String pattern Matching

Good Evening

Content

01. Permutations of A in B
02. Rabin karp Algo
03. Check if palindrome in a running stream

Q Given two strings A & B, A with length N B with length M. Count all permutations of A present in B as a substring.

Note :- A & B contains lowercase characters

Eg :- A = "xyz" → n

B = "xyzxy" → m

Ans = 3

A = "abac"

B = "abcaacbbccbaca"

Ans = 4

Idea 1 → Get all the substrings of B with length = A & then check if $S_1 == S_2$ or not

↓

Sort them & compare

No. of substrings = $(m-n+1) * n\log n$

$\approx O(mn\log n)$

Idea 2 → Sliding window + freq array

A = a b a c (0 1 2 3)

B = a b c a a c b b c c b a c a

i = 4 - 4

4 5 6

0 1 2 3 ↑

freqA = { 2 1 1 0 0 0 .... }
0 1 2 3 4 5 ...
a b c d e f

if (compare(freqA, freqB))
ans = ans + 1

2
+
freqB = { 2 1 1 }
0 1 2 3 4 5 ...
a b c d e f

Slide window → add the next character
→ drop the first character of previous window

Pseudocode →

01. Create a freqA of string A

```
for (i=0; i<n; i++) {
    freqA[A[i] - 'a']++;
}
```

02. Create freqar of first window of B string

```
for (i=0 → n-1)
    freqB[B[i] - 'a']++;
}
```

```
if (compare(freqA, freqB) == true) ans = ans+1;

for (i=n; i<m; i++){
    freqB[B[i] - 'a']++;   // add next char
    freqB[B[i-n] - 'a']--; // drop last char
    if (compare(freqA, freqB) == true) ans = ans+1;
}
```

$$TC = O(m)$$

$$SC = O(26) = O(1)$$

```
boolean compare(int[] freqA, int[] freqB)
    for (i=0; i<25; i++){
        if (freqA[i] != freqB[i]) return false
    }
    return true;
}
```

Rabin karp Algo

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

Note :- M < N && all lowercase characters

A = "abcbcabca" Ans = 2

B = "bca"

A = "ababcababac" Ans = 3

B = "aba"

A = "abcababac" Ans = 2

B = "aba"

n

m

Brute force Approach → Create all the substrings of A with length = B & iterate & check if substring is equals to B or not

TC = O(n-m+1) * m

Problem → Compare two strings linear time

Compare two integers → O(1) time

B = "acd"

H(B) = 1 + 3 + 4 = 8

a → 1
b → 2
c → 3
d → 4
⋮
z → 26

A = bacdef

H(first) = "bac" = 2 + 1 + 3 = 6

if (H(B) ≠ H(first)) → implies $S_1 \neq S_2$

H(sec) = "acd" = 1 + 3 + 4 = 8

if (H(B) == H(sec)) → implies $S_1 == S_2$

h(acd) = h(adc)
= h(cad)
= h(dca)
⋮
h(bcc)

} 8 → Multiple inputs are providing the same hash value → collision

Avoid this collision

326 = 3 * 10^2 + 2 * 10 + 6
623 = 6 * 10^2 + 2 * 10 + 3

} Order has some weightage

$$h(acd) = a * 10^2 + c * 10 + d$$

$$= 1 * 10^2 + 3 * 10 + 4$$

$$= 134$$

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

$p = 1 \longrightarrow$ collision

$p = 2 \longrightarrow$ Collisions

$S_1 = "aa" = 1 * 2^1 + 1 * 2^0 = 2 + 1 = 3$

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

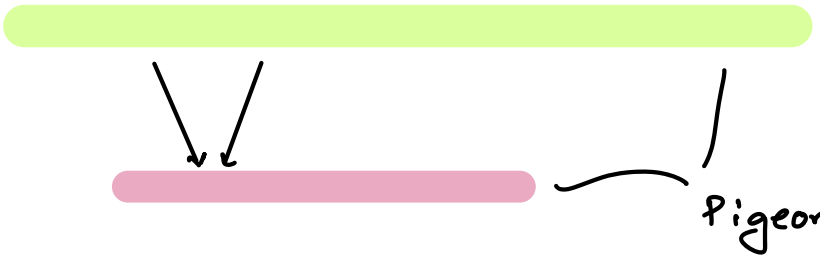
Usually $p = 29$ → prime no. are always preferred in hash function

