Todays Content:

→ longest substring with all distinct characters

→ Permutations of A in B

→ Treemap / TreeSet Intro & Problems ✓

1Q) length of **longest substring** with all distinct characters?

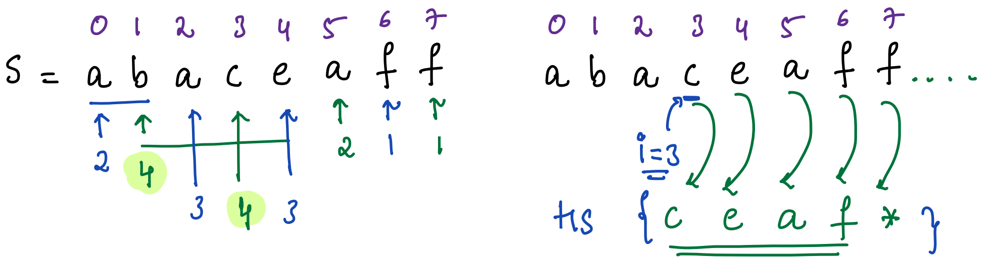
```
      0 1 2 3 4 5 6 7
s1 =  a b c a b c d d    ans = 4 ✓

      0 1 2 3 4 5 6
s2 =  s i p p i e r      ans = 4 ✓

      0 1 2 3 4
s3 =  a a a a a          ans = 1 ✓
```

Idea1: For every substring check, if it contains all distinct characters

```
int londis( String s){  TC: O(N3)  SC: O(N)
    i = 0; i < N; i++){ // i = start of substring
        j = i; j < N; j++){ // j = end of substring
            [ [i - j] substring, insert all
              characters in hashset hs ]  TC: O(N)
            if hs.size() == j-i+1 // all distinct characters
                ans = man(ans, j-i+1)
            }
        }
    }
}
```

Idea2: With every s[i] as start of substring, get max length which contains all distinct characters

Ex:



```
int lonDis( String s){  TC: O(N^2)  SC: O(N)
    i = 0; i < n; i++){ //Start substring at i
        hashset<char> hs;
        j = i; j < n; j++){
            if( hs.search( s[j]) == false){
                hs.insert( s[j])
            }
            else { break }
        }
        ans = max (ans, hs.size())
    }
    return ans;
}
```

Idea3:

$$\begin{array}{cccccccccccccccc} & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ S & = & a & b & c & g & h & e & g & k & l & m & h & a & b & h \\ & & & & & & & & & & & & & i & & & j \end{array}$$

HashSet:

~~a~~ ~~b~~ ~~c~~ ~~g~~
~~h~~ ~~e~~ ~~g~~ ~~k~~
~~l~~ ~~m~~ ~~h~~ a
b , h

i	j	
0	0	
0	1	1, 1
0	2	1, 2
0	3	1, 3
0	4	1, 4
0	5	1, 5

0 6 → 1, 6 → 2, 6 → 3, 6 → 4, 6

4, 6 → 4, 7 → 4, 8 → 4, 9 → 4, 10

4, 10 → 5, 10 → 5, 11 → 5, 12 → 5, 13

5, 13 → 6, 13 → 7, 13 → 8, 13 → 9, 13

10, 13 → 11, 13 → 11, 14 : j is our stop

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

```
londis (String s){
    i=0, j=0, ans=0
    hashset<char> hs
    while( j<n){
        if( hs.search (s[j]) == false){
            hs.insert(s[j])  j=j+1
            ans = max(ans, hs.size())
        }
        else{
            hs.remove(s[i])  i++
    }   }
    return ans;
}
```

Idea4: Binary Search ⇒ TC: $\log N * N$ ⇒ $O(N\log N)$ SC: $O(N)$

a) Target : length longest Substring

b) Search Space : low high

1 N

c) Discard :

↓

Ex1: String with 10 characters lo hi mi

1 10 5

// Is it Possible to have substring of len = 5 will distinct char

TODO = O(N)

{ //check function = O(N) }

TODO = O(N)

. . . 3 4 5

T T T T T

// Is it Possible to have substring of len = 8 will distinct char

8 9 10 . . .

F F F

// Patten: T T T T . . T F F F F F

2Q) Given 2 strings of equal length, check if they are permutations of each other

Input : String contains only english alphabet ≈ 26

Eg 1:

S_1	S_2	
cat	tac	yes
mata	tamt	No
anat	tana	Yes

Idea: freq of characters should be same in both S_1 & S_2

$S_1 \xrightarrow{\text{store}} hm_1 : O(N)$

$S_2 \xrightarrow{\text{store}} hm_2 : O(N)$

comp hm1 → hm2 : $O(26) \Rightarrow O(1)$

TC: $O(N)$ SC: $O(26)$?

