a=1 b=2 c=3 d=4 e=5 f=6 g=7
h=8 i=9 j=10

Rabin-Karp Algorithm:

n = 6 m = 3

Text: a a a a a b pattern: a a b

3 ≠ 4

1+1+2 = 4

hash func

hash code

a a a a a b

3 ≠ 4

a a a a a b

3 ≠ 4

a a a a a b

4 = 4

→ Keep on checking for the ~~hash~~ hash function of each m letters

→ This sliding is called as rolling hash func

E.g. 2

~~n=5~~ n = 8

a b c d a b c e pattern: b c e

1 2 3

2+3+5 = 10

6 ×

9 ×

8 ×

7 ×

67

10 ✓

a=1 b-2 c-3 d-4 e-5 f-6 g-7
h-8 i-9 j-10

Rabin - Karp Algorithm :

n = 6 m = 3

Text : a a a a a b pattern : a a b

3 ≠ 4

1 + 1 + 2 = 4

hash func^n — hash code

a a a a a b

3 ≠ 4

a a a a a b

3 ≠ 4

a a a a a b

4 = 4

→ Keep on checking for the ~~hash~~ hashfunction of each m letters

→ This sliding is called as rolling hash func^n

E.g. 2

~~n = 5~~ n = 8

a b c d a b c e pattern : b c e

1 2 3 2 + 3 + 5 = 10

6 ×

9 ×

8 ×

7 ×

6 ×

10 ✓

Drawback:

Spurious hits: Hash codes are matching but pattern is not

n = 11

c c a c c a a e d b a

m = 3

d b a

4 2 1 = 7

7x

7x

7x

7x

Here the hashcodes match but the pattern isn't the same, ∴ there are spurious hits.

In ~~cast~~ case of worst case, the time complexity is ~~O(m~~ O(mn) & in the best case O(n-m+1)

So, we can't use such a weak hash function we need to use a strong hash function.

Let's consider the same e.g.

n = 11

Text: c c a c c a a e d b a

m = 3

pattern: d b a

4 2 1

300 + 30 + 1 = 331

421

313

133

331

$4 \times 10^2 + 2 \times 10 + 1 \times 10^0$

$p[1] \times 10^{m-1} + p[2] \times 10^{m-2} + p[3] \times 10^{m-3}$

400 + 20 + 1 = 421

By using this algo. we have reduced the chances of worst-case.

Best case TC : $O(n-m+1)$

Worst case TC : $O(mn)$

Naive String matching :

algo:

→ It is the simplest method using brute force approach

→ It compares 1st character of the pattern with searchable text. If match is found, pointers in both strings are advanced.

→ If match not found, pointer of text is shifted one place to right from previous starting position & pointer of pattern is reset.

→ The process is repeated until the end of text

$$T[S+1 \ldots S+m] = P[1 \ldots m]$$

$$T[S+j] = P[j] \qquad 1 \le j \le m$$
$$0 \le S \le n-m$$

E.g. T = acaabc P = aab

Iter-1 of

a	c	a	a	b	c

✓ ✗

S=0

a	a	b

← P

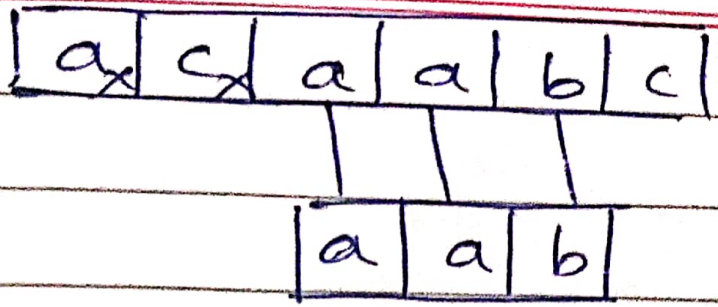
Iter-2 of

a	c	a	a	b	c

S=1 ×

a	a	b

Iter-3



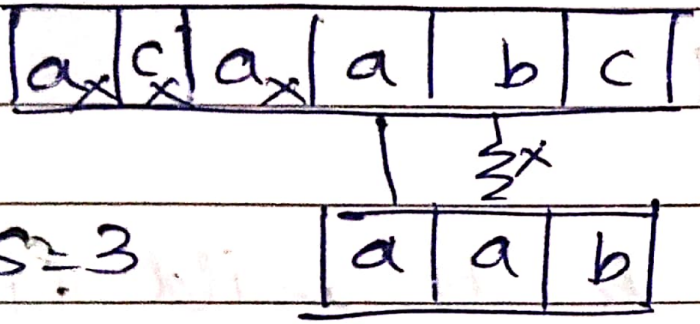
$s = 2$

$\therefore$ Pattern is found at $s = 2$

Now we need to check ~~whether~~ whether there are for patterns ahead

Iter-4

Iter-4



$s = 3$

Stop here

$\therefore$ Pattern found at $s = 2$

Finite Automata

→ Idea of this approach is to build finite automata to scan text T for finding all occurences of pattern P

→ This approach examines each character of text exactly once to find the pattern ∴ Thus it takes linear time for matching but pre-processing time may be large.

