# BIPARTITE GRAPH :—

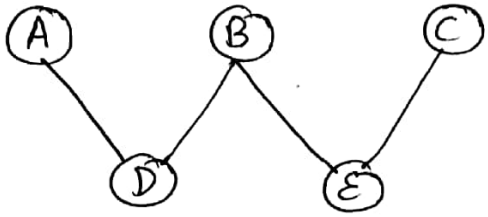
**Definition :-** A graph $G = (V, E)$ is bipartite if the vertex set $V$ can be partitioned into two sets $V_1$ and $V_2$ (the bipartition) such that :

1) $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \phi$ (Null)

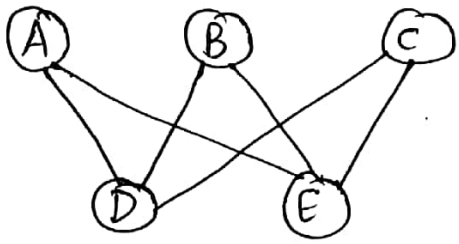
2) No two vertices in the same subset are connected by an edge in $E(G)$ i.e. whenever $v_1, v_2 \in V_1$ then $\{v_1, v_2\} \notin E(G)$ and also whenever $v_3, v_4 \in V_2$ then $\{v_3, v_4\} \notin E(G)$.

* The partition $V = V_1 \cup V_2$ is called a bipartition of $G$.
* If each vertex of $V_1$ is joined with each vertex $V_2$, then the graph $G$ is called complete bipartite graph.
* Trees are examples of bipartite graphs. If $G$ is bipartite, it is usually denoted by $G = (V_1, V_2, E)$, where $E$ is the set of edges.
* No odd cycle in Bipartite

Example:-




⇒ Bipartite graph '$G_1$'




⇒ Complete Bipartite Graph '$G_2$'

$V_1 = \{A, B, C\}$ → (No Edge Connected in b/w A, B & C)

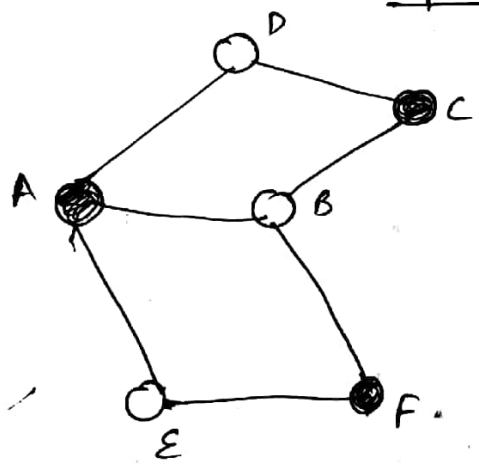
$V_2 = \{D, E\}$ → (No edge b/w D, E)

$V_1 \cup V_2 = V$

$V_1 \cap V_2 = \phi$

$V = \{A, B, C, D, E\}$

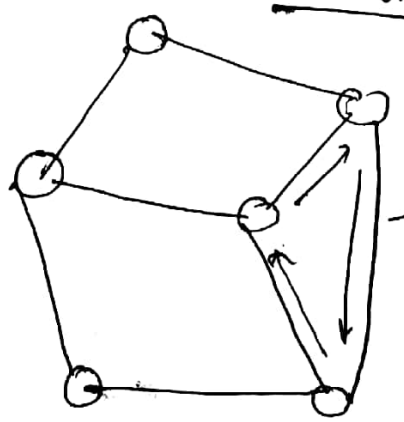
**Bipartite**




$V_1 = \{A, C, F\}$

$V_2 = \{D, B, E\}$

**Non-Bipartite**




✗

# The Euclidian Algorithm :-

## GCD :-

Ex:- gcd (10, 15) = 5

### (1) Brute force Method :-

I/P = Two no.'s a & b

O/P = GCD (a, b)

Brute force says you have to try all No.'s.

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

⇒ only 1 & 5 divide both 10 & 15

So we can check upto 10 only.

* gcd ≤ min(a, b)

∴ gcd (10, 15) = 5

O(min(a, b)) iteration

So, we say that Brute force is inefficient becoz it takes more Time to compute result.

