Pattern - Searching - 1

Question: 2 strings

t: text , s: pattern

t = "hello_scaler" , s = "scaler"

→ Does the pattern 's' exist in text?

→ Count no. of occurences of 's' in text.

< Pattern Matching >

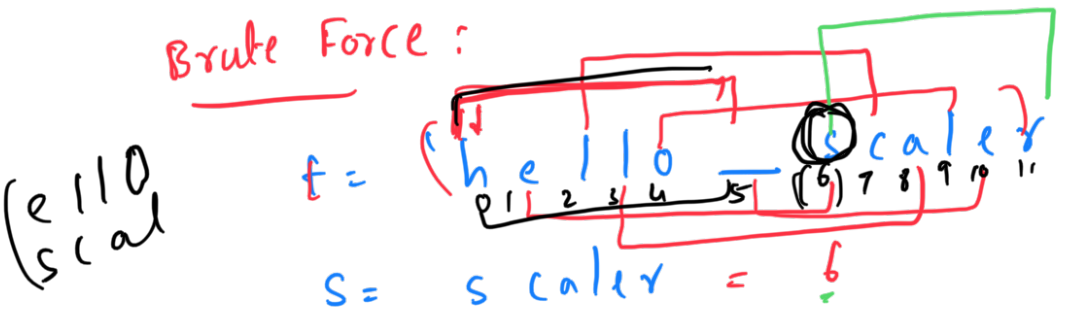
Text:

1) Huge web pages
2) Gmail Inboxes
3) Google Docs
4) Github
5) Plagarism check

} t: $O(10^5 - 10^5)$

hello_ (|s|)(t-s)
scaler

Brute Force:



t = hello_scaler

s = scaler = 6

→ |t| = 12
→ |s| = 6
t-s

ello
scal

: (t-s) x s

T: O(t-s)xs
trs

```
search ( string s, string t) {
    int count = 0;
    for( i = 0; i <= t.length() - s.length(); i++) {
        if( t.substr ( i, s.length) == s) {
            count++;
        }
    }
    return count;
}
```

s1 : a b c d e
s2 : a b c d g

int

O(s)

n(t-s) x s => O(trs)

T.C :

S.C : $O(1)$

a b c d
g h i j

Rabin-Karp Algorithm

→ Compare numbers or strings()

$O(1)$ $O(n)$ $< > \leq \geq$.

→ Map a string to a number (Integer)

t = a a b c a d a

f : Maps String to a Number

s = ada

f(ada) = x

f(aab) = y_1
f(abc) = y_2
f(bca) = y_3
f(cad) = y_4
f(ada) = y_5

(x)

We have found the pattern

s_1 s_2

Function Requirements :

1) If 2 strings are not equal, even $f(s_1) \neq f(s_2)$

2) If 2 strings are equal
s_1 = aba, s_2 = aba
$f(s_1)$ = $f(s_2)$

Define this Function

Option 1 : ASCII value of 1st character

f(aab) : 97
abc : 97
. 98

aab, abc, abb ⇒ 97

: To use all the characters strings.

bca :
aab : 97

Option 2: Sum of ASCII value of all characters

aab : 97 + 97 + 98 = 292
abc : 97 + 98 + 99 = 294
bca : 98 + 99 + 97 = 294
abb : 97 + 98 + 98 = 293

Conclusion: Consider the position of a character.

