Knuth-Morris-Pratt (KMP)Algorithm

TEAM MEMBERS:

ABDULLAH

MUSTUFA

ABDUL QUDOOS

STRING SEARCH ALGORITHM

string-searching algorithms called string-matching algorithms

an important class of String algorithm

try to find a place where one or several Strings(also called patterns) are found within a larger string or text

INTRODUCTION

- Knuth-Morris and Pratt introduce
- solution to the string search problem
- keeps a track of the comparison of characters between main text and pattern,
- Pattern-text

INTRODUCTION

The implementation of Knuth-Morris-Pratt algorithm is efficient because it minimizes the total number of comparisons of the pattern against the input string.

The running time of the KMP algorithm is optimal (O(m + n)), which is very fast.

INTRODUCTION

- Doesn’t work so well as the size of the alphabets increases. By which more chances of mismatch occurs.

APPLICATIONS

- Checking for Plagiarism in documents etc
- Bioinformatics and DNA sequencing
- Digital libraries
- Spelling checkers

APPLICATIONS

Spam	Spam filters
Search	Search engines, or for searching content in large databases
Word	Word processors

COMPONENTS OF KMP ALGORITHM

Prefix Function

KMP Matcher

Prefix Function

PREFIX FUNCTION

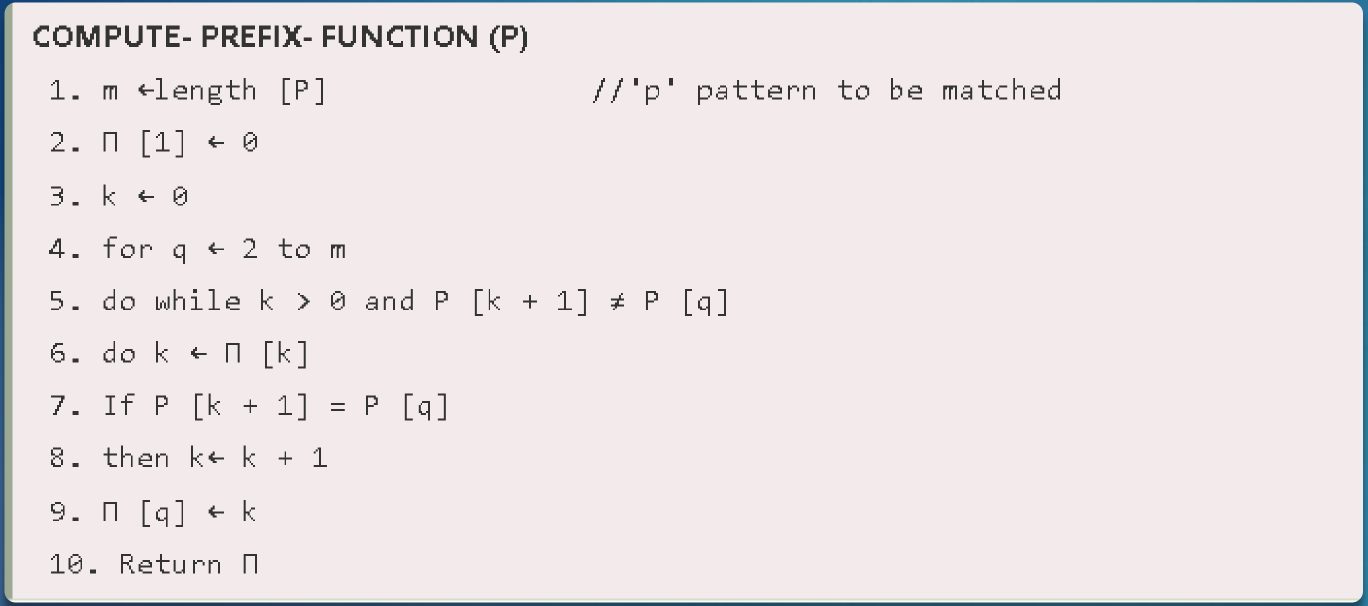
THE PREFIX FUNCTION (Π):

Π FOR A PATTERN ENCAPSULATES KNOWLEDGE

HOW THE PATTERN MATCHES AGAINST THE SHIFT OF ITSELF.

THIS INFORMATION USED TO AVOID A USELESS SHIFT OF THE PATTERN 'P.'

