The Knuth-Morris-Pratt (KMP)Algorithm

Knuth-Morris and Pratt introduce a linear time algorithm for the string matching problem. A matching time of O (n) is achieved by avoiding comparison with an element of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

Components of KMP Algorithm:

1. The Prefix Function (Π): The Prefix Function, Π for a pattern encapsulates knowledge about how the pattern matches against the shift of itself. This information can be used to avoid a useless shift of the pattern 'p.' In other words, this enables avoiding backtracking of the string 'S.'

2. The KMP Matcher: With string 'S,' pattern 'p' and prefix function 'Π' as inputs, find the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrences are found.

The Prefix Function (Π)

COMPUTE- PREFIX- FUNCTION (P)

```
1. m ←length [P]              //'p' pattern to be matched
2. Π [1] ← 0
3. k ← 0
4. for q ← 2 to m
5. do while k > 0 and P [k + 1] ≠ P [q]
6. do k ← Π [k]
7. If P [k + 1] = P [q]
8. then k← k + 1
9. Π [q] ← k
10. Return Π
```

Running Time Analysis:

In the above pseudo code for calculating the prefix function, the for loop from step 4 to step 10 runs 'm' times. Step1 to Step3 take constant time. Hence the running time of computing prefix function is O (m).

Example: Compute Π for the pattern 'p' below:

P :	a	b	a	b	a	c	a

Initially: m = length [p] = 7 Π [1] = 0 k = 0

Step 1: q = 2, k = 0

Π [2] = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0					

Step 2: q = 3, k = 0

Π [3] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1				

Step3: q =4, k =1

Π [4] = 2

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	A
π	0	0	1	2			

Step4: q = 5, k =2

Π [5] = 3

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3		

Step5: q = 6, k = 3

Π [6] = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	

Step6: q = 7, k = 1

Π [7] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

After iteration 6 times, the prefix function computation is complete:

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

The KMP Matcher:

The KMP Matcher with the pattern 'p,' the string 'S' and prefix function 'Π' as input, finds a match of p in S. Following pseudo code compute the matching component of KMP algorithm:

KMP-MATCHER (T, P)

```
1. n ← length [T]
2. m ← length [P]
3. Π← COMPUTE-PREFIX-FUNCTION (P)
4. q ← 0            // numbers of characters matched
5. for i ← 1 to n   // scan S from left to right
6. do while q > 0 and P [q + 1] ≠ T [i]
7. do q ← Π [q]             // next character does not match
8. If P [q + 1] = T [i]
9. then q ← q + 1           // next character matches
10. If q = m                           // is all of p matched?
11. then print "Pattern occurs with shift" i - m
12. q ← Π [q]                       // look for the next match
```

Running Time Analysis:

The for loop beginning in step 5 runs 'n' times, i.e., as long as the length of the string 'S.' Since step 1 to step 4 take constant times, the running time is dominated by this for the loop. Thus running time of the matching function is O (n).

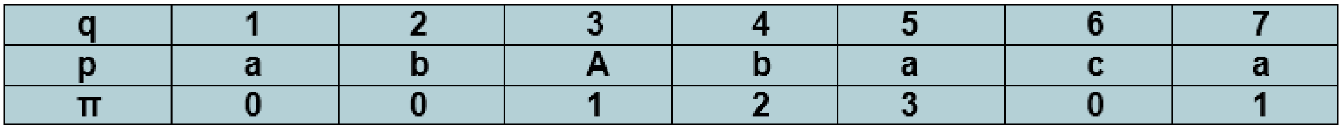
Example: Given a string 'T' and pattern 'P' as follows:

T:	b	a	c	b	a	b	a	b	a	b	a	c	a	c	a

P:	a	b	a	b	a	c	a

Let us execute the KMP Algorithm to find whether 'P' occurs in 'T.'

