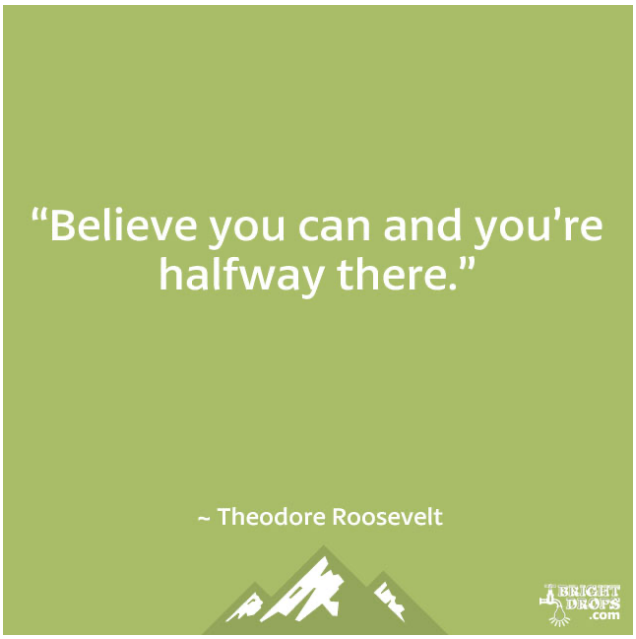
STRING PATTERN MATCHING



Today's content

01. Boring substring

02. Pattern searching (Rabin karp Algo)

→ Probability of collissions

Q. Given a string s (char[]), check whether it is possible to rearrange the characters such that there is no boring substring in S.

Boring substring → length = 2 && consecutive alphabetically

eg:- ab, cd, ed, zy, dc, bc

eg: s = "abc" → false

s = "abcd" → true → bdac or cadb

s = aabccb → false

s = "aabccdb" → True ⇒ ccaadbb

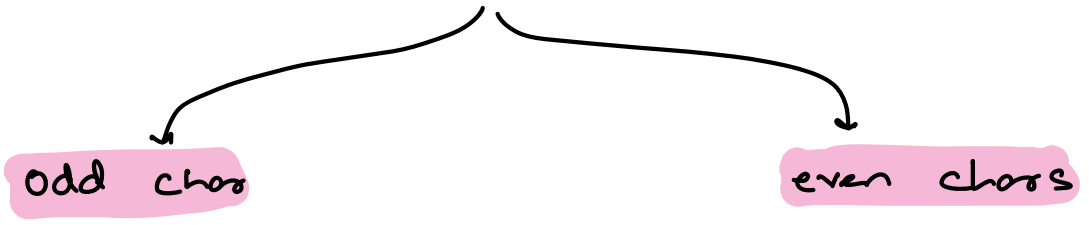
Total no. of permutations for length n = n!

Brute force → Generate each and every permutation of string length n & then check if it is forming any boring substring or not.

$$TC = O(n! * n)$$

All possible alphabets - { a b c d e f .... z }

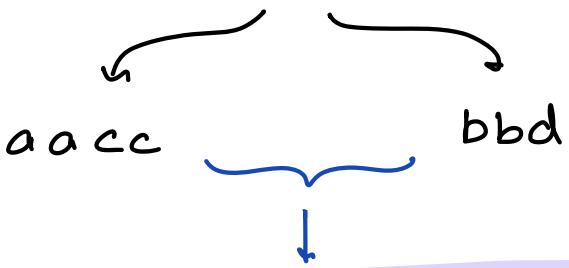
97 98 99 100 101 102 122



{ a c e g i ... w y } { b d f h ... x z }

Obs → Splitting the string in odd-even can give two sets which will never form boring substring

s = "aabccdb"



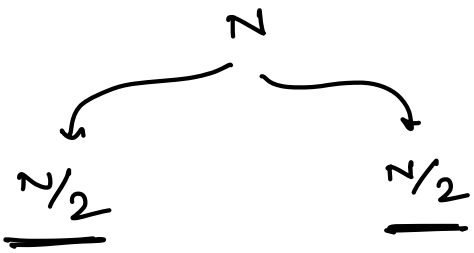
good concatenation point

→ if you can find an alphabet from LHS & an alphabet from RHS which when concatened don't form a boring substring

a c c a d b b → True

* For concatenation

01. Check every character from LHS & every character from RHS



TC = O(N2)