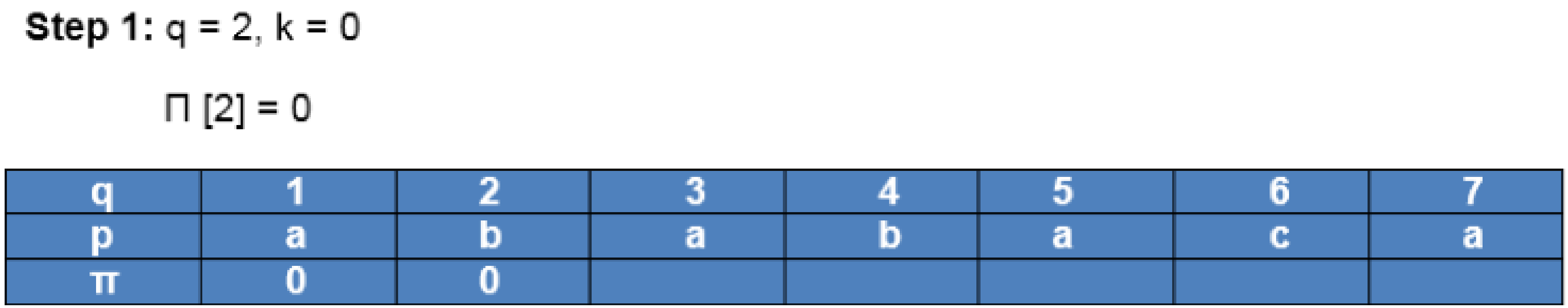
PREFIX FUNCTION



COMPUTE- PREFIX- FUNCTION (P)

```
m ←length [P]                    //'p' pattern to be matched
Π [1] ← 0
k ← 0
for q ← 2 to m
    do while k > 0 and P [k + 1] ≠ P [q]
        do k ← Π [k]
    If P [k + 1] = P [q]
        then k← k + 1
    Π [q] ← k
Return Π
```

