Strings

27th April, 2022

AGENDA

- What are strings ?
- Operations on strings
  (Flip the case,
  Reverse string,
  Sort string)
- Prefix and palindromic strings
- Some interesting problems

Strings

→ array of characters
↳ sequence / list / combination

what is a character ?

Symbol.

a-z
A-Z
} → alphabets

0-9 } → numeric characters

% . + - & } → special characters.

* Characters are the building blocks of a string.

A
B C → 101001

ASCII value.

* A → 65 → 1000001
B → 66
C → 67
D → .
.
.
Z → 90

+25

* a → 97
b → 98
c → 99
.
.
.
z → 122

+25

* '0' → 48
1 → 49
2 → 50 . . . . '9' → 57

'0' and 0 are different.

'9' and 9 are different.

			7 6 5 4 3 2 1 0	
Capital	A	→	0 1 0 0 0 0 0 1	(65)
Small	A	→	0 1 1 0 0 0 0 1	

Capital	E	→	1 0	+32
Small	E	→	1 1	

F ⟶ f

f ⟶ F

Toggle 5th bit from LSB.

↓ (0-indexed)

Q. Given a string, flip the case of each character.

Inp → AruN
Out → aRUn

```
for i in range(len(A)):
    if ord(A[i]) >= 97 && ord(A[i]) <= 122:
        A[i] = chr(ord(A[i]) - 32)
    else if
        A[i] = chr(ord(A[i]) + 32)
return A.
```

ord → Gives you the ascii value.

ord('a') → 97
ord('z') → 90

chr → Gives you the character for an ascii code.

chr(97) → 'a'
chr(90) → 'z'

Strings are immutable.

A = "hello"
A[0] = 'c' ✗

You cannot change part of a string.

string = ""
string += 'a' ✓

"abc" ✗
↖ d
"abcd"

2 approaches.

1. Create a new string,
   st = ""
   Keep on appending each character.

TC: O(N)
H.W

→ (2.) Convert the string to list, perform the above operation, rejoin
   ↳ convert the list back to string.

TC: O(N)
H.W

st = "hello"
l = list(st)
list("hello")
// l = ['h', 'e', 'l', 'l', 'o']
↖ List of characters.

l[0] = 'c'

// l = ['c', 'e', 'l', 'l', 'o']

ans = "".join(l)

//ans = "cello"

" ".join(l)

↓ delimiter →

",".join(l) → Join the string with comma.

list("hello")

→ "h", "e", "l", "l", "o"

list("He is good.")

→ ["h", "e", "_", "i", "s" ....]

s1 = "He is good"

s1.split('_')

→ l = ["He", "is", "good"]

"_".join(l)

→ "He_is_good"

",".join(l)

→ "He,is,good"

st = "He,is good"

st.split(",") ← ["He", "is good"]

st.split(" ") ← ["He,is", "good"]

Q. Given a string, (of lowercase alphabets only) sort the string.

1) Convert string to list.
2) Sort the list ← O(N log N)

```
l = ['a', 'c', 'b']
l.sort()
  ↳ ['a', 'b', 'c']
```

3) Re-join.

O(N log N).

Perform this sort O(N) time!

* Compare and swap. → O(N log N) or O(N²)

inp = "ābbcāābācdāb"

out → aaaaabbbbccd

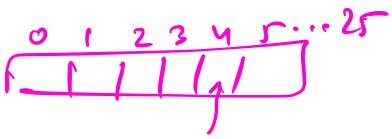
* Count occurrence of each character.

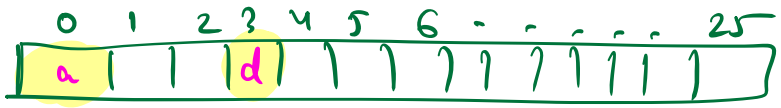
Store the count of each character.

```
l = [0]*26          # l[i] ← will denote count of ith character.

for c in A:
    l[ord(c)-97] += 1
```

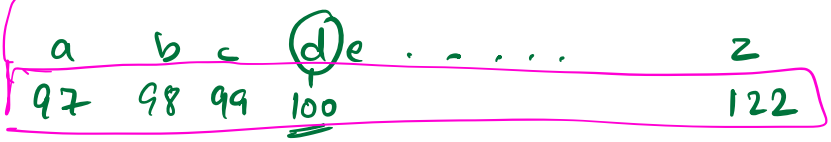
O(N)





"hello"

-97



100-97 = 3

size=26

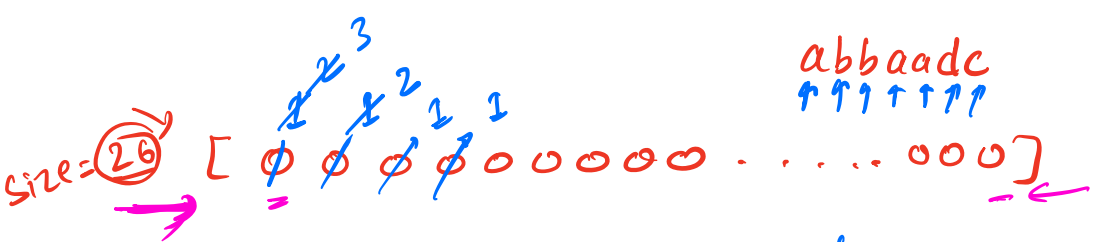
a b c d

l = [3, 2, 0, 2, 0, 0, 0, ...]

```
ans = ""
for i in range(26):
    for j in range(l[i]):
        ans += chr(i+97)
return ans
```

