Design and analysis of Algorithms

(Prof. Parminder Kaur Wadhwa)

String Matching Algorithms

Part A :- includes :-

1. [Simple / Naive Algorithm
2. Brute-Force Algorithm] Same
3. Rabin-Karp Algorithm.

Part B :- includes :-

4. Boyer-Moore Algorithm
5. Knuth-Morris-Pratt Algorithm
6. Application of string matching algorithms.

Design and Analysis of Algorithms

(Prof. Parminder Kaur Wadhwa)

String Matching Algorithms (Part A)

Topics covered :-

1. Simple / Naive Algorithm } are same
2. Brute Force Algorithm }
3. Rabin-Karp Algorithm.

String Matching Algorithms

1. Simple or Naive String Matching
2. Brute Force Pattern Matching

(These are same algorithms)

3. Rabin Karp Algorithm
4. Boyer-Moore Algorithm
5. Knuth-Morris-Pratt (KMP) Algorithm.
6. Applications of string matching algorithms.

The string matching problem:-

It is the problem of detecting the occurrence of a particular substring called pattern, in another string, called the text.

The array P is considered to contain the pattern.

The array T[1...n], is considered to contain the text.

P[1...m]

P → pattern (to be found)

T → text → T[1...n] (from which the pattern is to be found)

n → size of text

m → size of pattern

1. Simple or Naive String Matching

In this, comparisons are done (in left to right order) on the pairs of characters in array P and T.

When first mismatch occurs, the pattern is moved one position forward with respect to text, and comparisons start again at the left end of pattern.

For example :-

T : A B A B A B C C A

P : A B A B C

⇒ T : A B A B A B C C A

P : A B A B C

✓ ✓ ✓ ✓ ✗

Mismatch occured

Now, Shift P towards left by one position and start again

T : A B A B A B C C A

P : A B A B C

→ ✗

First Mismatch

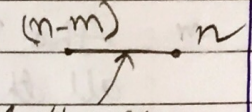
T : A B A B A B C C A

P : A B A B C (match found)

→ → ✓ ✓ ✓ ✓ ✓

Simple_StringMatching (T, p)

$n \leftarrow$ length [T]
$m \leftarrow$ length [p]



Till the m-sized substring in T

for $i \leftarrow 0$ to $n-m$

shift by 1

do if $p[1..m] = T[i+1 \; .. \; i+m]$

then print "Match found with shift" i

Running time $O[(n-m+1)m] \approx O(nm)$

If $m = \frac{n}{2}$ (i.e. if substring is half the size of text T)

then $\Rightarrow O(n^2)$

2. Brute-Force Pattern Matching

The Brute-Force pattern matching, tests all the possible placements of P relative to T.

```
BruteForceMatch (T, P)

for i ← 0 to n−m    (for each candidate index in T)
{
    do
        j ← 0                          (compare characters)
        while ( j < m and T[i+j] = P[j] ) do
        {
            j ← j+1                    // shift by one
        }
    if j = m then
        return i                       // substring found at i
    else
        return "There is no substring of T matching P".
}
```

This algorithm consists of two nested loops, with the outer loop indexing through all possible starting indices of the pattern in the text, and the inner loop indexing through each character of the pattern, comparing it to its potentially corresponding character in the text.

In this algorithm, for each candidate index in T, we can perform m character comparisons to discover that P does not match T at the current index.

The outer-for-loop is executed $n-m+1$ times, and the inner loop is executed at most m times.

Thus, the running time of the brute-force method is $O((n-m+1)m)$ which is $O(nm)$.

When, $m = \frac{n}{2}$, this algorithm has quadratic running time $O(n^2)$.

The Naive string matching algorithm is same as Brute-force Algorithm.

Sometimes we say:- the algorithm as:-

Naive or Simple or Brute Force pattern Matching Algorithm.