(21) Euclid's Algorithm is used to Compute GCD :-

Formula :-

Recursive Case → $GCD(a,b) = GCD(b, a \% b)$

Base Case → $GCD(a,0) = a$

$gcd(10,14) = gcd(14, 10\%.14)$

$= gcd(14, 10)$

$= gcd(10,14)$

So,

formula: $gcd(a,b)$ (Convert into) $gcd(b,a)$ where $a<b$, $b>a$

$= gcd(10, 14\%.10)$

$= gcd(10, 4)$

$= gcd(4, 10\%.4)$

$= gcd(4, 2)$

$= gcd(2, 4\%.2)$

$= gcd(2,0)$ — Base case

$= \boxed{2}$ Ans.

$\boxed{GCD(10,14) = 2}$ Ans.

GCD ⇒ largest no. that divides both no.s evenly.

Eg:- GCD of 27, 36 ⇒ 9

**Algorithm :- Euclid Division GCD(m,n) ;-**

1) If n=0, return 'm' as gcd & stop. else go to step 2.

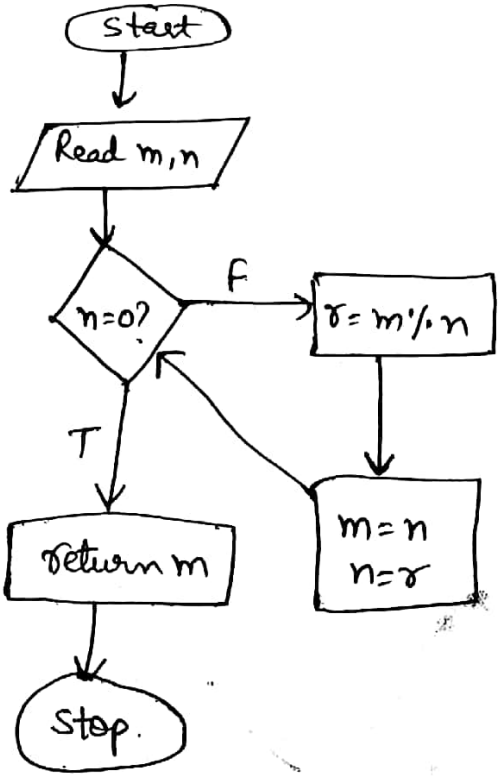
2) Find the remainder of m/n & assign it to r.

3) Assign the value of n to m & r to n & go to step 1.

**Pseudocode :-**

```
Euclid GCD(m,n)
// Inputs : 2 integers m & n
// Outputs : GCD of m & n.
While (n != 0)
    r ← m % n
    m ← n
    n ← r
return m
```

**Flow Chart :-**



# RABIN-KARP ALGORITHM :-

## The Problem of STRING MATCHING :

- String matching involves finding all occurrences of a given pattern in a text string.
- Use Cases :
  - Searching for a word in a file
  - Bio-informatics - DNA sequence searching
  - Search Engines
  - Plagiarism detection software

## Terminologies :-

1) Pattern to search is a array $P[1....m]$
2) Text is represented by $T[1...n]$
3) Both draw characters from finite alphabet $\Sigma$
4) P occurs with a valid shift 's' in 'T' if
$$P[1...m] = T[1+s ..... m+s]$$
5) The string-matching problem is the problem of finding all valid shifts with with a given pattern P occurs in a given text 'T'.

## NAIVE APPROCH :-

⇒ This method checks for matches of the Pattern 'P' at every place in the text 'T'.

```
Naive-String-Matcher (T, P)
1) n = T.length
2) m = P.length
3) for s = 0 to n-m
4)     if P[1...m] == T[s+1 ..... s+m]
5)         print "Pattern occurs with shift" 's'.
```

Complexity :- $O((n-m+1)m)$

Disadv. →

# Rabin - Karp Algorithm →

Developed by Michael O. Rabin & Richard M. Karp.

Pattern = Substring with text using Hashing.