For 'p' the prefix function, ? was computed previously and is as follows:



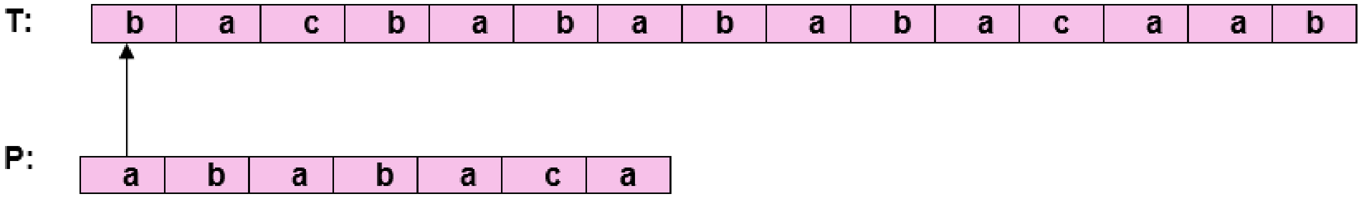
q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

Solution:

```
 Initially: n = size of T = 15
m = size of P = 7
```

Step1: i=1, q=0

Comparing P [1] with T [1]



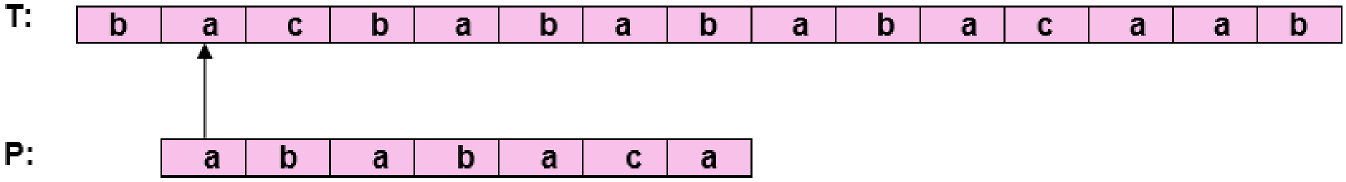
T: b a c b a b a b a b a c a a b

P: a b a b a c a

P [1] does not match with T [1]. ‘p’ will be shifted one position to the right.

Step2: i = 2, q = 0

Comparing P [1] with T [2]



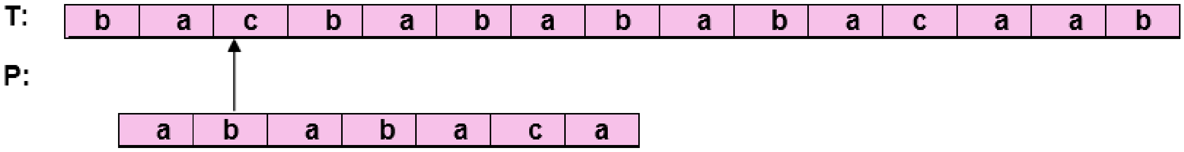
T: b a c b a b a b a b a c a a b

P: a b a b a c a

P [1] matches T [2]. Since there is a match, p is not shifted.

Step 3: i = 3, q = 1

Comparing P [2] with T [3] P [2] doesn’t match with T [3]



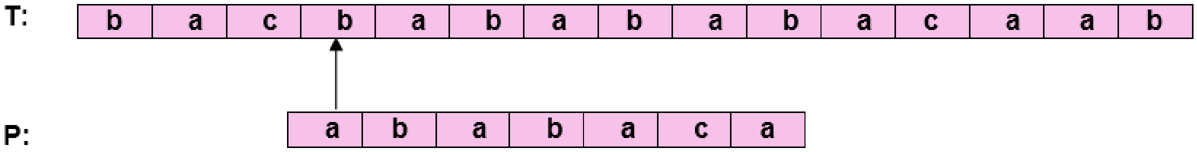
T: b a c b a b a b a b a c a a b

P: a b a b a c a

Backtracking on p, Comparing P [1] and T [3]

Step4: i = 4, q = 0

Comparing P [1] with T [4] P [1] doesn’t match with T [4]

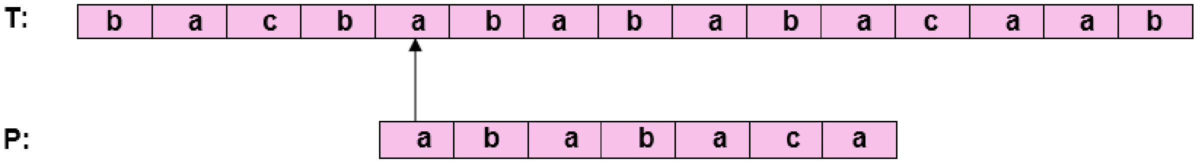


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step5: i = 5, q = 0

Comparing P [1] with T [5] P [1] match with T [5]

