

[Home](#)

[DAA](#)

[DS](#)

[DBMS](#)

[Aptitude](#)

[Selenium](#)

[Kotlin](#)

[C#](#)

[HTML](#)

[CSS](#)

[JavaScript](#)

# 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 ( $\Pi$ ):** The Prefix Function,  $\Pi$  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 ' $\Pi$ ' 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 ( $\Pi$ )

Following pseudo code compute the prefix function,  $\Pi$ :

### COMPUTE- PREFIX- FUNCTION (P)

```
1. m ← length [P]           //'p' pattern to be matched
2.  $\Pi$  [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 ←  $\Pi$  [k]
7. If P [k + 1] = P [q]
8. then k ← k + 1
```

```
9.  $\Pi[q] \leftarrow k$ 
```

```
10. Return  $\Pi$ 
```

## 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  $\Pi$  for the pattern 'p' below:

**P :**

a	b	a	b	a	c	a
---	---	---	---	---	---	---

**Solution:**

Initially:  $m = \text{length}[p] = 7$

$\Pi[1] = 0$

$k = 0$

**Step 1:**  $q = 2, k = 0$ 

$$\Pi[2] = 0$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
$\pi$	0	0					

**Step 2:**  $q = 3, k = 0$ 

$$\Pi[3] = 1$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
$\pi$	0	0	1				

**Step3:**  $q = 4, k = 1$ 

$$\Pi[4] = 2$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	A
$\pi$	0	0	1	2			

**Step4:**  $q = 5, k = 2$ 

$$\Pi[5] = 3$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
$\pi$	0	0	1	2	3		

**Step5:**  $q = 6, k = 3$ 

$$\Pi[6] = 0$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
$\pi$	0	0	1	2	3	0	

**Step6:**  $q = 7, k = 1$ 

$$\Pi[7] = 1$$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
$\pi$	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
$\pi$	0	0	1	2	3	0	1

## The KMP Matcher:

The KMP Matcher with the pattern 'p,' the string 'S' and prefix function ' $\pi$ ' 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  $\leftarrow$  length [T]
2. m  $\leftarrow$  length [P]
3.  $\pi \leftarrow$  COMPUTE-PREFIX-FUNCTION (P)
4. q  $\leftarrow$  0 // numbers of characters matched
5. for i  $\leftarrow$  1 to n // scan S from left to right
6. do while q > 0 and P [q + 1]  $\neq$  T [i]
7. do q  $\leftarrow$   $\pi$  [q] // next character does not match
8. If P [q + 1] = T [i]
9. then q  $\leftarrow$  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  $\leftarrow$   $\pi$  [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,  $\pi$  was computed previously and is as follows:

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
$\pi$	0	0	1	2	3	0	1

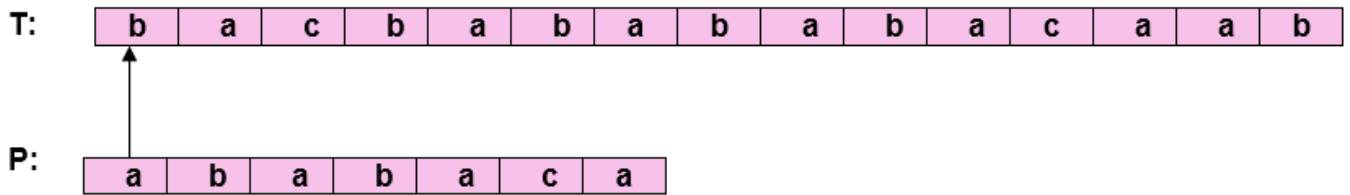
**Solution:**

Initially:  $n = \text{size of } T = 15$

$m = \text{size of } P = 7$

**Step1:**  $i=1, q=0$ 

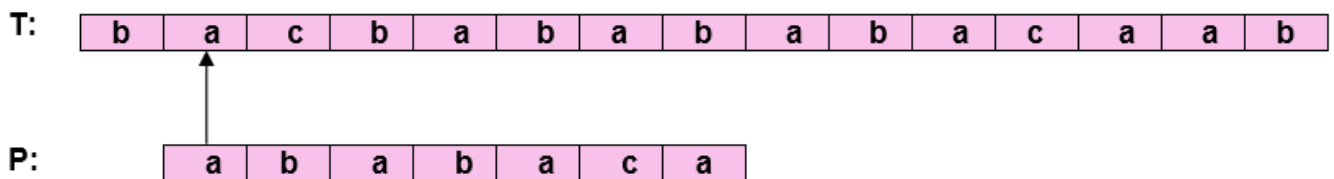
Comparing P [1] with T [1]



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

**Step2:**  $i = 2, q = 0$ 

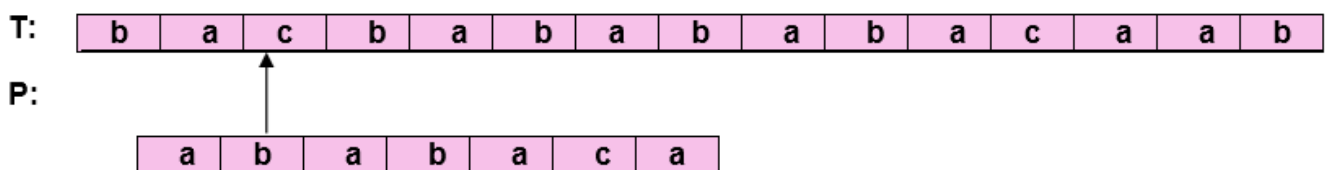
Comparing P [1] with T [2]