↳ At max we will have 26 keys

Idea 2: Sort strings & compare

↳ TC: $O(N\log N + N)$

↳ TC: $O(N + N)$

1) Sort a string in $O(N)$ time

2) Countsort: $O(N+26)$ time

308) Count no: of substrings of B are permutations of A

Ex1:

	0	1	2	3	4	5	6	7	8
B :	a	b	c	b	a	e	a	b	c

N

	0	1	2
A :	a	b	c

k

0-2 : Yes
1-3 : No
2-4 : Yes
3-5 : No
4-6 : No
5-7 : No
6-8 : Yes

#ans = 3

Idea: For all substrings of len k check if its permutation to A

↳ TC: O(N-K+1) * (K)

#no: of substrings of len = k

comparing 2 strings anagrams are not

Idea2: Sliding window + hashmap

TODO: TC: (N-K+1) * (26)
↳ TC: O(N) * 26
TC: O(N)
SC: O(26) → O(1)

	0	1	2	3
A :	d	a	b	a

→ hm1 {a:2, b:1, d:1}

	0	1	2	3	4	5	6	7
B :	a	a	b	d	a	b	a	d

0-3 : hm2 {a:2, b:1, d:1} == hm1 { } ✓ c=c+1

1-4 : hm2 {a:2, b:1, d:1} == hm1 { } ✓ c=c+1

2-5 : hm2 {a:1, b:2, d:1} $\not=$ hm1

3-6 : hm3 {a:2, b:1, d:1} == hm1 { } c=c+1 ✓

4-7 : hm3 {a:2, b:1, d:1} == hm1 { } c=c+1 ✓

Tree map /

→ Treemap<k, v> is enactly same as hashmap<k, v>

insert( )
search( )
delete( )
update( )
} O(logN)

N is number of keys

O(1) {
insert( )
search( )
delete( )
update( )
}

Iterate in TreeMap :

O(N)

Iterate in hashmap :

TC: O(N)

Implemention : BBST :

(Balanced Binary search Tree) /

[Red black Tree]

Implementati: HashTable

few hashing techiques

In TreeMap : data is Sorted based on keys

Treeset: Everything is same as Treemap, only thing we can only insert keys, data will be sorted based on keys

Ex:

key country	val pop
India	100
america	40
China	140
japan	200
pak	60

(data sort keys)

Treemap

america	40
China	140
India	100
japan	200
pak	60

(no order of keys)

Hashmap

japan	200
america	40
India	100
China	140
pak	60

Given ar[5] = { 6 8 2 14 20 } sort it?

Treeset Ts : →

Iterate in Treeset : 2 6 8 14 20 ✓

↳ O(N)

2
6
8
14
20

Insert all N ar() in Treeset → TC : NlogN + N ⇒ O(NlogN)   SC : O(N)

floor(9) = 8, ceil(9) = 14

Note : If data repeats it will fail

// Operations in Treemap / TreeSet

logN
- insert( )
- search( )
- delete( )
- update( )

floor(k) : It will return greatest key <= k

ceil(k) : It will return smalle key >= k

↳ If floor n ceil doesn't exist for k it will return NUll ?

// Initial N countries & their current population is given

→ Q queries : 1) Increase population of a given Country by X

2) Get top k min populated countries

Initial data :

Country	population
India	50
China	70
US	30
pak	20
Japan	15 45
korea	25

Type Query :

2 : 3 : Japan / pak / Korea : O(K)

1 : Japan : 30 →

2 : 3 : pak / Korea / US

1 : pak : 30

1) order based on population

2) order based on countryname

combined is a single key

TreeSet < pair < population, country > >

~~{15 Japan}~~

~~{20 pak}~~

{25 korea}

{30 US}

{45 Japan}

{50 India}

{50, pakistan}

{70 China}

key Country	value population
India	50
China	70
US	30
pak	50
Japan	45
korea	25

For Query Type: 2

Iterate on first k keys from Treeset ⇒ $O(k)$

For Query Type: 1 ⇒ $O(\log N)$

: Step 1: get populatn of county : from hashmap : $O(1)$

: Step 2: delete <pop, county> from Treeset : $O(\log N)$

: Step 3: update pop in of county in hashmap : $O(1)$

: Step 4: Insert new <pop, county> into Treeset : $O(\log N)$

SC: $O(\underbrace{N}_{\text{Treeset}} + \underbrace{N}_{\text{Hashmap}})$

String = a a b d a b a d

Pattern = a a d b

$S = \boxed{a \ a \ a \ a} \ b \ b \ d \ d$

$^4C_2 * {}^2C_1 * {}^2C_1$

P = a a b d