Limitation

h("jogeshyadav") → impossible to store the hash function value in an integer or long (overflow)

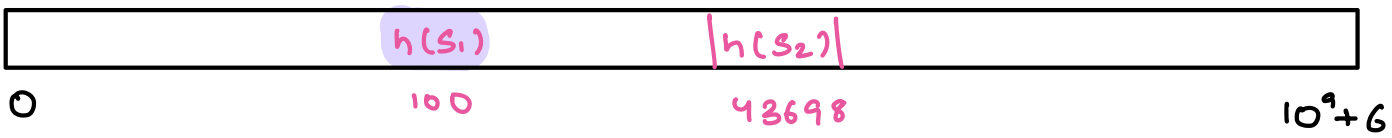
To make sure, we are always in Range = mod

Usually $m = \underbrace{10^9 + 7}$ Large prime no.



Collisions

* Probability of collision



String	Hash(string)	Probability of collision
S_1	$H(S_1)$	0
S_2	$H(S_2)$	$\frac{1}{M}$
S_3	$H(S_3)$	$2/M$
S_4	$H(S_4)$	$3/M$
⋮		
S_n	$H(S_n)$	$n-1/m$
S_{n+1}	$H(S_{n+1})$	n/m

$m = 10^9 + 7$

$n = 10^5$

Probability of collision $= \frac{10^5}{10^9+7} \approx 0.0001$

$\approx 0.1\%$

neglect the collision probability

Hash fn $= \sum_{i=0}^{n} s[i] * p^{n-1-i} \% m$

Rolling hash function

$M = 10^9 + 7$

$p = 29$

B = " acd "

A = " bacd "

Hash(B) = hash(acd) $= 1 * 29^2 + 3 * 29^1 + 4$

$= 841 + 87 + 4 = 932$

$$Hash(bac) = 2 * 29^2 + 1 * 29 + 3$$

$$= 1682 + 29 + 3$$

$$= 1714$$

if $(hash(B) \neq hash(bac) \longrightarrow S_1 \neq S_2$

$$hash(bac) = (2 * 29^2 + 1 * 29 + 3) \% m$$

Slide window

$$hash(acd) = ((2 * 29^2 + 1 * 29 + 3 - 2 * 29^2) * 29 + 4) \% m$$

$$(1 * 29^2 + 3 * 29 + 4) \% m$$

Dropped the first char

update the p value of previous characters

added the last char

$$hash(acd) = 932$$

$hash(B) == hash(acd) \longrightarrow$ 99.9% of time

$S_1 == S_2$

Steps

01. Calculate the hash value for B string → TC: O(M)

02. Calculate the hash value for substring of A equals to length B → TC: O(M)

03. Slide the window for all the substrings → TC = O(N-M)

TC = O(n)

SC = O(1)

Code → {TODO}

10:27 pm → 10:37 pm

Q Given a running stream of characters, check at every step if the current string is palindrome

nayan
racecar
abba

Eg: x y x y y x y x .....
T F T F F F F T

Brute force → For all the inputs, check if string is palindrome or not by travelling on it.

TC: $O(n^2)$ SC: O(1)

str = reverse(str)

hash(str) = hash(reversed(str))

$xy = x * p + y$
$yx = y * p + x$

if these hash(xy) == hash(yx) then I can increase my ans

hashfn

S_1 = a → a → fh = 1

rev(S_1) = a → a → bh = 1

S_1 = ab → a * p + b → fh = 1 * 29 + 2

rev(S_1) = ba → a + b * p → bh = 1 + 29 * 2

S_1 = abc → $a * p^2 + b * p + c$ → fh = (1 * 29 + 2)29 + 3

rev(S_1) = cba → $a + b * p + c * p^2$ → bh = $(1 + 2 * 29) + 3 * 29^2$

fh = 0 bh = 0 , base = 29 , String s = ...
p = 1

```
for (i=0; i<n; i++)
    char ch = s[i] - 'a'
    fh = (fh * base + ch) % m

    bh = bh + ch * p
    p = p * 29

    if (fh == bh) print (true)
    else print false:
}
```

TC = O(n)

SC = O(1)