02 Use hashMap

→ Only check for unique character

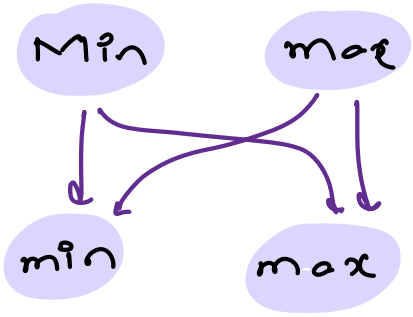
LHS = { 13 ele } RHS = { 13 el }

13 unique 13 unique

TC = O(13 * 13) = O(1)

03 Odd set → Min max

Even set → min max



s = " c c e b d f "

c c e b d f

m in0 = c

max0 = e

min e = b

max e = f

Min- Min ce cb df ✗

Min - max ce cf db → True

Max - min cc eb df → True

Max - max c cef d b x

Q String s = "acefdd"

Min odd = a

max odd = e

min even = d

max even = f

TC = O(n)

SC = O(1)

minodd x mineven = _ ad _ → True

minodd x max even

max odd x min even

max odd x max even

String s = " _ ---- " ;

minO = ∞
maxO = -∞
mine = ∞
maxe = -∞

```
for ( i = 0 ; i < n ; i++ ) {
    char ch = s.charAt(i);
    if ( ch % 2 == 0 ) {
        mine = Math.min (mine , ch);
        maxe = Math.max (maxe, ch);
    }
    else {
        minO = Math.min (minO, ch);
        maxO = Math.max (maxO, ch);
    }
}
```

if (minodd − mineven) > 1 return true

if (minodd − maxeven) > 1 return true

if (maxodd − mineven) > 1 "

if (maxodd − maxeven) > 1 "

→ abs value

Pattern matching

02 Given a large text (string A of length N) & small pattern (string B of length M). Find the count of times B is present as a substring in A.

Note :- M < N && all lowercase characters

A = "abcbcabca" Ans = 2

B = "bca"

A = "ababcabaac" Ans = 3

B = "aba"

A = "abcababac" Ans = 2

B = "aba"

Brute force → For all substring of length M in A, check if it is equal to B or not.

Sliding window X

No. of substrings of length M = n-m+1

arr[n] = 

Subarr = m size

A = "abcabaabac"

B = "aba"

→ sliding window when comparing strings is of no use.

TC = O(n-m+1) * m → O(n·m)

Problem → Comparing two strings takes linear time

Solve → Two integer takes O(1) time

Robin Karp Algorithm

B = "acd"

h("acd") = 1+3+4 = 8

a → 1
b → 2
c → 3
⋮
z → 26

A = "bacdef"

B = "acd" → 1+3+4 = 8

First window = "bac" = 2+1+3 = 6

if $(h(s_1) \neq h(s_2)) \rightarrow$ implies $s_1 \neq s_2$

Second window = "acd" = $1+3+4 = 8$

if $(h(s_1) == h(s_2)) \rightarrow$ imply $s_1 = s_2$ ✗

$h(acd) = h(adc)$
$= h(cad)$
$= h(cda)$
$\vdots$
$= 8$

Multiple inputs are producing the same hash value

Collission

Avoid Collission → Weightage to the order

$326 = 3*10^2 + 2*10^1 + 6*10^0$
$263 = 2*10^2 + 6*10^1 + 3*10^0$

Order has some weightage

$h("acd") = a*10^2 + c*10^1 + d*10^0$
$= 1*10^2 + 3*10^1 + 4*10^0$
$\Rightarrow 134$