A finite automata M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$

(i) Q is a finite set of states

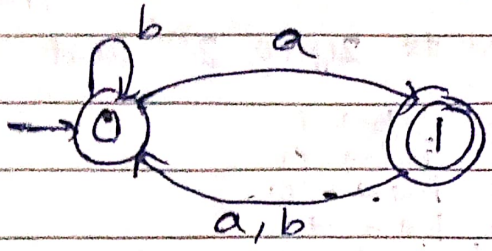
(ii) $q_0 \in Q$ is the starting state

(iii) $A \subseteq Q$ is the set of accepting states

(iv) Σ is the input alphabet (finite)

(v) $\delta : Q \times \Sigma \rightarrow Q$ is the transition function

e.g. of finite automata.



Starting state = 0
Final state = (0)

∴ Here $Q = \{0, 1\}$
$\Sigma = \{a, b\}$

$\delta(0, a) \to 1$ $\quad \delta(1, a) \to 0$
$\delta(0, b) \to 0$ $\quad \delta(1, b) \to 0$

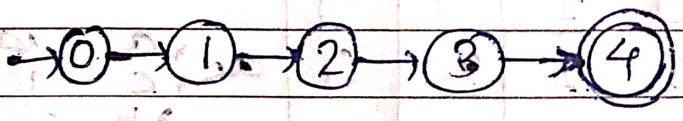
$q_0 = \{0\}$ $\quad A = \{1\}$

Table:

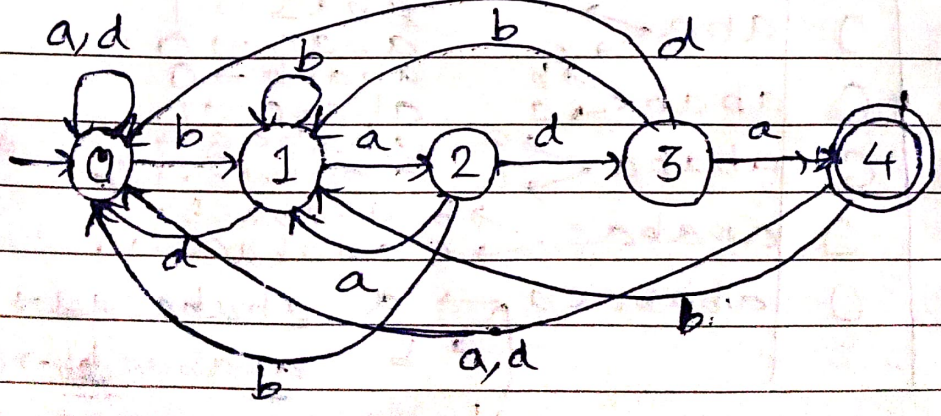
State	Input a	Input b
0	1	0
1	0	0

Construction of transition table with given string.

String: bada



At the 1st step make this



Check how many prefixes are matching how many suffixes

	a	b	d
→ 0	0	1	0
1	2	1	0
2	1	0	3
3	4	1	0
* 4	0	1	0

b ← b (bb → 1), d (bd → 0)

ba ← a (baa → 0), b (bab → 1)

bad ← b (badb → 1), d (badd → 0)

bada → a (badaa → 0), b (badab → 1), d (badad → 0)

Note: Prefix starts from 1st letter & goes upto 2nd last letter

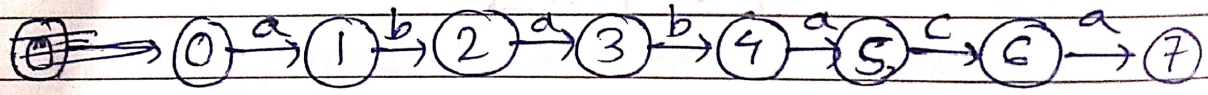
Suffix starts from last letter & goes upto 2nd letter

Now with the help of transition table complete the automata

E.g. P = ababaca

T = abababacaba

Make finite automata for pattern



	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	7	0	0
7	1	2	0

a ← a (aa → 1), c (ac → 0)

ab ← b (abb → 0), c (abc → 0)