Eg:- T = 3 1 2 3 4 8 6 2

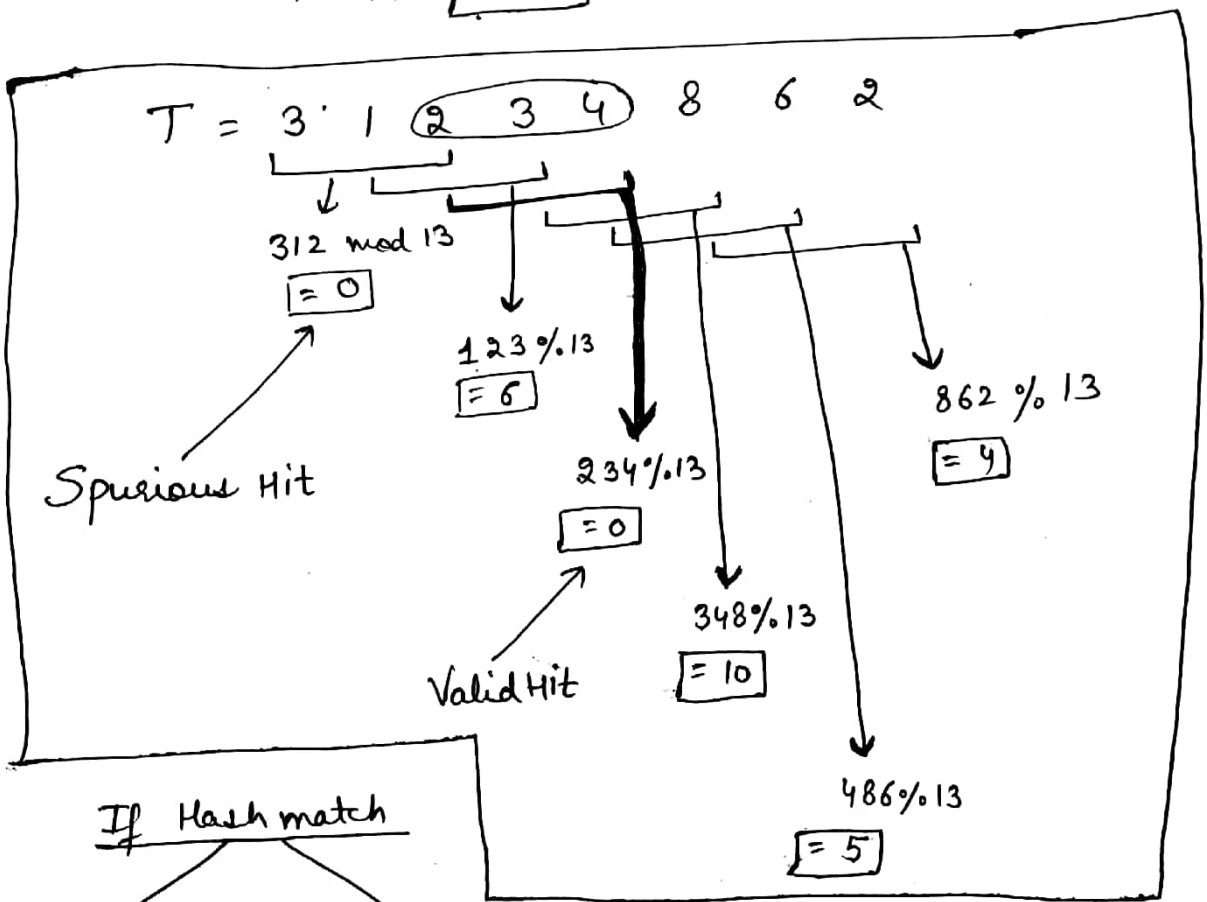
P = 2 3 4

Hashing 'q' = Random Prime No.

let q = 13

P mod q = 234 mod 13

Hash ⇒ [= 0] → is the Hash of Pattern




If Hash match

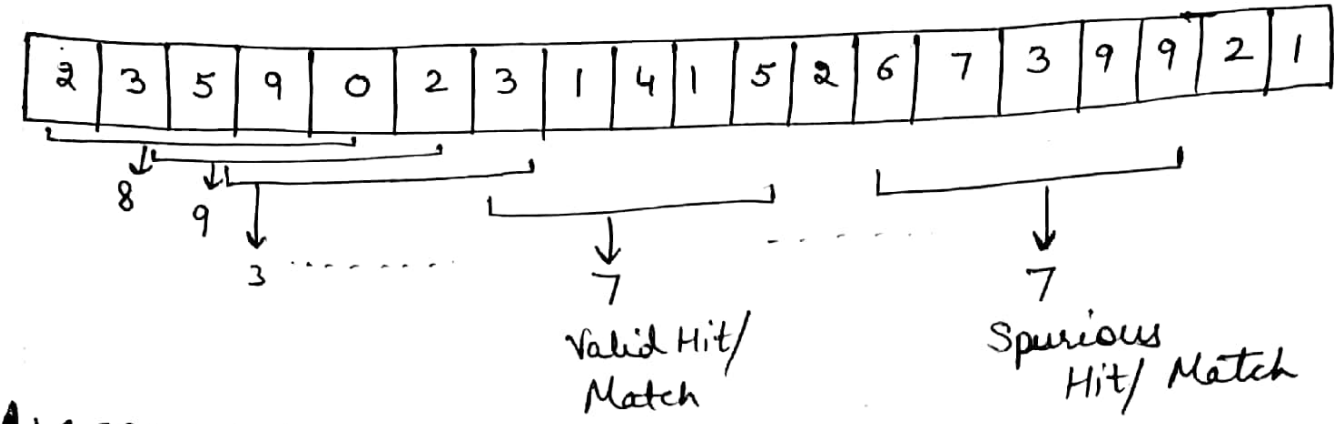
- if all the characters of 'P' & 'T' matches ⇓ Valid HiT
- if all the characters of 'P' & 'T' doesn't matches ⇓ Spurious HiT