26 ← for i

O(N) ← for j



ord('a') - 97 = 0

l[0] += 1

ord('b') - 97 = 1

l[1] += 1

→ { l = [3 2 1 1 0 0 0 0 0 0 0 . . . . .] }

-26 0

→ for i in range(26):
    for j in range(l[i]):
        ans = ans + chr(i+97)

//i=0

O(N) + O(26*N) ≃ O(N)

O(26* N)
= O(N)

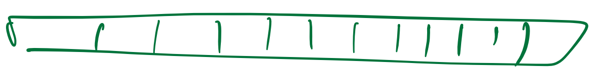
(**) Count SORT

↓ O(N)

[5, 10, 10, 3, 5, 8]

[1, 10^9, 10^8, 2, 2×10^5]

↓ SC = O(N)



[0, 0, 0, 1, 3, 7, 8, 7, 10]

↓ Range of nos is fixed
like 0-9.
0-20.

↓

Use count sort!

Q. Given a string, reverse it.

`s[::-1]` ←

or convert to list, reverse it.

Q. Given, a string. reverse a part of the string. (substring)

0 1 2 3 4 5 6 7 8 9 10 11

whereso syious

whysoserious

```
def revsub(A, i, j):
    A[0:i] +
    A[j:i-1:-1] +
    A[j+1:]
```

** Two pointers

for (i) in A: ← pointer

Convert string to list.

(**)

whereso syious

i j

```
while(i<j):
    a[i], a[j] = a[j], a[i]   // Swap two characters
    i++
    j--
```

(***) Convert list back to string. (join)

Break till 10:25

Q. Reverse words in a sentence.

Inp: "He is well"

Out: "well is He"

split on space.
["He", "is", "well"]
O(N)
reverse.
O(N)
→ rejoin on space!
O(N)

HW

Q.

Inp = "He is well"

Out = "eH si llew"

HW

A → a → 97

65 → 1000001

B → b. +32

K → k +32

```
  1000001
+ 100000
-------
  1100001
```

32

```
  1000001
```

H → h

90
1011111
97

Prefix string

st[0:i]

0
Srinivas.

- S
- Sr
- Sri
- Srin
- Srini
- Sriniv
- Sriniva
- Srinivas ✓

For a string of length N. how many prefix strings?

0 → n-1

(n) prefix strings

Q. Given 2 strings, calculate the length of longest common prefix string.

ans = 5

HW

i → 0 to N.
cnt = 0
if A[i] == B[i]

* Palindromic strings

Palindrome → a string which is same as its reverse.

* Dad
  Madam
  Mom
  Nitin
  Malayalam
  Lol
  Sms

Q. Check if a given string is palindrome or not.

↳ Reverse and check if equal.

↳ Two pointers

```
i=0, j=N-1
while(i<j)
    if (A[i] != A[j])
        return False
    i++
    j--
return True
```

TC: O(N)

Q. Given a string, find the length of longest palindromic substring.

★★★ (Interview problem)

S = "abacab"

ans = 5. ✓

S = "abaeabf"

ans = 5. ✓

Brute force

1. Get all substrings. — $O(N^2)$
2. Check if each substring is palindrome. — $O(N)$
   if yes, keep its count
   ans = max(ans, len(__))

"a" → Yes

"aa" → Yes

No. of substrings → $\frac{N(N+1)}{2}$

H.W

"stringisgood"

(i) 0 to N-1

(j) i to N-1

Brute force = $O(N^3)$

Pseudo-code

```
ans = --
N ← for i in range(0, len(A)):
    N ← for j in range(i, len(A)):
            # we have a substring.   ← A = A[::-1]
N ←         if isPalindrome(A, i, j):
                ans = max(ans, j-i+1)
```

O(N³)

"abkcdefjgh"

i j

j - i + 1

Optimised approach

1st observation

baacaab

odd: alphabet will be centre

baab

even: an imaginary line will be centre.

2nd observation

abacdefa

cd ✗

acde ✗

bacdef ✗ } Useless.

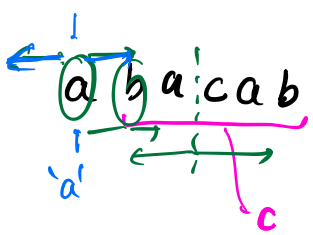
3rd observation

aaaaaaaaa

Yes.

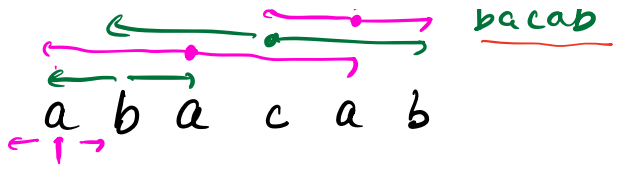
Palindrome yes

Lot of time waste.