Option 3: Take position into count

aab : 1×97 + 2×97 + 3×98 = 585
abc : 1×97 + 2×98 + 3×99 = 590
bca : 1×98 + 2×99 + 3×97 = 587
abb : 1×97 + 2×98 + 3×98 = 587

Option:

n = length

$$f(S) = \sum_{i=0}^{n-1} S[i] \times k^{n-i-1}$$

$$f(S) = S[0] \times k^{n-1} + S[1] \times k^{n-2} + S[2] \times k^{n-3} + \ldots + S[n-2] * k + S[n-1]$$

'k' is a number greater than |Σ|

i.e., Lower case characters

$k \geq 26$ $k = 31$

$$f(abcd) = a \times 26^3 + b \times 26^2 + c \times 26 + d$$

$$f(\underbrace{abcd}_{(n)}) = \left(a \times k^3 + b \times k^2 + c \times k^1 + d \times k^0\right) \% M$$

Any Issues ?

int : 10^9
long : 10^{18}

len(string) = 100

ar(k ...)   26

Polynomial Rolling Hash Function

$h(s) = \left(\sum_{i=0}^{n-1} s[i] \times k^{n-i-1} \right) \% M$ → (define)

$H(abc) = (a \times 26^2 + b \times 26 + c) \% (M)$

[10t

0 to M − 1

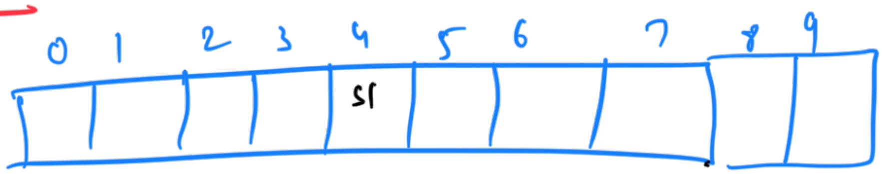
L: $f(x) = x \% 10$

0 − 9

% M

Collisions:

M = 10



$0 \le h(s) \le 9$

s1 : 0

s2 : 1/M ;

s3 : 2/M

s_n : $\frac{n}{M}$

$Pr(collision) = \left(\frac{n-1}{M}\right)$ → Increase

Very Large Number & Prime Number

$M = 10^9 + 7$

$Pr = \frac{n}{10^9 + 7}$ → $= 10^5, 10^6$

$Pr = \frac{10^6}{10^9} = 10^{-3}$

$Pr = 0.001$

→ There will not be any collision

Pattern Matching

t = ababaaaadfgab ,   s = aadf

h(abab), h(baba), ...   h(a..)

O(n(t−s)) ⇒ O(s × t)

O(S) × ( )

Can we make this Better

k = 26

t = a b c d e f g h i j , s = f g h i

h1 = h(abcd) = a × k^3 + b × k^2 + c × k + d

h2 = h(bcde) = b × k^3 + c × k^2 + d × k + e

(b k^2 + c × k + d) × k

b k^3 + c k^2 + d k

O(1)

h2 = (h1 − a k^3) × k + e   (−2) % M

a × k^(|s|−1)

STEPS

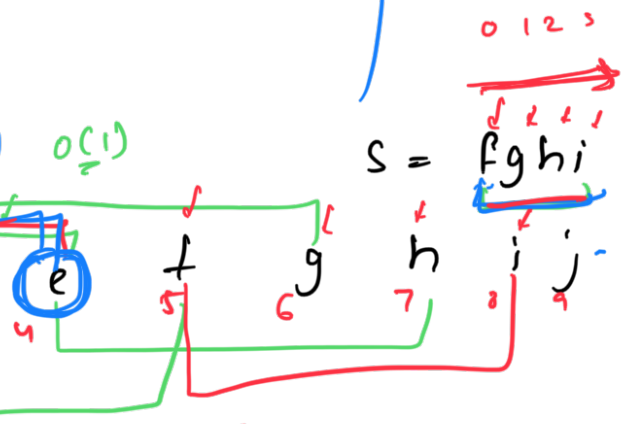
Step 1: h = (h − (s[0] × (k^(|s|−1)) % M + M) % M

Step 2: h = (h * k) % M

Step 3: h = (h + s[4]) % M   O(1)

|s| = 4

O(s) = O(1)



i-1

a b c d
hash_t + = (a k^3 + b k^2 + c k + d)   h(t) = h(s)

count++;