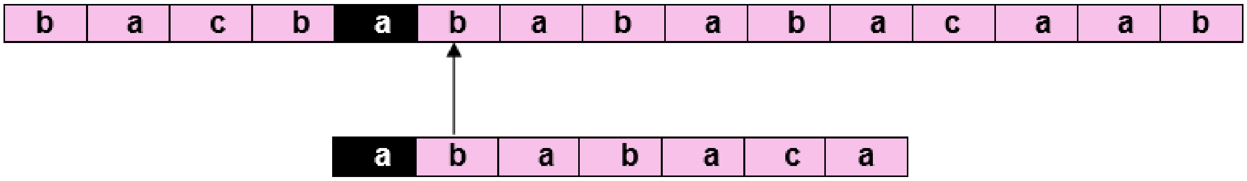


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step6: i = 6, q = 1

Comparing P [2] with T [6]   P [2] matches with T [6]

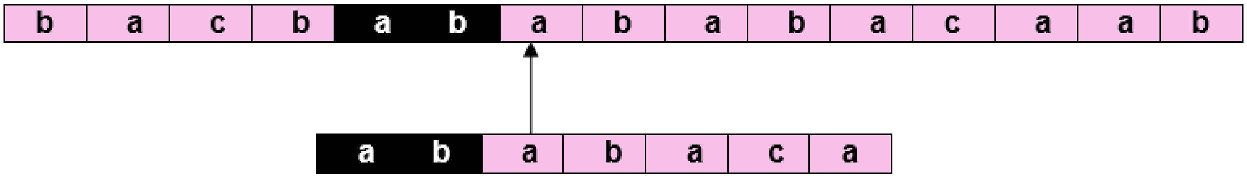


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step7: i = 7, q = 2

Comparing P [3] with T [7]   P [3] matches with T [7]

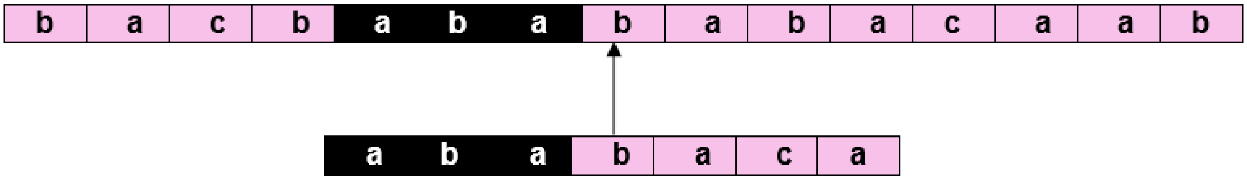


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step8: i = 8, q =3

Comparing P [4] with T [8]   P [4] matches with T [8]

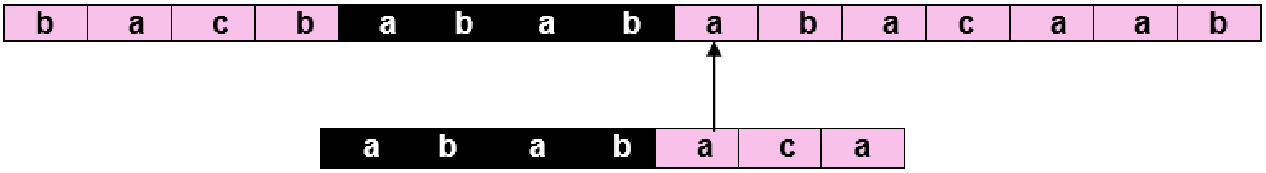


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step9: i = 9, q = 4

Comparing P [5] with T [9]   P [5] matches with T [9]