$h("acd") = a * p^2 + c * p^1 + d * p^0$

$p = 1 \rightarrow$ collission

$p = 2 \rightarrow$ ×

Eg:- $S_1 =$ "aa" $= 1 * 2^1 + 1 * 2^0 \Rightarrow 3$

$S_2 =$ "c" $= 3 * 2^0 = 3$

Usually $p = 29$ → prime no. are always preferred in the hash fn

↓

Min collission

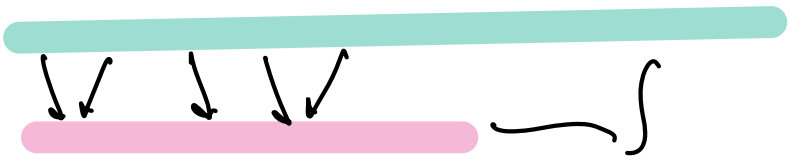
Limitation

h("yogeshyadav") → impossible to store it in an integer or long (Integer overflow)

To make sure, we are always in range → Mod

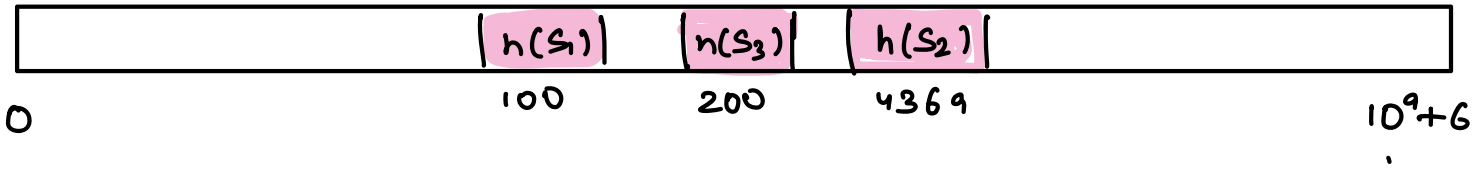
Usually $m = 10^9 + 7$

large prime no.



Pigeon hole → Collissions

Probability of collission



String	h(s)	Probability of collission
S_1	$h(S_1) = 100$	0
S_2	$h(S_2) = 4369$	$\frac{1}{M}$ or $\frac{1}{10^9+6}$
S_3	$h(S_3) = 200$	$\frac{2}{M}$
S_4	$h(S_4)$	$3/M$
⋮	⋮	⋮
S_n	$h(S_n)$	$\frac{(n-1)}{m}$
S_{n+1}	$h(S_{n+1})$	$\frac{n}{m}$

$m = 10^9+7 \rightarrow$ large

Probability of collision $= \frac{n}{m} = \frac{10^5}{10^9} \simeq 0.0001$

neglect the

collission prob

Hash function

$$h(s) = \sum_{i=0}^{n} s[i] * p^{n-1-i} \% M$$

$M = 10^9 + 7$

$p = 29$

Rolling hash fn

B = "acd"

A = "bacd..."

$h(B) = h(\text{"acd"}) = 1 * 29^2 + 3 * 29 + 4 * 1$

$= 932$ } → TC = O(M)

$h(\text{"bac"}) = 2 * 29^2 + 1 * 29 + 3 * 1$

$= 1714$

if $(h(s_1) \neq h(s_2)) \longrightarrow s_1 \neq s_2$

$$h(\text{"bac"}) = 2*29^2 + 1*29 + 3*1$$

Slide window

$$-\left[2*29^2 + 1*29^1 + 3*29^0 - 2*29^2\right]*29 + 4$$

- $2*29^2$ → Remove first
- $*29$ → update place value of other chars
- $+4$ → add next char

$$\left[(2*p^2) + (1*p^1) + (3*p^0) - (2*p^2)\right]*p + 4*p^0$$

if $(h(s_1) == h(s_2))$ → 99.99 % of time $s_1 = s_2$

To get 100% accurate result

if $(h(s_1) == h(s_2))$ → compare both strings char by char

Steps

01. Calculate the hash value → string B → TC = O(m)

02. Calculate the hash value for the first window → TC = O(m)

03. Sliding window → TC = O(n-m+1)

Overall TC = O(n)

SC = O(1)

Code → {TODO}