Conservative & Extra safe

```
int rabinKarp( string s, string t) {
    int M = 10^9 + 7;
    int K = 26;
    powers[|s|]   // stores powers of k.
    hash_t = 0;
```

hash_s = h(fghi)
hash_t = h(abcd)

```
int hash_s = 0, hash_t = 0;
for (i=0; i < s.length(); i++) {
    hash_t = ((hash_t + t[i] * power[|s|-i-1]) % M + M) % M;
    hash_s = ((hash_s + s[i] * power[|s|-i-1]) % M + M) % M;
}                                                          O(|s|)
if( hash_s == hash_t && t.substr(0,|s|) == s)
    count++;

for( i = 1; i <= |t|-|s|; i++ ) {
    hash_t = (hash_t - (t[i-1] * pow[|s|-1]) % M + M) % M;   step1 :
    hash_t = (hash_t * k) % M;                               step2 :
    hash_t = (hash_t + t[i+|s|-1]) % M;                      step3 :
    if( hash_t == hash_s && t.substr(i, |s|) == s )          O(s)
        count++;
}
return count;
```

t = a b c d
f = a b c d
O(w)

O(s * f)

T.C : O(|t|) + O(|s|)

Average T.C : O(|t| + |s|) : O(|t| * |s|)

t = "a a a a a a a a a a a a a a a a"

s = "a a"

Worst : O(f * s)

Average : O(f + s)

O(|t| + |s|)

Z - Algorithm =>

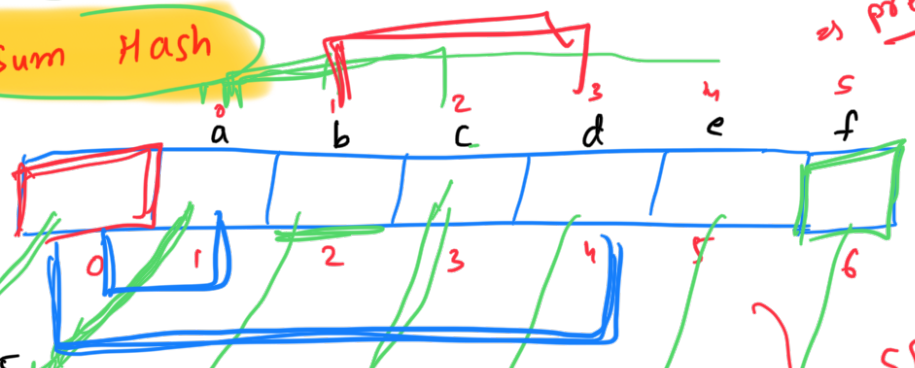
Palindromic Substring Queries

Prefix Sum Array

str = 'abcdef'

S[3:5] ⇒ pre[6] − pre[3]

Prefix Sum Hash



pre =

pre[0] = 0

pre[1] = ak^5

pre[2] = $ak^5 + bk^4$

pre[3] = $ak^5 + bk^4 + ck^3$

pre[4] = $ak^5 + bk^4 + ck^3 + dk^2$

pre[5] = $ak^5 + bk^4 + ck^3 + dk^2 + ek$

pre[6] = $ak^5 + bk^4 + ck^3 + dk^2 + ek + f$ (k^0)

S[1:5] → bcd

pre[4] − pre[1]

pre[j+1] − pre[i]

f[3:5]

pre[6] − pre[3]

pre[j+1] − pre[i]

k^{n-j-1}

s = a b c d e f (0 1 2 3 4 5)

Query = (i, j) = (1, 3)