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

**Step 3:**  $i = 3, q = 1$ 

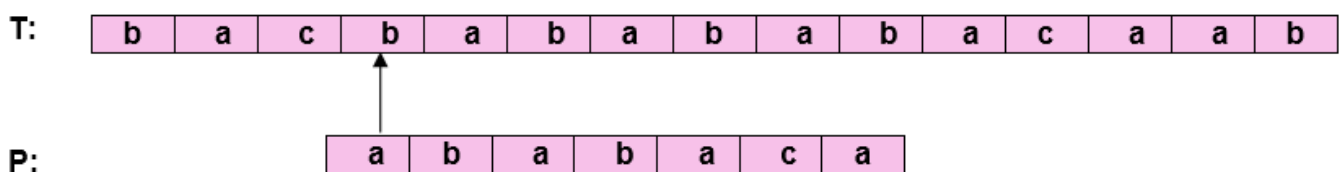
Comparing P [2] with T [3]      P [2] doesn't match with T [3]



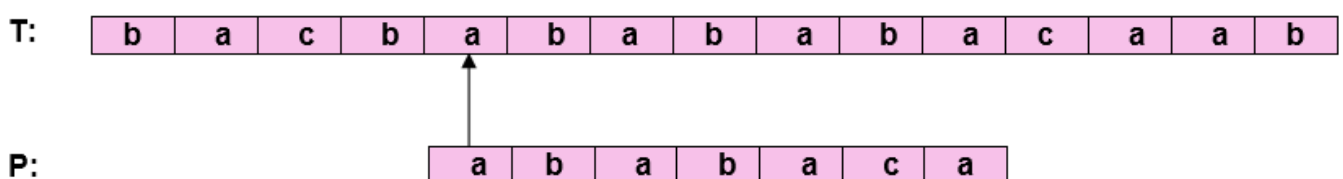
Backtracking on p, Comparing P [1] and T [3]

**Step4:**  $i = 4, q = 0$ 

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

**Step5:**  $i = 5, q = 0$ 

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



Step 6:  $i = 0$  to  $n - 1$

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.

← Prev

Next →



 [For Videos Join Our Youtube Channel: Join Now](#)


## Feedback


- Send your Feedback to [feedback@javatpoint.com](mailto:feedback@javatpoint.com)

## Help Others, Please Share





## Learn Latest Tutorials


 [Splunk tutorial](#)  
Splunk


 [SPSS tutorial](#)  
SPSS


 [Swagger tutorial](#)  
Swagger

 [T-SQL tutorial](#)  
Transact-SQL

 [Tumblr tutorial](#)  
Tumblr


 [React tutorial](#)  
ReactJS

 [Regex tutorial](#)  
Regex

 [Reinforcement learning tutorial](#)  
Reinforcement Learning


 [R Programming tutorial](#)  
R Programming

 [RxJS tutorial](#)  
RxJS


 [React Native tutorial](#)  
React Native

 [Python Design Patterns](#)

Python Design  
Patterns


 Python Pillow  
tutorial  
Python Pillow


 Python Turtle  
tutorial  
Python Turtle

 Keras tutorial  
Keras

## Preparation

 Aptitude  
Aptitude

 Logical  
Reasoning  
Reasoning

 Verbal Ability  
Verbal Ability

 Interview  
Questions  
Interview Questions


 Company  
Interview  
Questions  
Company Questions


## Trending Technologies

 Artificial  
Intelligence  
Artificial  
Intelligence


 AWS Tutorial  
AWS

 Selenium  
tutorial  
Selenium

 Cloud  
Computing  
Cloud Computing

 Hadoop tutorial  
Hadoop


 ReactJS  
Tutorial  
ReactJS

 Data Science  
Tutorial  
Data Science

 Angular 7  
Tutorial  
Angular 7

 Blockchain  
Tutorial  
Blockchain


 Git Tutorial  
Git


 Machine  
Learning Tutorial  
Machine Learning


 DevOps  
Tutorial  
DevOps


## B.Tech / MCA

 DBMS tutorial  
DBMS


 Data Structures  
tutorial  
Data Structures


 DAA tutorial  
DAA


 Operating  
System  
Operating System


 Computer  
Network tutorial  
Computer Network


 Compiler  
Design tutorial  
Compiler Design


 Computer  
Organization and  
Architecture  
Computer  
Organization

 Discrete  
Mathematics  
Tutorial  
Discrete  
Mathematics

 Ethical Hacking  
Ethical Hacking


 Computer  
Graphics Tutorial  
Computer Graphics


 Software  
Engineering  
Software  
Engineering


 html tutorial  
Web Technology


 Cyber Security  
tutorial  
Cyber Security


 Automata  
Tutorial  
Automata


 C Language  
tutorial  
C Programming


 C++ tutorial  
C++

 Java tutorial  
Java

 .Net  
Framework  
tutorial  
.Net

 Python tutorial  
Python

 List of  
Programs  
Programs

 Control  
Systems tutorial  
Control System

 Data Mining  
Tutorial  
Data Mining

 Data  
Warehouse  
Tutorial  
Data Warehouse