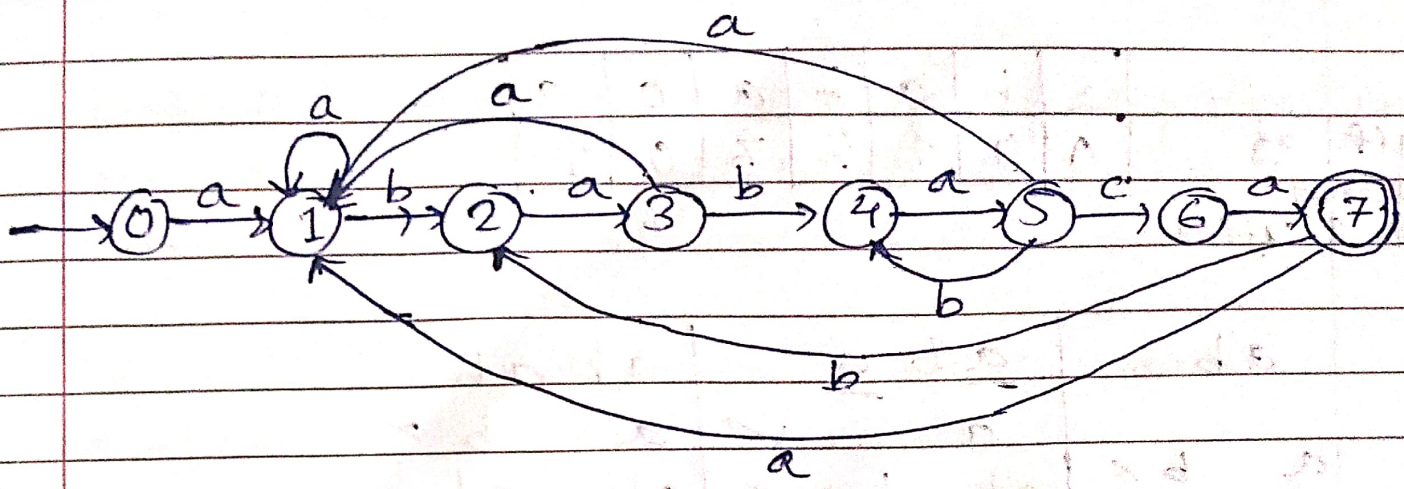
aba → a (abaa → 1), c (abac → 0)

abab ← b (ababb → 0), c (ababc → 0)

ababa → a (ababaa → 1), b (ababab → abab)

abababac ← b (abababacb → 0), c (abababacc → 0)

abababaca → a (abababacaa → 1), b (abababacab → 0), c (abababacac → 0)



Write the text string given

	1	2	3	4	5	6	7	8	9	10	11
State	a	b	a	b	a	b	a	c	a	b	a
0	1	2	3	4	5	4	5	6	7	2	3

Take these values from transition table & previous value in the state

We get $q = 7$ = length of substring & check the index above which is 9.

$\therefore$ Shift $= 9 - 7 = 2$

$\therefore$ Shift 2 bits from left & you get the substring from index

T.C. = ~~Σ~~ $\Theta(m^2|\Sigma|)$

Prefix table Construction:

a	b	a	b	a	c	a
0	0	1	2	3	0	1

Length of matching prefix & suffix

ab

a b ✗

aba

a ✓ a ✓

ab ba ✗

ababa

a ✓ a ✓

ab ba ✗

aba aba ✓

abab baba ✗

ababab

a b ✗

ab ab ✓

aba bab ✗

abababac

a c ✗

ab ac ✗

aba bac ✗

abab abac ✗

ababa babac ✗

ababacaa

a a ✓

ab ca ✗

aba aca ✗

abab baca ✗

ababa abaca ✗

ababac babaca ✗

Knuth-Morrison-Pratt

KMP is a linear time string matching algorithm run in $\theta(n)$. KMP uses concept of prefix & suffix for generation of Π table.

Π table longest prefix that is same as suffix (LPS)

e.g.

1) a b c d a b e a b f

Π 0 0 0 0 1 2 0 1 2 0

→ If is the same as prefix table

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

P a b a b d

Π

	0	1	2	3	4	5
P		a	b	a	b	d
Π		0	0	1	2	0

Steps: S-1) Take two ~~variables~~ i pointers i & j
i = 1 (1st character of text) & j = 0 ~~(null)~~ (not the 1st character of pattern)

S-2) Compare i & j+1

→ If matching move i & j ahead

→ Else ~~back~~ move j to $\Pi[j]$ & i remains where it was

→ Repeat this until ~~i = 1~~ i is not equal to length of string

PAGE NO.:

① a b a b c a b c a b a b a b d

	a	b	a	b	d
	0	0	1	2	0

$b \neq c$

put j to $T[4] = 2$

$\therefore j = 2$

② a b a b c a b c a b a b a b d

	a	b	a	b	d
	0	0	1	2	0

	a	b	a	b	d
	0	0	1	2	0

$b \neq c$

put j to 0

③ a b a b c a b c a b a b a b d

	a	b	a	b	d
	0	0	1	2	0

Now again $a \neq c$

& $j = 0$, so move i ahead

Now $a \neq c$

∴ Move j to 0

---

(2) (4)

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

i: 8 → 9 → 10 → 11 → 12

j:

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

c ≠ a

∴ Move i ahead

a ≠ d

∴ Move j to 2

---

(5)

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

i: 12 → 13 → 14 → 15

j:

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

j: 1 → 2 → 3 → 4 → 5

As j = length of pattern,
we have found the pattern in given text

T.C. = O(N+M) S.C. = O(M)