Eg:- $T = 2\,3\,5\,9\,0\,2\,3\,1\,4\,1\,5\,2\,6\,7\,3\,9\,9\,2\,1$

$P = 31415 \Rightarrow P \bmod q = 31415 \bmod 13$

$h^P \Rightarrow (= 7)$ Hash

let $q = 13$

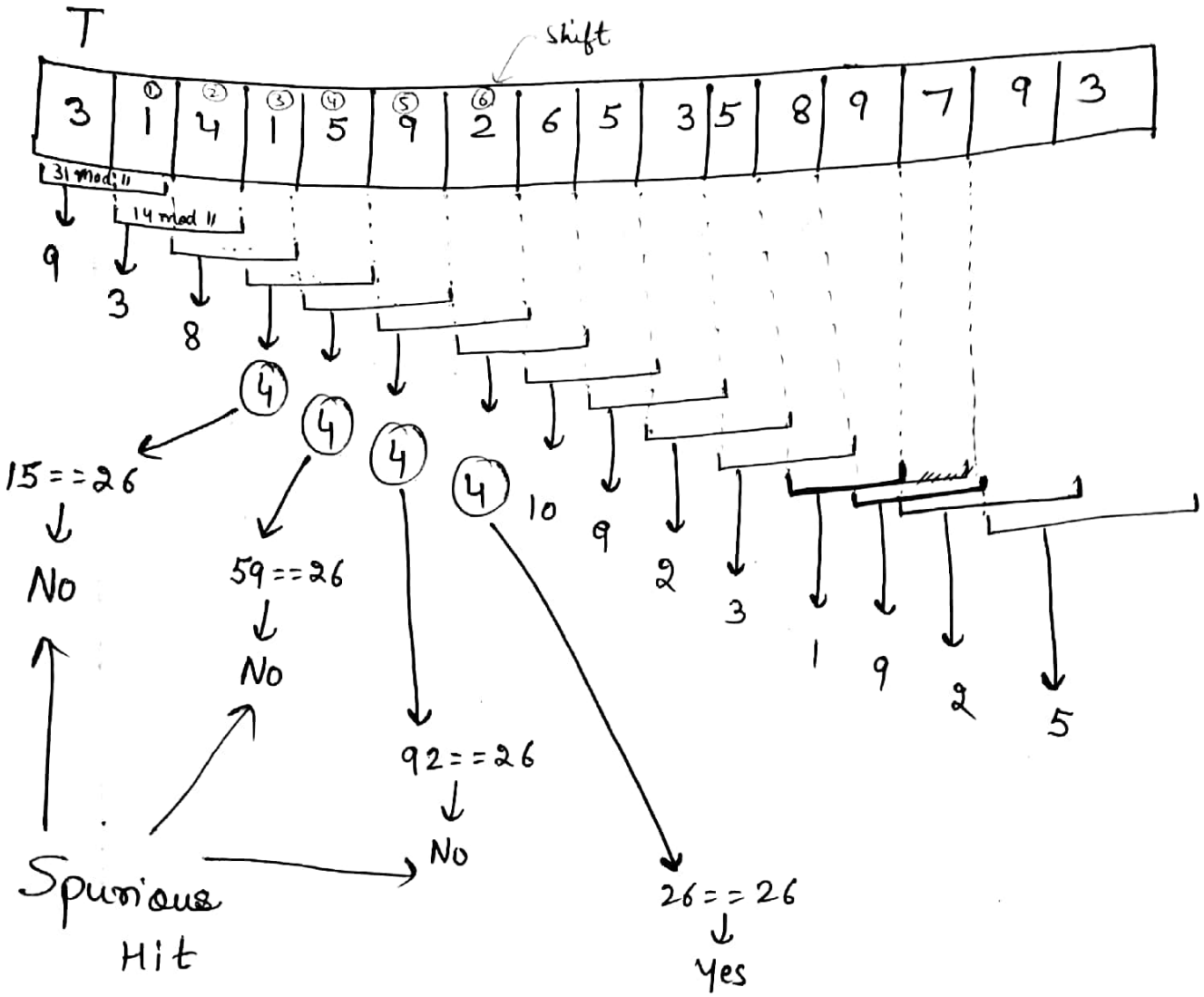



ALGORITHM:-

```
Rabin_Karp(T, P)
n = T.length
m = P.length
h^P = Hash(P[...])      // P mod q (where q → random prime No.)
h^T = Hash(T[...])      // T mod q
for s = 0 to n-m        // s ⇒ SHIFT
    if (h^P == h^T)          ↓ Matching all character of 'P' & 'T'.
        [if (P[0....m-1]) = T[s+0,.....s+m-1])]
            Print " Pattern found with shift "s

    if (s < n-m)
        h^T = Hash(T(s+1,..... s+m))
```

Eg:- P = | 2 | 6 | , q = 11 , $h^P$ = 26 mod 11 = (4)



Now, Consider Example for a Character

**Case 1 :-**

$T =$ a a a a a b (positions 1 2 3 4 5 6)