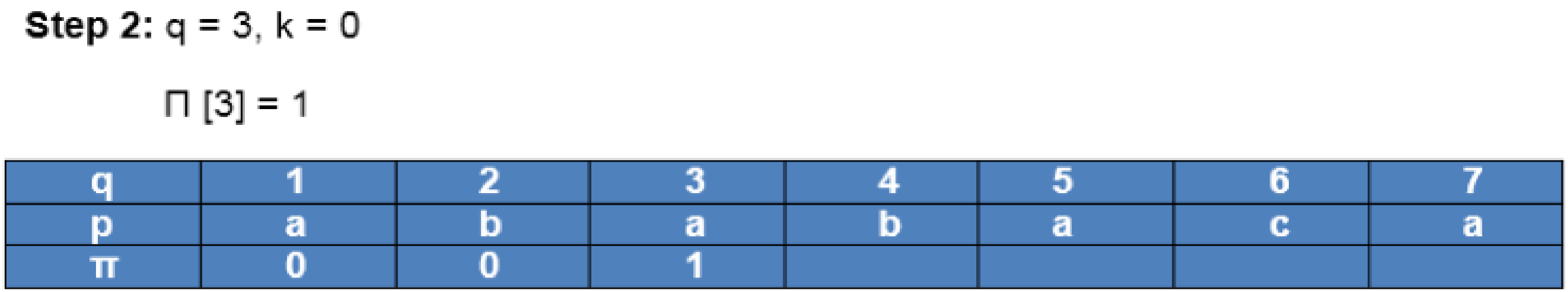
PREFIX FUNCTION



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					

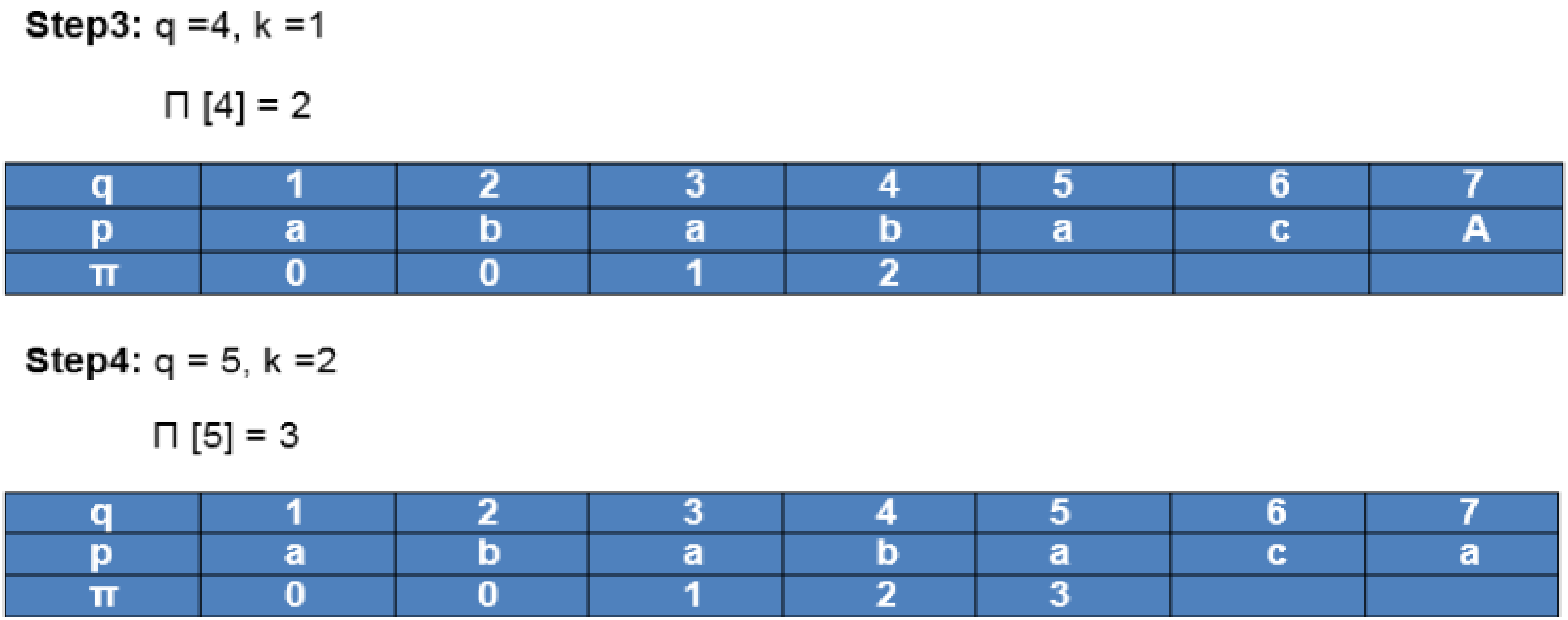


Step 2: q = 3, k = 0

Π [3] = 1

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

PREFIX FUNCTION



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

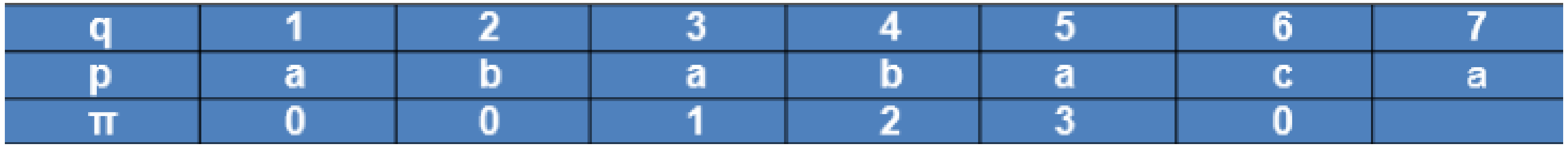
Π [5] = 3

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

PREFIX FUNCTION

Step5: q = 6, k = 3

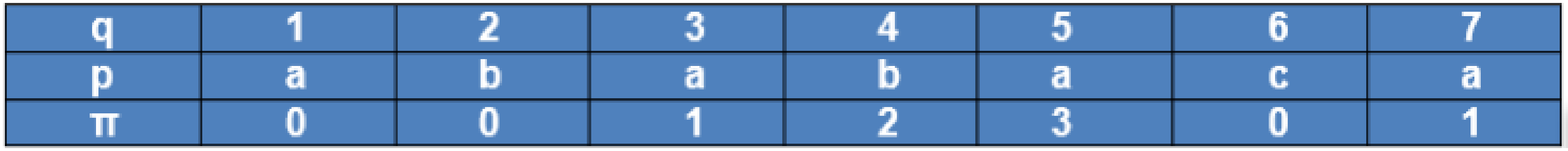
Π [6] = 0



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