S[1:3] = bcd : $bk^2 + ck + d$

pre[4] − pre[1] = $bk^4 + ck^3 + dk^2$ = $k^2(bk^2 + ck + d)$

k^2

j = 5 pre[6] − pre[3] = $(dk^2 + ek + f)k^0$

S[2:4] = $\frac{pre[5] - pre[2]}{k^{n-j-1}} = \frac{ck^3 + dk^2 + ek}{k^{n-j-1}} = (ck^2 + dk + e)k^1$

n = 6

j = 5, k^0

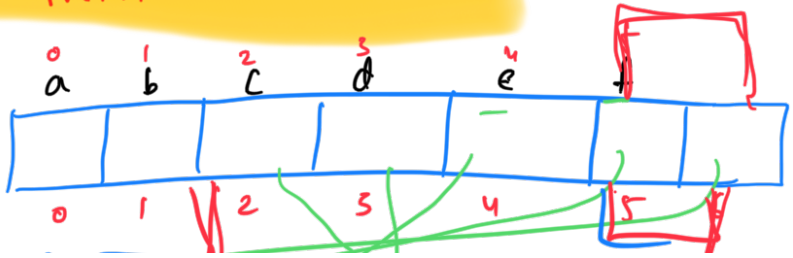
j = 4, k^1

j = 3, k^2

k^{n-j-1}

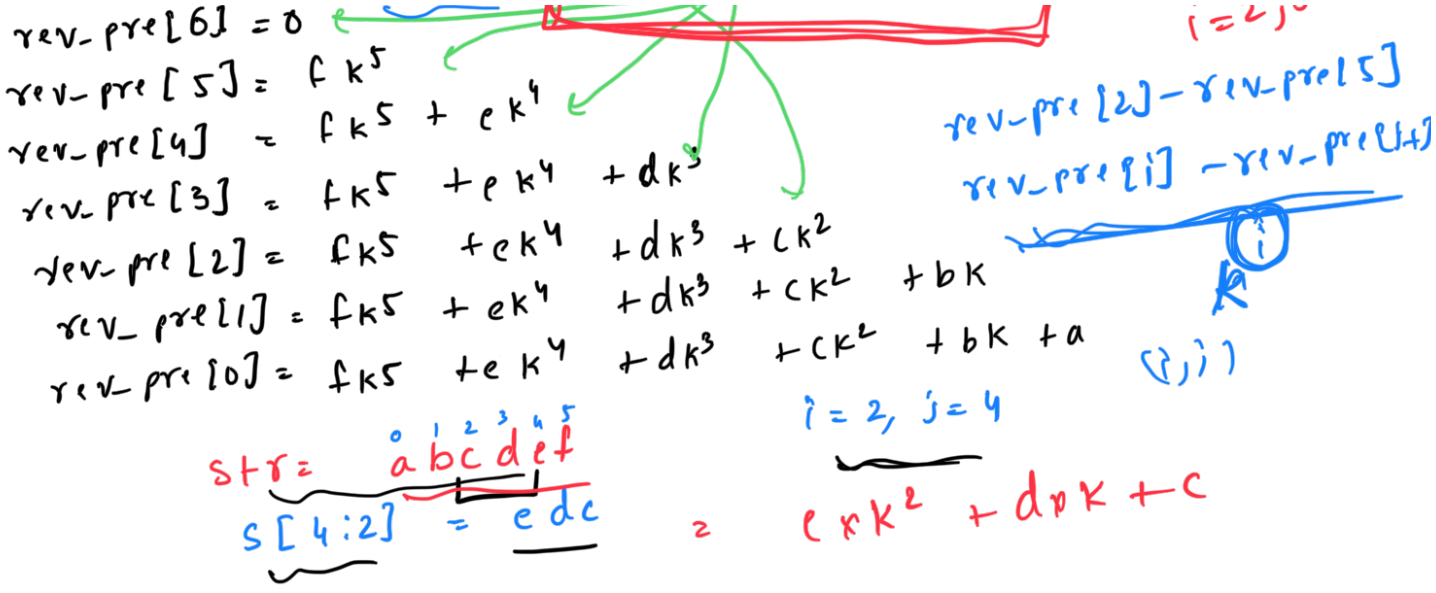
6 − 5 − 1 = 0

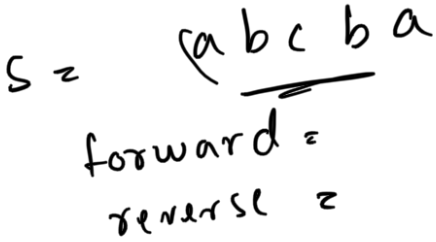
Reverse prefix Sum Hash



rev_pre :