$n = 6$ → Size of Text.

1+1+1 = 3

3

3

4

for simplification, we use a non code (or use ASCII code)

a → 1

b → 2

c → 3

d → 4

e — 5

f — 6

g — 7

h — 8

i — 9

j — 10

$P =$ a a b (positions 1 2 3)

$m = 3$ → Size of Pattern

1 + 1 + 2 = 4

$h^P = 4$

↑ if you want to multiply that is your wish to what type of hash fun.$^c$ you are using.

---

**Case 2 :-**

$T =$ a b c d a b c e (positions 1 2 3 4 5 6 7 8)

$n = 8$

1 + 2 + 3

6

Subtract First letter

6 (-1) + 4 ← add next letter

= 9

⇒ Rolling Hash fun.$^c$

8

7

6

10 Valid Hit

$P =$ b c e (positions 1 2 3)

$m = 3$

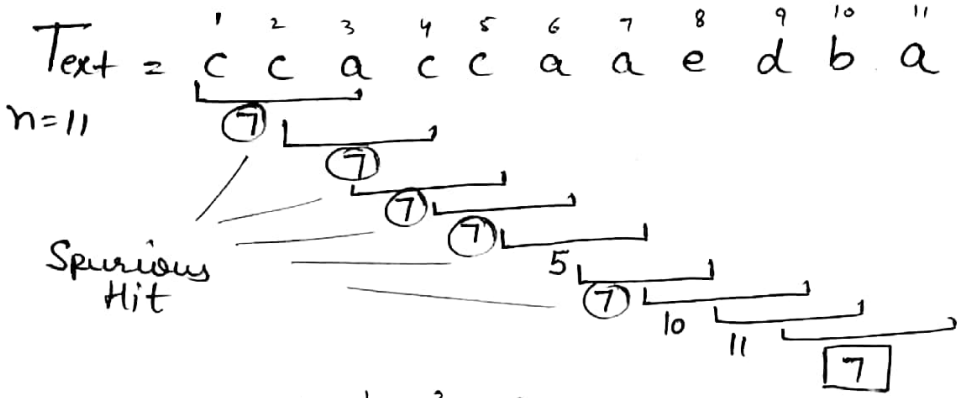
2 + 3 + 5 = 10

Hash

Av. Case of Rabin Karp ⇒ $O(n-m+1)$

## Drawback of Case 1, Case 2 Example :—

Consider another example to discuss about Drawback :