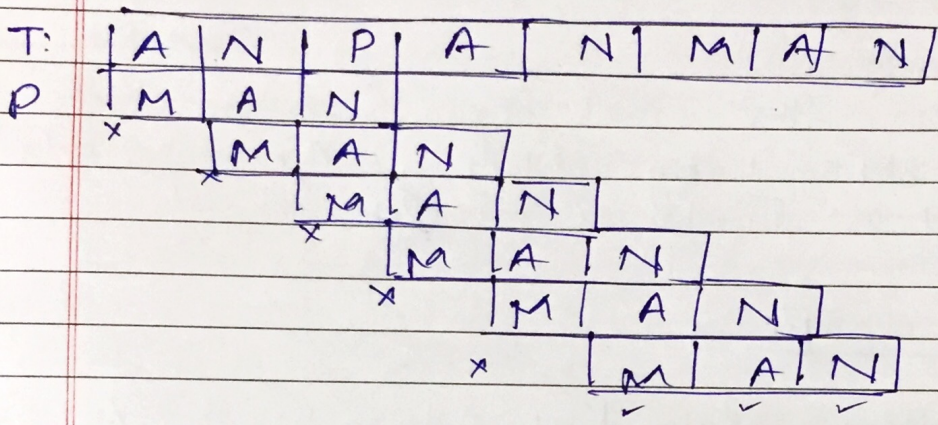
This algorithm, always shifts the window by exactly one position to the right.

CAMPUS 2 Date _____ Page _____

For example :- Naive/Simple/Brute Force Pattern Matching

T = ANPANMAN

P = MAN



An example of Naive or Simple or Brute Force Pattern Matching Algorithm.

3. **Rabin - Karp Sting Matching Algorithm**

In this algorithm, first a calculation is done before doing exact comparisions of characters.

Like, if :-

T is text T[1..n]
P is pattern [1..m]

assume any number q, (a prime number)

then compute p mod q

$$r = p \bmod q$$

Now, the substring of text T that is required to check in order to detect p, is first involves calculation of mod with q.

like; if substring S_1 is to be checked in T,

then first compute $S_1 \bmod q$

if this value matches with r, that,

there is some chance to get the pattern only then the exact comparisons of characters

is done.

If (s_1 mod q) does not match with r, then there is no chance of finding the occurrence of pattern P for s_1, so, the actual comparison is not done.

The next sequence to be checked is obtained by dropping the higher order bit of a number in the text.

For example:-

Applying Rabin Karp string matching algorithm

(Assuming numerical values of characters)

T: 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1

P = 3 1 4 1 5 , m = size of p = 5

q = 13 (given)

window of size = m = 5 to be checked

T: 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1

First compute r = p mod q

= 3 1 4 1 5 mod 13

= 7 (is remainder)

```
      2416
13 ) 31415
     26
     ---
      54
      52
      --
       21
       13
       --
        85
        78
        --
         7
```

thus,

check S_1 = 23590

$(S_1 \bmod q) == r$?

23590 mod 13 == r ? == 7 ?

remainder comes as 8 ≠ 7 ∴ No exact matching of characters will be done.

(Since, there is no chance of getting matching of pattern).

Compute next sequence to be matched:-

$S_2 = 10 \times (23590 - 10000 \times 2) + 2$ (old value)

$= 10 \times (23590 - 20,000) + 2$

$= 10 \times 3590 + 2$

$= 35900 + 2$

$= 35902$

Already checked S_1

T: 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1

next window S_2 to be checked

$S_2 = 35902 \bmod 13 = 9 \neq 7$ No match

$S_3 = 10 (35902 - 10,000 \times 3) + 3$

$= 10 (35902 - 30,000) + 3$

$= 10(5902) + 3$

$= 59020 + 3 = 59023$

S_3 mod q ?