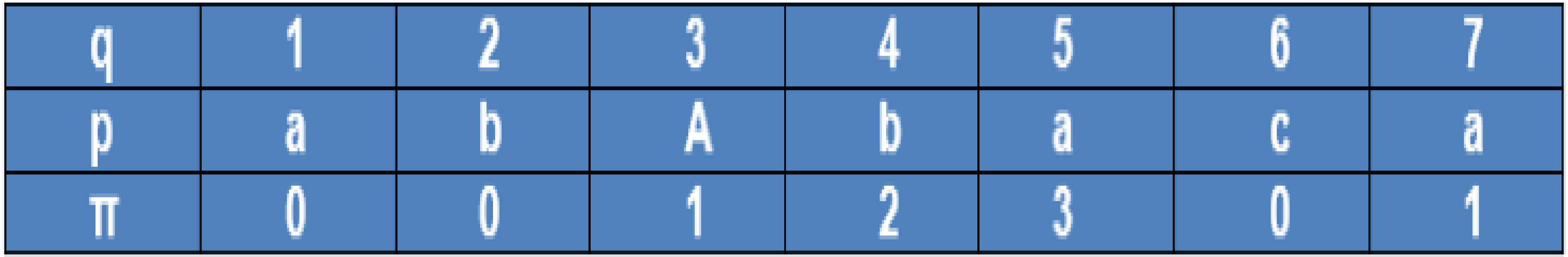
Π [7] = 1



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

PREFIX FUNCTION

- Computation Complete

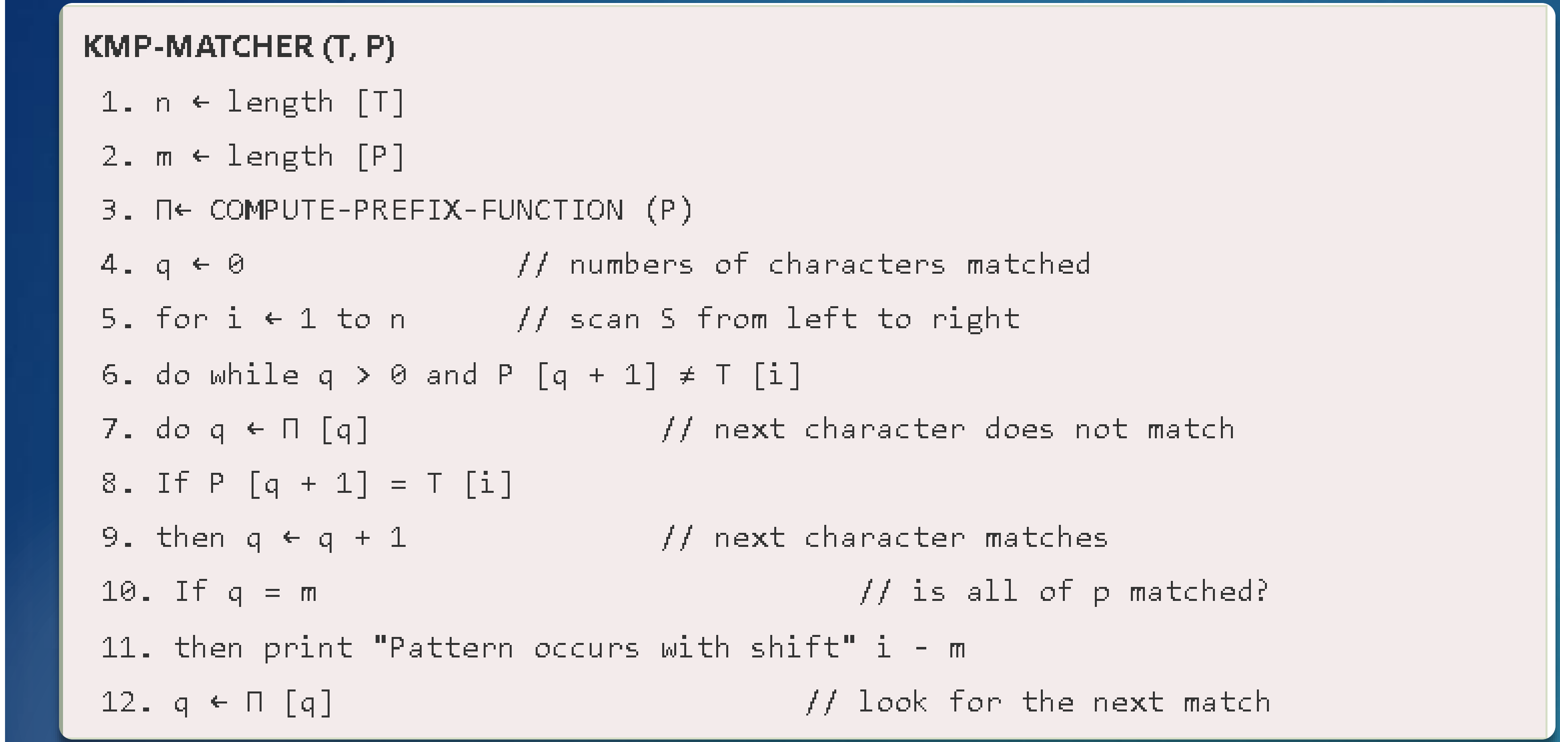


q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

KMP MATCHER

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



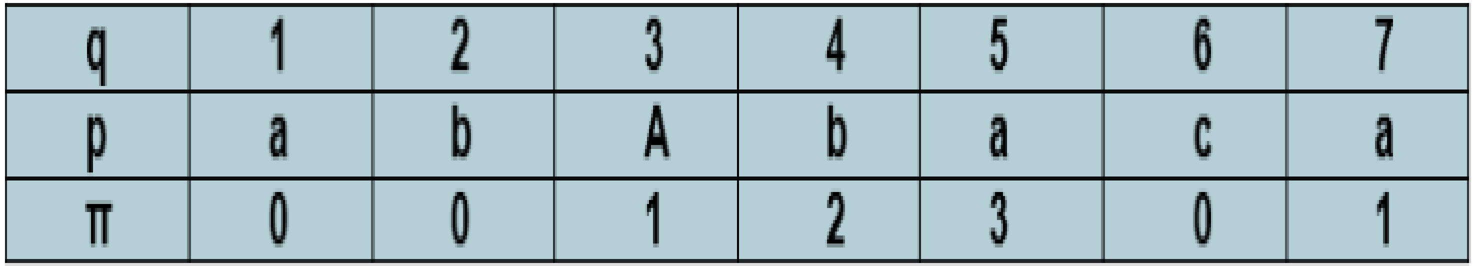
KMP-MATCHER (T, P)

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

KMP MATCHER

KMP MATCHER

- 'p' the prefix function was computed previously and is as following;