Pattern = d b a
1 2 3
m=3

4 + 4 + 1

$h^p = 7$ Hash code/value

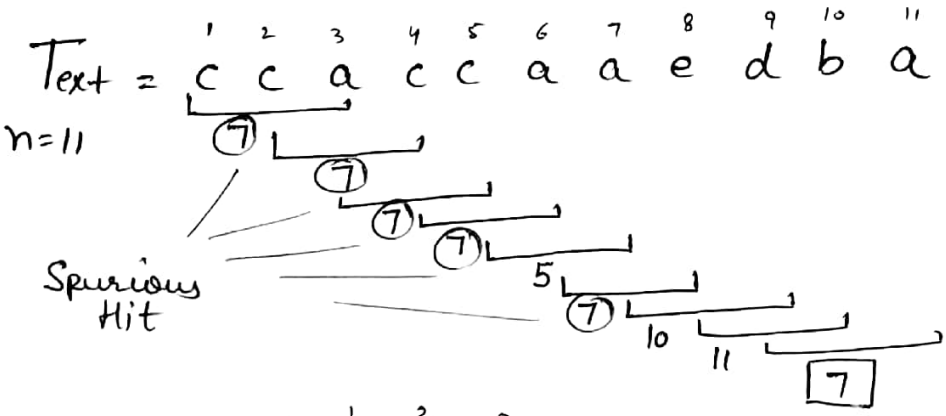
Because, we are using very simple Hashing method, the result is no. of Spurious Hit is maximum as compared to valid Hit.

So, we can say that if we take such a simple Hash function then there is a possibility of collision with the other substring which are having the same code though they are not having the same code.

| | |
|---|---|
| Max. Time taken | $\Rightarrow O(mn)$ |
| When there is no Spurious Hit. | $\Rightarrow O(n-m+1)$ |

## Drawback of Case 1, Case 2 Example :-

Consider another example to discuss about Drawback :




Pattern = d b a
1 2 3
m=3

4 + 4 + 1

$h^p = 7$ Hash Code/Value

Because, we are using very simple Hashing method, the result is no. of Spurious Hit is maximum as compared to valid Hit.

So, we can say that if we take such a simple Hash function then there is a possibility of collision with the other substring which are having the same code though they are not having the same code.

| | |
|---|---|
| Max. Time taken | $\Rightarrow O(mn)$ |
| When there is no Spurious Hit | $\Rightarrow O(n-m+1)$ |

So, how to avoid Spurious Hit :-

we have to take a strong Hash function.

$T$ (n = 11) = c c a c c a a e d b a ⇒ (421)

133

$3\times10^2 + 3\times10^1 + 1\times10^0$

$300 + 30 + 0$

$= 331$

331

$3\times10^2 + 1\times10^2 + 3\times10^0$ OR $= 300 + 100 + 3$ $= 313$

Rolling Hashing fun.

$$\left[\left[3\times10^2 + 3\times10^1 + 1\times10^0\right] - 3\times10^2\right]\times10 + 3\times10^0$$

old value = 331; $-C$ First value; $+C$ add next value

$\Rightarrow [331 - 300]\times10 + 3$

$\Rightarrow [31\times10] + 3$

$= 313$

$P$ (m = 3) = d b a

$4\times10^2 + 2\times10^1 + 1\times10^0 = 421$

Base Value [if 26 alphabet take (a ---- z / 1 ---- 26)] ⟹ Then we take (26) as a base value

In q ⇒ [a ---- j] 10 alphabet we are using

General ⇒

If m ⇒ letter in a pattern

$$P[1]\times10^{m-1} + P[2]\times10^{m-2} + P[3]\times10^{m-3}$$

So, By doing this method, we can minimize the Spurious Hit.

Time Taken by this algo. ⇒ $O(n-m+1)$

Worst Case ⇒ $O(mn)$

→ If still there is Spurious Hit

# KNUTH - MORRIES - PRATT ALGORITHM (KMP):-

This algorithm works on proper prefix and proper Suffix.

NOTE:- we are not backtrack in the main string.

abcd → Proper Suffix : Starts from R.H.S.

{ d, cd, bcd } abcd

Proper Prefix : Starts from L.H.S

{ a, ab, abc } abcd

## Complexity :-

Let Text 'T' = m . length

Pattern 'P' = n . length

$$O(m+n)$$

∴ of this, KMP algorithm is Better than Rabin-Karp algorithm.

Becoz this KMP has better Complexity than Rabin-Karp algorithm.