59023 mod 13 = 3 ≠ 7 no match

removing higher order bit

$S_4 = 10(59023 - 10{,}000 \times 5) + 1$ ← next digit after window

$= 10(59023 - 50{,}000) + 1$

$= 10(9023) + 1$

$= 90230 + 1$

$= 90231$ ← window is slided at this new sequence

S_4 mod q ? 90231 mod 13 = 11 ≠ 7

no match for removing higher order bit

$S_5 = 10(90231 - 10{,}000 \times 9) + 4$

$= 10[0231] + 4$ ← new sequence

$= 02314$

This is obtained by looking next digit number in text

S_5 mod 13 = ? 02314 mod 13 = 0 ≠ 7

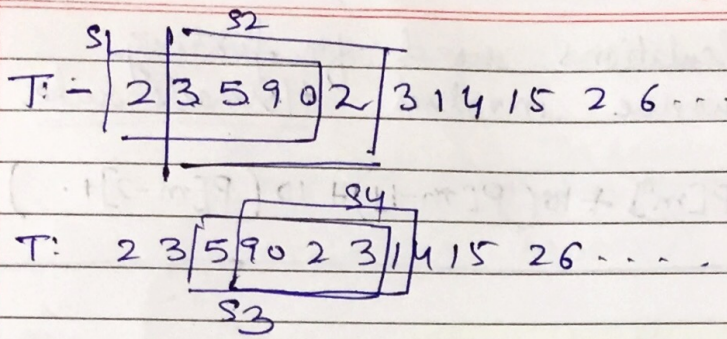
no match

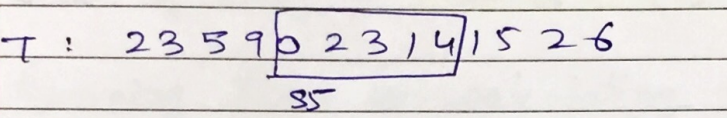
.
.
.

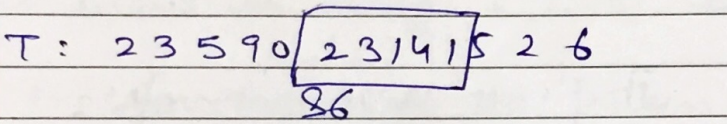
$S_7 = 31415$

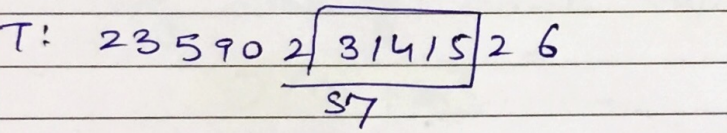
S_7 mod q ? = 31415 mod 13 = 7

Match found









This method can be viewed as sliding window of size m which moves along the text array, T, and numbers in this window are matched with window containing the pattern.

The window slides along text till it get a window ~~width~~ with same modulo as that of pattern window.

If modulo is same but pattern is not matched, then it is called as Spurious Hit (or a false Hit) (or a fake match of modulo as pattern is not found).

The calculations involved in finding next window e.g.

$$s_2 = 10\,(23590 - 10{,}000 \times 2) + 2$$

(To remove higher order bit; ↑ next digit)

This is called as Horner's Rule.

Running time for calculating n different windows in array T[1..n] is O(n) and for matching the pattern O(m), so the running time of Rabin-Karp method is

$$O(n+m)$$

HASHING in Rabin-Karp Algo:-

The Rabin-Karp algorithm involves the calculation of hash values (tricks) to convert text into a numerical value.

Following is the example, how Rabin-Karp algorithm is applied on the following text T and pattern P:-

T:- A A B A A C B A A

P:- B A A.

T: A A B A A C B A A

P: B A A

alphabets are:-

	A	B	C	D	.....	Z	
Value =	1	2	3	4		26	

Total Alphabet size = d = 26

Assume a prime number (large value)
(as large value ensures less collisions) :-

Let, prime number = q = 5381

n = size of text ⇒ 9

m = size of pattern = m

⇒ B A A
1 2 (3)

A	A	B	A	A	C	B	A	A
1	2	3	4	5	6	7	8	(9) = n

∴ window size to be checked = m = 3

Computing Hash value of pattern

(strong hash function)

2	1	0
B	A	A

A-1
B-2
C-3

B ⇒ 2×26^2 ; A ⇒ 1×26^1 ; A ⇒ 1×26^0

$$\left[\left[\left(2 \times 26^2\right) \bmod q + \left(1 \times 26^1\right)\right] \bmod q + 1 \times 26^0\right] \bmod q$$

$$= \Big[\Big[(2\times 26^2) \bmod 5381 + (1\times 26^1)\Big] \bmod 5381 + (1\times 26^0)\Big] \bmod 5381$$

$$= \Big[\Big[(1352 \bmod 5381) + 26\Big] \bmod 5381 + (1\times 1)\Big] \bmod 5381$$

$$= \Big[\Big[1352 + 26\Big] \bmod 5381 + 1\Big] \bmod 5381$$

$$= \Big((1378 \bmod 5381) + 1\Big) \bmod 5381$$

$$= (1378 + 1) \bmod 5381$$

$$= 1379$$

Hash value for pattern $= r \Rightarrow \boxed{p \bmod q}$

has been made stronger

$\therefore$ Here = hash value for pattern (pattern $\Rightarrow$ BAA) $\rightarrow = r = 1379$

