hm[arr[i]]
    if (freq == 1)
        return arr[i]
}

return -1  // no element is unique
```

10:35

TC : O(N)

SC : O(N)

3. Given an array of N elements, check if there exists a subarray with sum equal to 0.

		0	1	2	3	4	5	6	7	8	9
N = 10	ar :	2	2	1	-3	4	3	1	-2	-3	2

ans : true

N = 10 ar : 2 3 4 -2

ans : false

Brute Force : Go to all subarrays, iterate and calculate sum. If any subarray has 0 sum, return true otherwise at the end return false

1 subarray → O(N) (Iterate and get sum)

N^2 " → $O(N^2 * N)$

TC : $O(N^3)$

SC : O(1)

bf

TC : $O(N + N^2)$
$\approx O(N^2)$

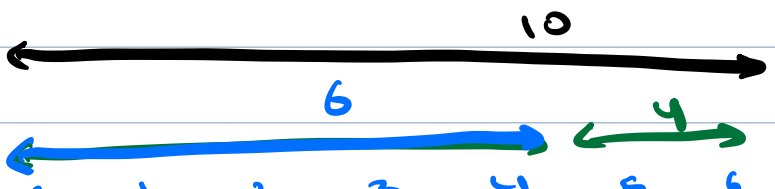
SC : O(N)

carry forward

TC : $O(N^2)$

SC : O(1)

Optimization: Sum of a subarray : Prefix Sum



	0	1	2	3	4	5	6	7	8	9
ar:	2	2	1	-3	4	3	1	-2	-3	2
pf:	2	4	5	2	6	9	10	8	5	7

* Subarray

sum(l → r) = 0

sum(0 → r)

pf[r] = 0

sum(l → r)

pf[r] - pf[l-1] = 0

⇒ pf[r] = pf[l-1]

Duplicates in pf[]

	0	1	2	3	4	5	6	7	8	9
ar:	2	2	1	-3	4	3	1	-2	-3	2
pf:	2	4	5	2	6	9	10	8	5	7

// int arr[], int n

```
Hashset<int> hs;   int sum=0
for(int i=0 ; i<n ; i++){
    sum = sum + arr[i]
    hs.insert(sum)
}
```

```
if((hs.size() < N) || (hs.contains(0) == true))
        return true
else
        return false
```

TC : O(N)

SC : O(N)

↓

hashset

4. Given an array of N integers, check if there exists a subarray with sum k.

	0	1	2	3	4	5	6	7	8
a[ ] =	2	3	9	-4	1	5	6	2	5

k = 11 true <2, 3, 9, -4, 1> <5, 6>

k = 15 true <-4, 1, 5, 6, 2, 5>

a : [5 10 20 100 105] k = 110 false

Optimization: Sum of a subarray : Prefix Sum

subarray sum (l → r) = k

sum (0 → r)

pf[r] = k

sum (l → r)

pf[r] - pf[l-1] = k

a - b = k

A pair in pf[] whose diff is k

→ b = a - k

	0	1	2	3	4	5	6	7	8	
a[ ] =	2	3	9	-4	1	5	6	2	5	K = 11

pf → 2 5 14 10 11* 16

partner → -9 -6 3 -1 0 5 ✓

x x x x x

HS (all pfsum() of past)

2 5

14 10

11

* pf[i] = K = 11

sum(0-i) = K

```
// int arr[], int N, int K
HashSet<int> hs        // pfsum[]
int sum = 0
for (int i=0 ; i<n ; i++){
    sum = sum + arr[i]
    if (sum == K)
        return true
    if (hs.contains(sum - K) == true)
        return true
    hs.insert(sum)
}
return false
```

pf[i] = K

TC : O(N)

SC : O(N) + hashset

Doubts

k = 1

2 1 2 1

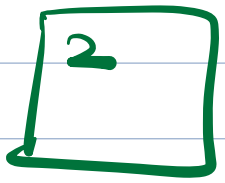
a - k

pb 2 3

↓ ↓

1 2

✓



① Assignment

Approach → Pen & Paper

Notes / Ex

↓

Code