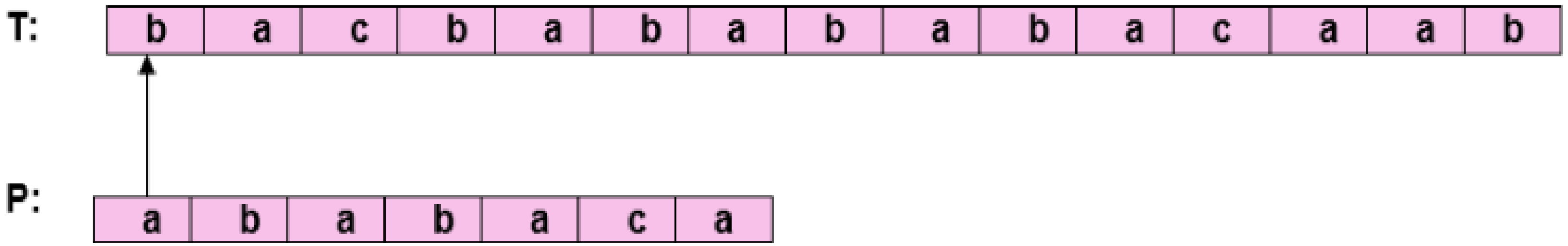


q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

KMP MATCHER

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

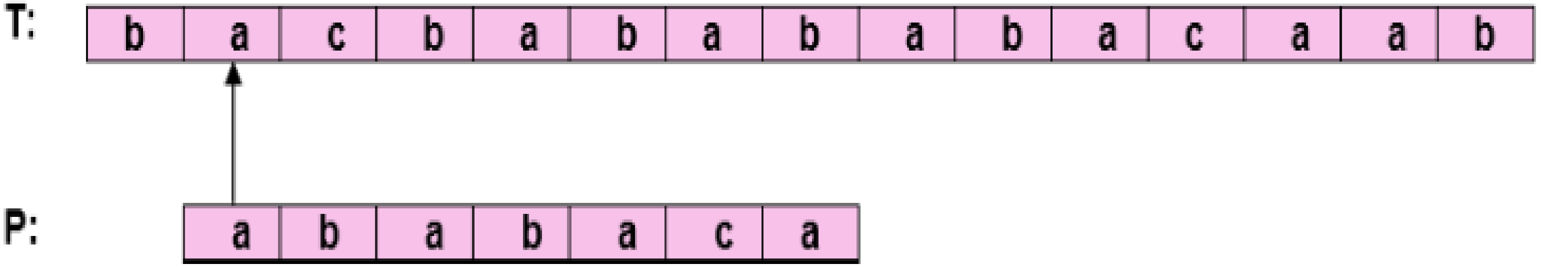
P:	a	b	a	b	a	c	a

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

KMP MATCHER

Step2: i = 2, q = 0

Comparing P [1] with T [2]



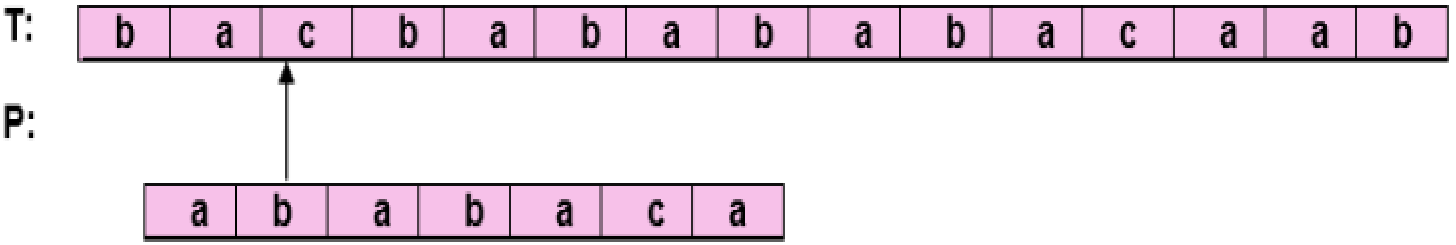
T:	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b
P:		a	b	a	b	a	c	a							