j = 4





Question: check if a string is palindrome ($P_1 == P_2$)

str = a c d f d c a

P_1 P_2 P_1 P_2

$|s| = 7$

T.C: O(n)

forward_hash = $(a*k^6 + c*k^5 + d*k^4 + f*k^3 + d*k^2 + c*k + a) \% M$ (n)

reverse hash = $(a*k^6 + c*k^5 + d*k^4 + f*k^3 + d*k^2 + c*k + a) \% M$ → (n)

if(reverse_hash == forward_hash)

return True;

T.C: $O(\frac{n}{2})$ $O(2n) = O(n)$

q, $n <= 10^5$

① queries of [i, j)

Question:

If substring [i : j] is a palindrome or not.

str = a c d c d e
0 1 2 3 4 5

Queries : q

1) (1, 3) : c d c Yes
2) (2, 5) : No
3) (2, 4) : Yes.

q * n

TC: O(q * N)
10^5 ↙ ↓ 10^5
10^{10} ⇒ T.LE

How can we optimize ?

→ Do some Precomputation

→ O(1)

n = length of whole string

$$\left(\frac{pre[j+1] - pre[i] + M}{k^{n-j-1}}\right) \% M$$

$$(a * b) \% M = (a \% M * b \% M) \% M$$

$$b = \left(\left(\frac{1}{k^{n-j-1}}\right) \% M\right)$$

$$(x^{-1} \% M) = (x^{M-2} \% M)$$ (Fermat's Little Theorem)

$$\left((pre[j+1] - pre[i] + M) \% M * (k^{n-j-1})^{M-2} \% M\right) \% M$$

T.C: O(n) + O(n) + q * O(1)

T.C = O(n + q)

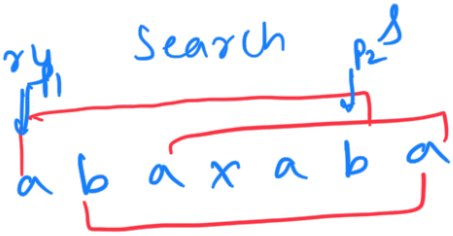
S.C : O(n)

Question: Longest Palindromic Substring

1) Brute Force : $O(n^3)$ $(K \times n-K) \Rightarrow \geq O(n) \Rightarrow O(n^2)$

2) $O(n^2)$

3) Binary Search Rolling Hash Function.



n = 11

mid = 6

Binary search :

Mysterious check function

low = 1, high = 1

mid = (6)

O(n)

Search Space:

low = 1

high = n

```
int ans
while (low <= high) {
    mid = (high + low)/2;
    if (check(mid) == True) {
        ans = mid;
        low = mid+1;
    }
    else {
        high = mid-1;
    }
}
return ans;
}
```

T.C : $n \log n$

S.C : $O(N)$

60% ⇒ 90%

Important Topics

1) Data structures Algorithm ★★★★

(Wednesday) 40%

2) Operating Systems
   Paging, Segmentation
3) Data base Systems
4) OOPS

1-3 hours

5) Project => 100% thorough

HLD & LLD

Coding Round

Interview:

1) Online (250) A.
   1) Aptitude →
      1) Probability
      2) PSC
      3) Statistics
   2) coding → (2-3)
      1) Linked list
      2) Binary Trees
      3) Arrays

CAT quant

2) 60 People
3) 1st Round → 20 people
4) 2nd Round → 13 people
5) 3rd Round → 8 people

1) stalks

TOP

2) 15.5 7

3)