# How to find the Π- Table / Longest Proper Prefix :-

- Only Pattern is Considered

$P_1$ :

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| a | b | a | b |
| 0 | 0 | 1 | 2 |

→ Π- table of $P_1$ pattern

↑ a appear at previous location i.e. at location 1
//ly b repeated at prev. loc 2

$P_2$ :

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| a | b | c | d | a | b | c | y |
| 0 | 0 | 0 | 0 | 1 | 2 | 3 | 0 |

→ Π- table of $P_2$

$P_3$ :

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | a | b | e | a | b | f |
| 0 | 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 0 |

$P_4$ :

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | a | b | f | a | b | c |
| 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 1 | 2 | 3 |

# Example on KMP Algorithm :-

T : i=1

| a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

P: j=0

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| a | b | a | b | d |
| 0 | 0 | 1 | 2 | 0 |

Π-table

STEPS :-

1) Take two variable i and j
   i = String (T(1)) , j = P[0]

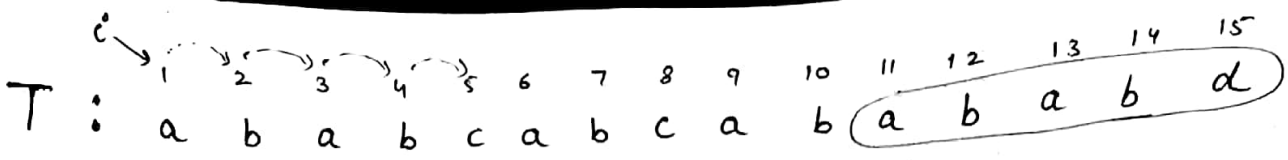
2) Compare T(i) with P(j+1)

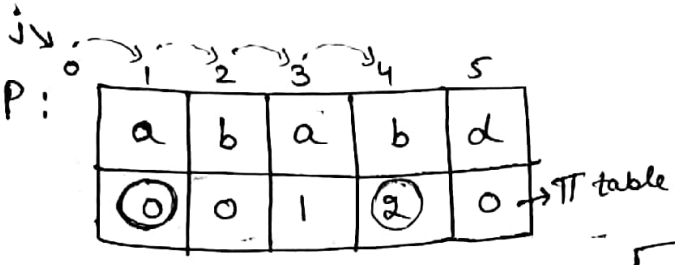
   a) if Match is found (Move both i and j to right)

   b) if Mismatch (move j to the location as per Π table Index)

   c) if j=0 (move i to the right)

→




STEP1:- $i = 1$ , $j = 0$

STEP2:- Compare $T[1]$ with $P[0+1]$

a) i.e. $a = a$

Now, $i = 2$ $j = 1$

$T[2] = P[1+1]$

$b = b$

$i = 3$ , $j = 2$

$T[3] = P[2+1]$

$a = a$

$i = 4$ , $j = 3$

$T[4] = P[3+1]$

$b = b$

$i = 5$, $j = 4$

$T[5] = P[4+1]$

$T[5] = P[5]$

$c \neq d$

Mismatch occur

b) Move j to the location as per Π-table index. $[j=4]$ → Index of Π-table is (2)

So, $j = 2$ — updated

$i = 5$, $j = 2$ ⇒

a) $T[5] = P[2+1]$

$T[5] = P[3]$

$c \neq a$

Again Mismatch.

b) Now $j = 0$ updated

c) if $j = 0$

Then move i to right

Now, $i = 6$

$i = 6$ , $j = 0$

a) $T[6] = P[1]$

$a = a$ ✓

$i = 7$ , $j = 1$

$T[7] = P[2]$

$b = b$ ✓

b) $i = 8$ , $j = 2$

$T[8] = P[3]$

$c \neq a$ Mismatch

Now,

b) $j = 0$

c) Now 'i' update

$i = 9$

a) $i = 9$ , $j = 0$

$T[9] = P[1]$

$a = a$ ✓

$i = 10$, $j = 1$

$T[10] = P[2]$

$b = b$ ✓

→

$i=11$, $j=2$

$T[11] = P[3]$

$a = a$ ✓

$i=12$, $j=3$

$T[12] = P[4]$

$b = b$ ✓

$i=13$, $j=4$

$T[13] = P[5]$

$a \neq d$

Mismatch

b) $\boxed{j=2}$ , $i=13$

updated

$T[13] = P[3]$

$a = a$ ✓

$i=14$, $j=3$

$T[14] = P[4]$

$b = b$ ✓

$i=15$, $j=4$

$T[15] = P[5]$

$d = d$ ✓

(End of string &
∵ i=15, reaches
end pt.

Pattern found