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

KMP MATCHER

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

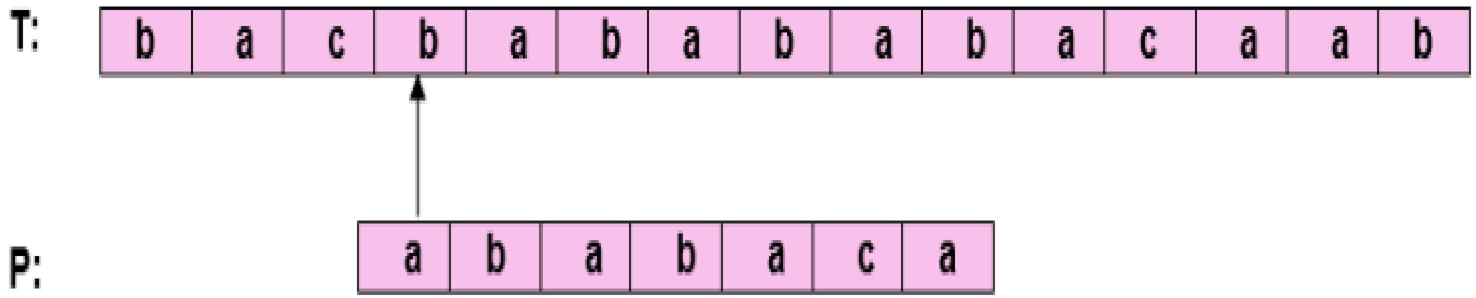
P: a b a b a c a

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

KMP MATCHER

Step4: i = 4, q = 0

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



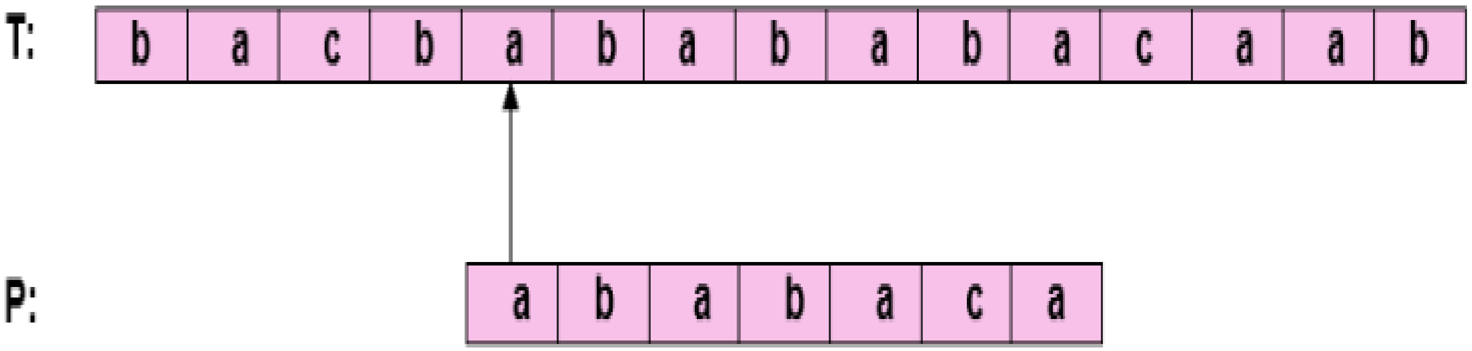
T: b a c b a b a b a b a c a a b

P: a b a b a c a

KMP MATCHER

Step5: i = 5, q = 0

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



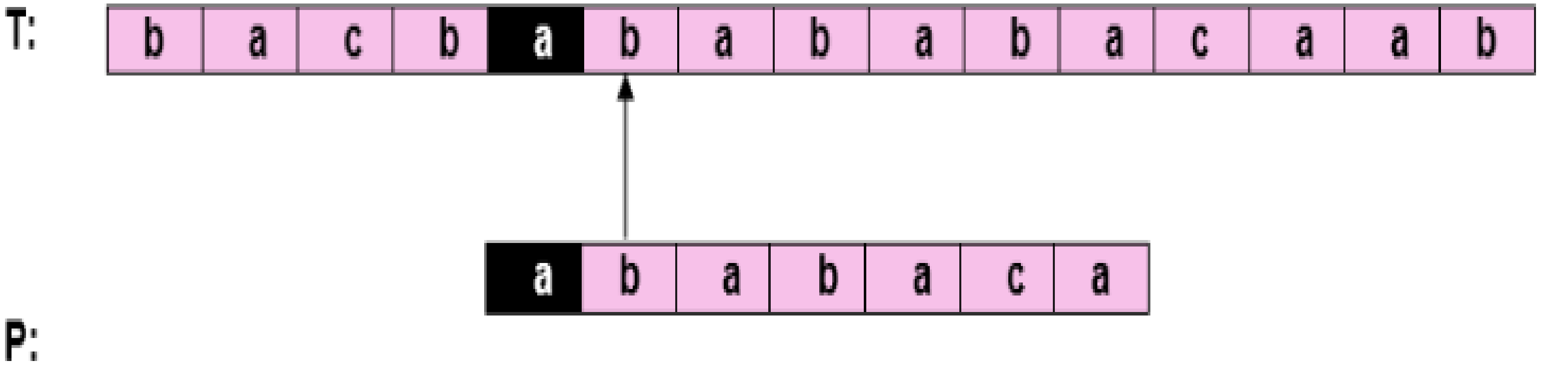
T: b a c b a b a b a b a c a a b

P: a b a b a c a

KMP MATCHER

Step6: i = 6, q = 1

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