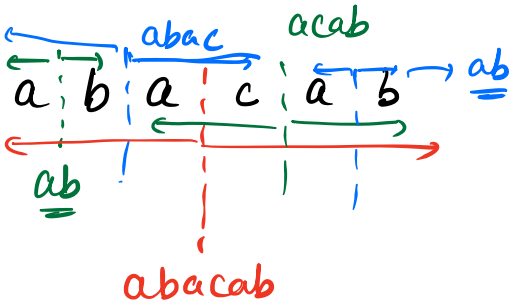
b

cab



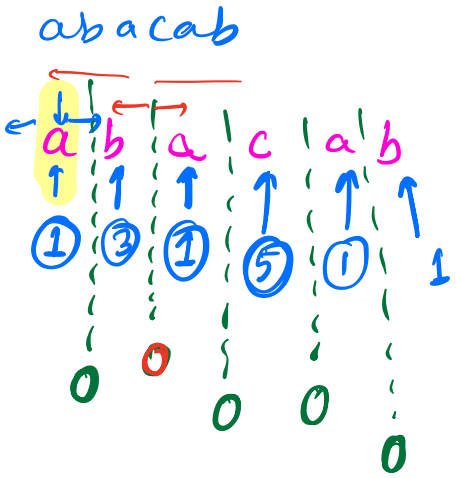
a aba abaca

odd length ⟵ N centres.
(because Alpha bets)



N-1 imaginary lines

Total no. of centres = 2N-1



TC:

How many centres? 2N-1 centres.

For each centre, if I expand to get the longest palindrome from that centre?

O(N)

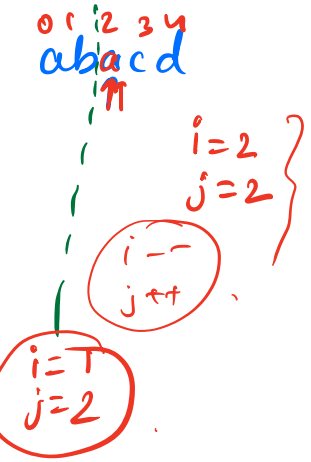
** TC: O(N²)

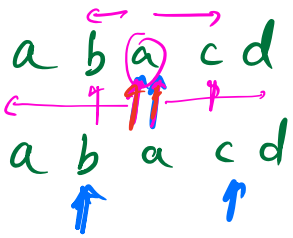
Pseudo-code:

```
def expand(A, i, j):

    ans = 0
    while (i >= 0 and j <= len(A)-1
           and A[i] == A[j])
        ans = j-i+1
        i--
        j++

    return ans
```





0 1 n-1
a b a c d a a d e f x x c a a

a b a c d

ans = expand(A, 0, 0)
expand(A, 0, 1)
expand(A, 1, 1)
expand(A, 1, 2)
expand(A, 2, 2)

↓

2n-1 times

length of the palindromic substring if centre is (0,0)

(0,0)
(0,1)
(1,1)
(1,2)
(2,2)
(2,3)
(3,3)
(3,4)

```
def caller(A):
    ans = 0
    for i in range(len(A)):          # i : 0 .. n-1
        ans = max(ans, expand(A, i, i))
        ans = max(ans, expand(A, i, i+1))
    return ans.
```

TC: $O(N^2)$

(**) Question can also be solved in O(N) time.

Manacher's algorithm

Advanced DSA.

Doubt session

a.b = b.a

m = no. of words

n = no. of characters in the whole string.

k = no. of characters in each word

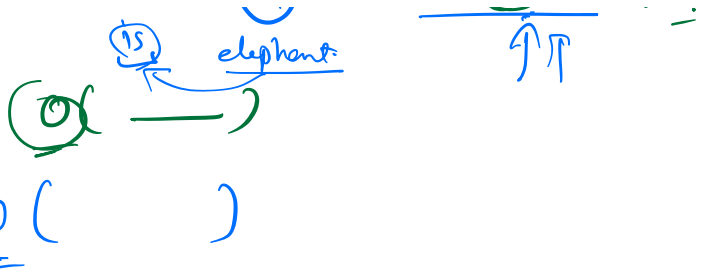
O(n)

O(m * k)

k * m

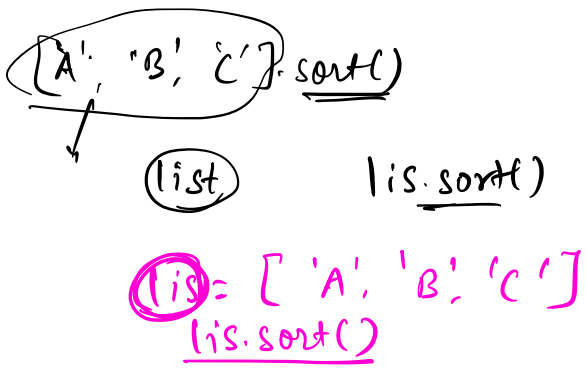
n

k * m = n (max no. of characters in any word)



TC: $O(N^2)$ → TC: $O(N)$

SC: $O(1)$ SC: $O(N)$



2 pointer