String.

Q88. Reverse words in a string.

→ ignore all the spaces and save each and every word inside an array of strings.

→ note " " acts as a word break.

→ then add all the elements in reverse order inside the array.

TC ⇒ $O(N) + O(W)$ SC → $O(W)$

Q89. Longest palindrome in a string

→ find longest palindromic substring in a string

→ Approach 1 Brute force TC ⇒ $O(N^3)$ SC ⇒ $O(1)$

→ Approach 2 Dynamic Program TC ⇒ $O(N^2)$ SC ⇒ $O(N^2)$

Approach 3 Best

consider each and every ~~string~~ character except and ones as center and find the palindromic substring

TC ⇒ $O(N^2)$ SC ⇒ $O(1)$

190. Roman number to integer and viceversa

Roman number to integer.

→ add all the Roman characters and value in a map and check whenever any character is less than character ahead of it.

→ if it is then that character tends to be subtracted

eg. IV

m[I] < m[V] thus we decrement m[I] from answer.

TC ⇒ $O(N)$ SC ⇒ $O(7)$ constant

Integer to Roman numeral

→ save all the numbers and their roman representation in a map in decreasing order

→ iterate over the map and also check whether ~~our~~ number is currently lesser that vector's number, if it is greater then we add roman rep in ~~string~~ and decrement its value.

TC ⇒ $O(N * 3)$ SC ⇒ $O(13)$

This is amortized and avised of while loop.

this can be ~~O(100~~

Date Page

Q 91. Implement ATOI / STR STR.

atoi() : Takes string as input and returns integer type value of string

→ handle errors
→ if string is not non numeric, it returns 0.
following ~~must~~ attribute is invalid.
→ discard all leading white spaces
→ sign of number
→ overflow
TC ⇒ O(N)

strstr() → checks whether a substring is substring of another one or not.
→ if it doesn't find, it returns '0'.
→ returns pointer pointing at the first occurence

Q92. Longest common prefix.
→ longest common prefix string ~~amo~~ amongst an array of strings.
→ use of size operator.
→ check prefix each and every character
TC ⇒ O(N) SC ⇒ O(1)

Q93. Rabin karp.
→ given a text and pattern, return all occurences of all patterns in the word.
→ sliding window
TC : O(N²) SC : O(K)

Date : Page :

Q94. Prefix function / z function

→ finds occurence of a pattern in a text in linear time.

TC: O(m+n)

z array: same length as string

→ stores length of longest substring from S[i] which is prefix of s[0.. n-1]

eg.

a	a	b	a	a	c	d
x	1	0	2	1	0	0

→ concatenate Pattern to text from front and with "$" sign, our string looks like P$T.

→ now check in linear time whether length is equal to pattern length. TC: O(m+n)
SC: O(m+n)

Q95 KMP algorithm (Knuth Morris Pratt)

→ whether any pattern exist in string

→ move pointer untill there is a match

→ if there's mismatch, move i at the next index of beginning of checker and j at begin of pattern

→ drawback: ~~dso~~ backtracking in naive algo. worst case → O(N*L)

KMP.

→ avoid comparision in case prefix has some elements in suffix.

→ create a array table

→ don't move back just move back as per table.

i = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ 12 13
j = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ 12 13

eg.

a b a b c a b a a b a b a b d

	a	b	a	b	d
p:	0	0	1	2	0

if there's a mismatch, go to position a[j] since its repeating.

Q 96 - Minimum characters needed to be added to string in order to make it palindrome

Approach 1 Naive Approach:

→ delete last characters untill you get a palindrome string

→ Ans = ~~minimum~~ characters deleted.

TC ⇒ $O(n^2)$ SC → $O(1)$

Approach 2 LPS array KMP algorithm

→ Now we will create 'LPS array' with the use of concatenation

→ given strings.

p ⇒ S + "$" + reverse(S)

create LPS array

S = AACECAAAA.

A	A	C	E	C	A	A	A	A	$	A	A	A	A	C	E	C	A	A
0	1	0	0	0	1	2	2	2	X	1	2	2	2	3	4	5	6	7

→ ans = 7

Ans = len(S) − 7 = 9 − 7 = 2

TC = O(N) SC = O(2N)

Date : Page :

Q97 count for anagrams

→ ~~dict~~ Approach 1. Brute force

TC ⇒ $O(N^2)$ SC ⇒ $O(N * M)$

Approach 2 sliding window

→ maintain ~~dict~~ map for keeping track for the occurence of char in any frame

→ moment when all second values are 0, we get an anagram.

TC ⇒ $O(N)$ for first frame and $O(1)$ for ~~later~~ frames

SC ⇒ ~~of~~ $O(N)$

Q98 count and say

→ given a number, count its digit and add count as well as digit to new number

→ return new number

→ do operation for n times TC ⇒ $O(n)$ SC → $O(1)$

Q99 compare version numbers

→ identify unique state of s/w product.

→ we can't compare them directly,

→ we could convert each and every number in between dots and check for comparision

→ in case both are equal, we go to next section.

TC ⇒ $O(n)$ SC = $O(1)$