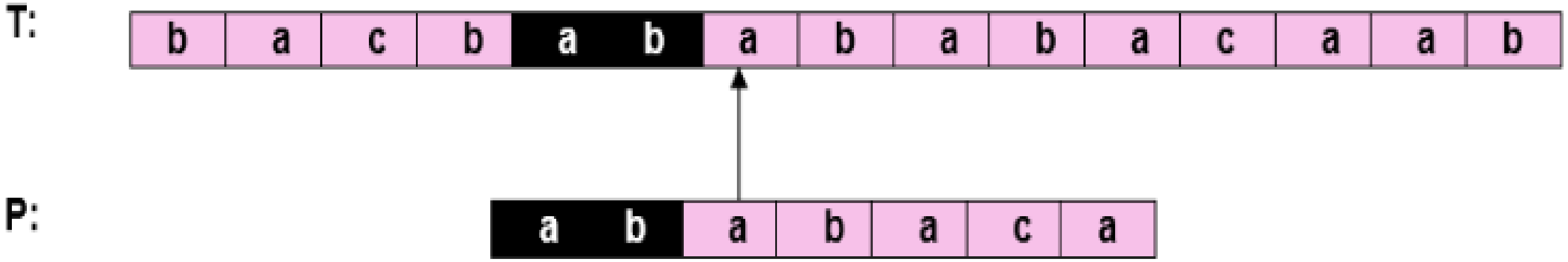
T: b a c b a b a b a b a c a a b

P: a b a b a c a

KMP MATCHER

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

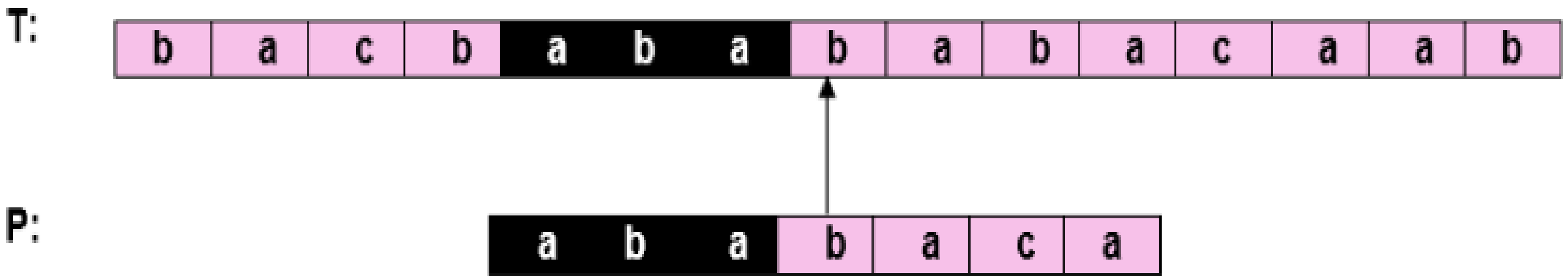
P: a b a b a c a

KMP MATCHER

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

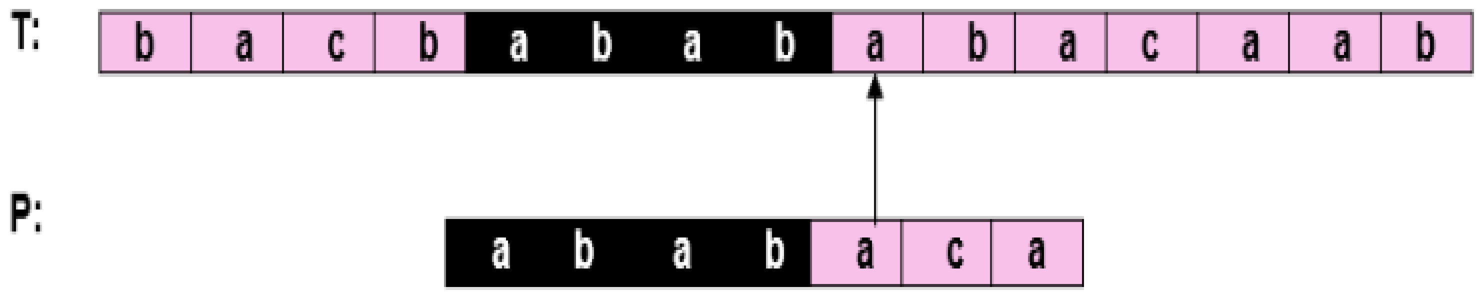
P: a b a b a c a

KMP MATCHER

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

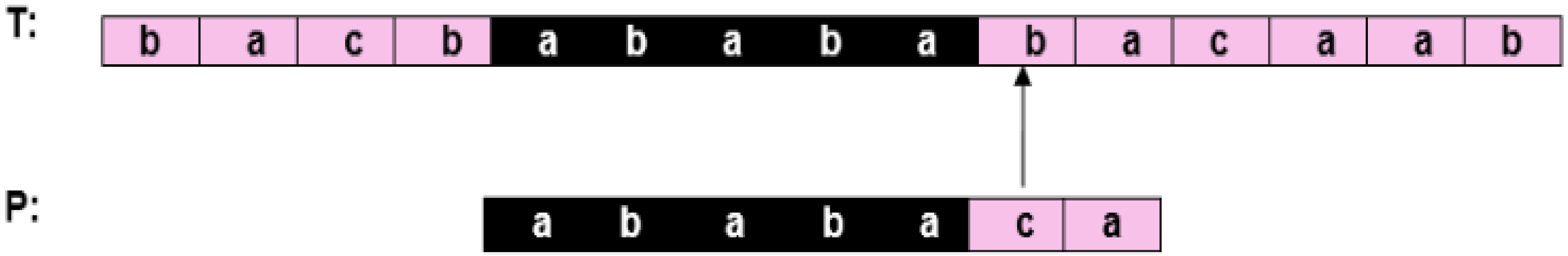
P: a b a b a c a

KMP MATCHER

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

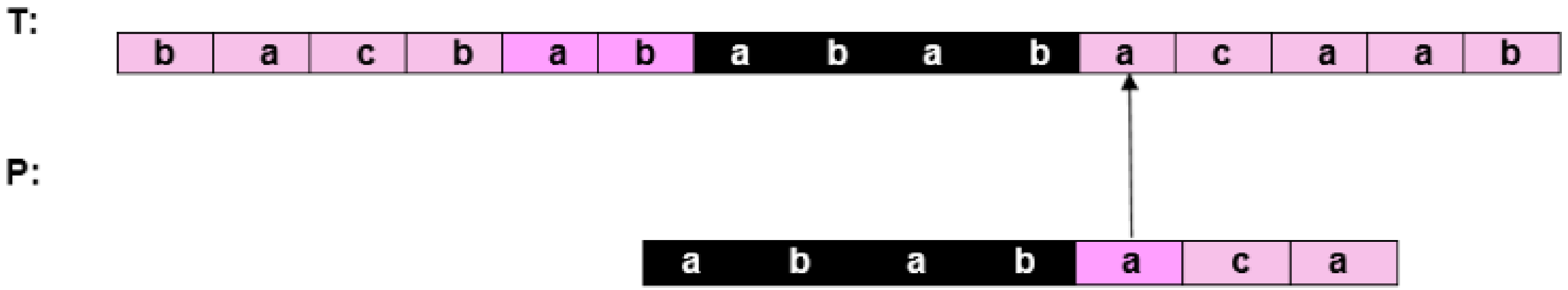
Backtracking on p, Comparing P [4] with T [10] because after mismatch q = π [5] = 3