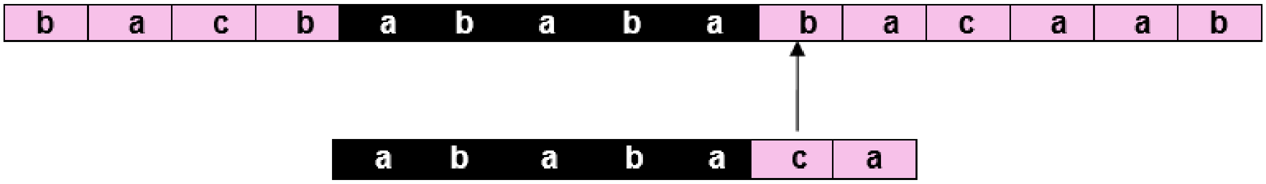


T: b a c b a b a b a b a c a a b

P: a b a b a c a

Step10: i = 10, q = 5

Comparing P [6] with T [10]   P [6] doesn’t match with T [10]



T: b a c b a b a b a b a c a a b

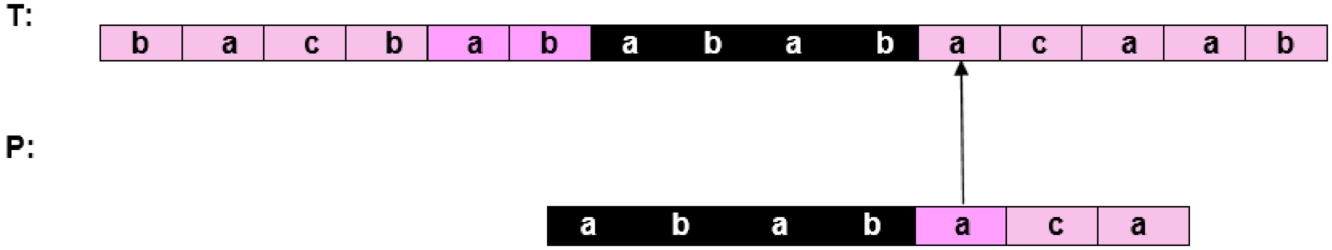
P: a b a b a c a

Backtracking on p, Comparing P [4] with T [10] because after mismatch q = π [5] = 3

Step11: i = 11, q =4

Comparing P [5] with T [11]

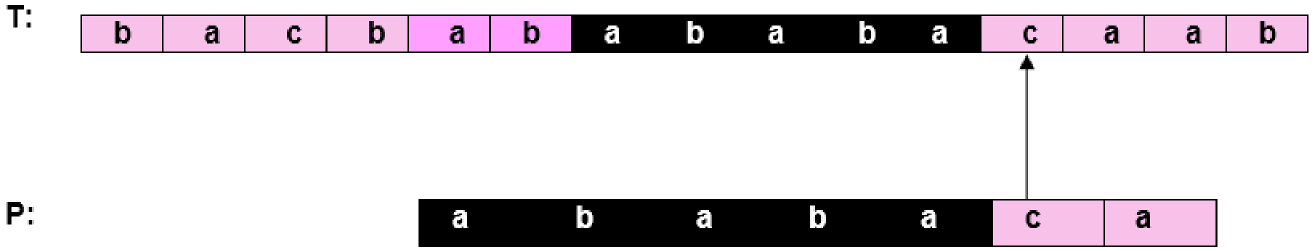
P [5] match with T [11]



Step12: i = 12, q = 5

Comparing P [6] with T [12]

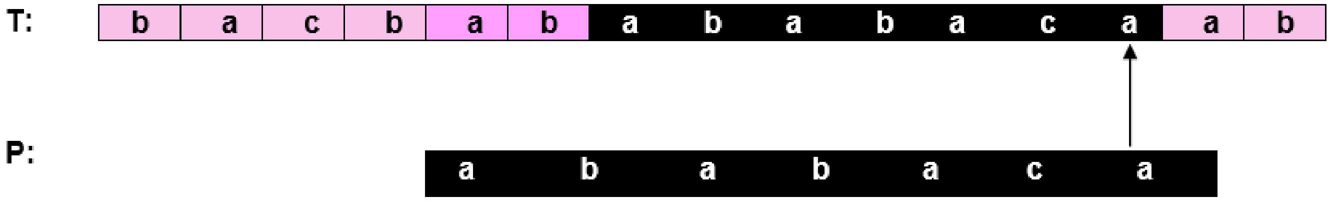
P [6] matches with T [12]



Step13: i = 3, q = 6

Comparing P [7] with T [13]

P [7] matches with T [13]



Pattern 'P' has been found to complexity occur in a string 'T.' The total number of shifts that took place for the match to be found is i-m = 13 - 7 = 6 shifts.