Now!- take window size = m=3,
& start comparing:-

T: AA B A A C B A A

P = BAA = 1379 is a i.e. Hash value of pattern

Hash value of AAB is $2 1 0$

A - 1
B - 2
C - 3
⋮

1×26^2 1×26^1 2×26^0

$$\therefore \Big[\Big[(1\times26^2) \bmod 5381 + (1\times26^1)\Big] \bmod 5381 + (2\times26^0)\Big] \bmod 5381$$

$$= \Big[\Big[(676 \bmod 5381) + 26\Big] \bmod 5381 + (2\times1)\Big] \bmod 5381$$

$$= \Big[(676+26) \bmod 5381 + 2\Big] \bmod 5381$$

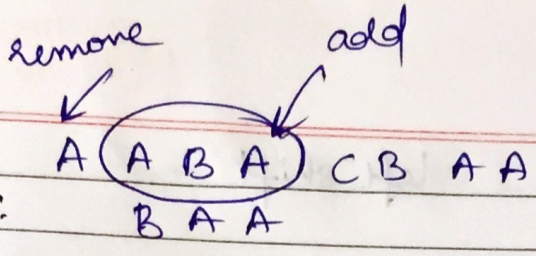
$$= (702+2) \bmod 5381$$

$$= 704 \bmod 5381 = 704$$

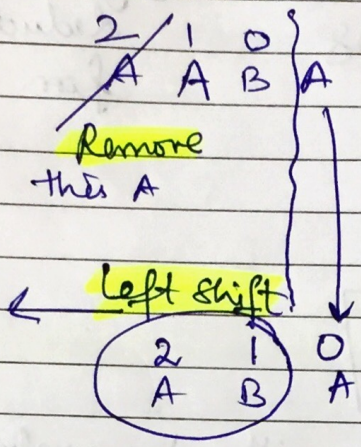
704 is not matching with 1379 (i.e. ≠)

Match not found.

So, Move the window.



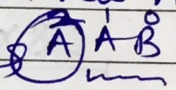
In order to move the window:-



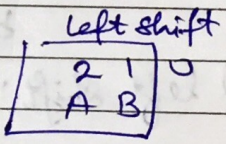
Left shif AB and then add new A

So, steps are:-

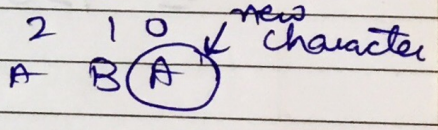
(1) Remove



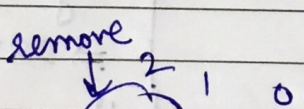
(2) left shift AB



(3) Add new character



Step 1



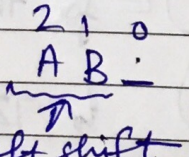
So, (1×26^2) is deducted from Hash value:-

Hash value (AAB) = 704

$\therefore$ Hash value (AAB) $-(1 \times 26^2) = 704 - 676$

$= 28$

Step 2:-



2 1 0
← A B left shift

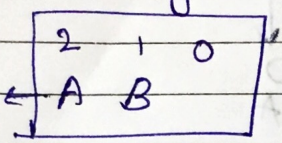
involves ↓

hash value of AB = 28 [got after deducting A from AAB]

↓

to be left shifted

This is obtained by :-



by multiplying it with base value = d = 26

28 X 26

↑ base value = d

will left shift :

= 728

Note :-
If normal number is shifted
← 28
it is multiplied by 10
⇒ 28 × 10
is 280
Left shifted

Step 3 Now : 2 1 0
A B is 728

add new character

2 1 0
A B (A) ← for this

add (1 × 26^0) to 728

∴ 728 + (1 × 26^0) = 728 + 1 = 729

But 729 ≠ 1379

So, match not found

∴ move the window.

T: A A B A C B A A
P: B A A

and the process will go on, till we find the match or text is completely scanned and pattern is not there.