KMP MATCHER

Step11: i = 11, q =4

Comparing P [5] with T [11]

P [5] match with T [11]



T:

b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P:

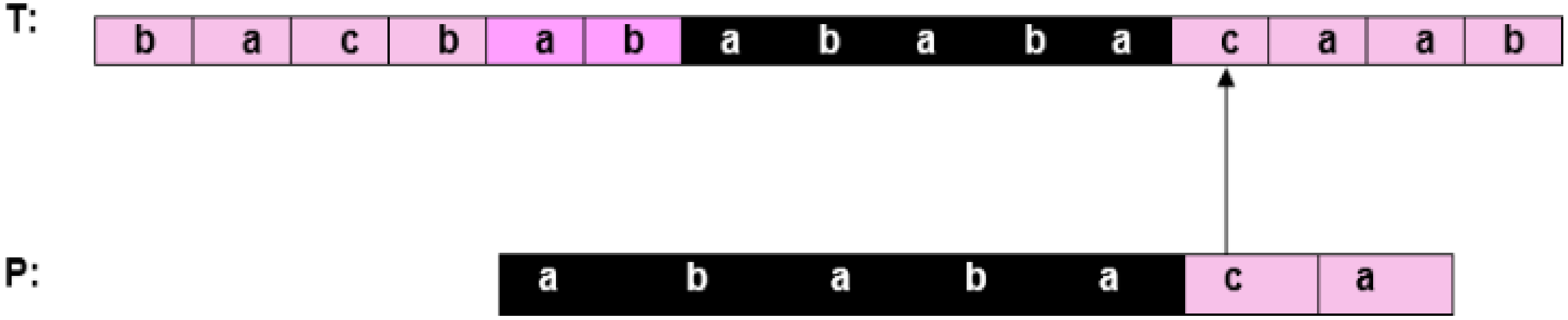
a	b	a	b	a	c	a

KMP MATCHER

Step12: i = 12, q = 5

Comparing P [6] with T [12]

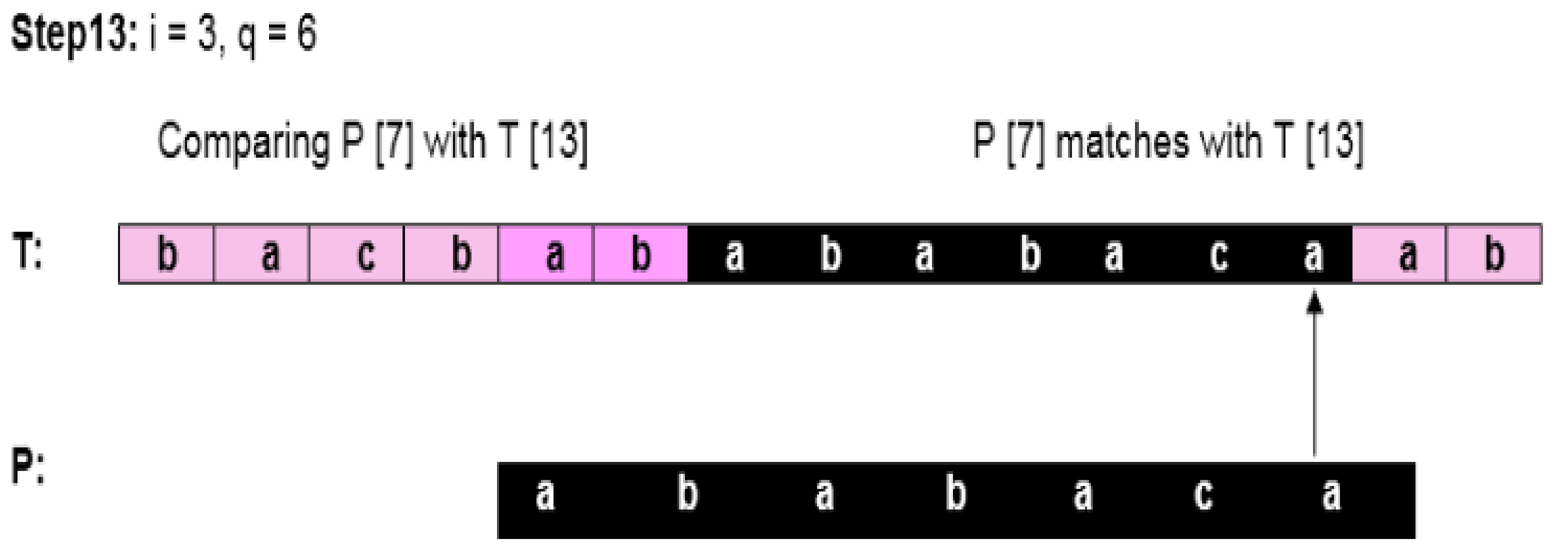
P [6] matches with T [12]



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

P: a b a b a c a

KMP MATCHER



Step13: i = 3, q = 6

Comparing P [7] with T [13]

P [7] matches with T [13]

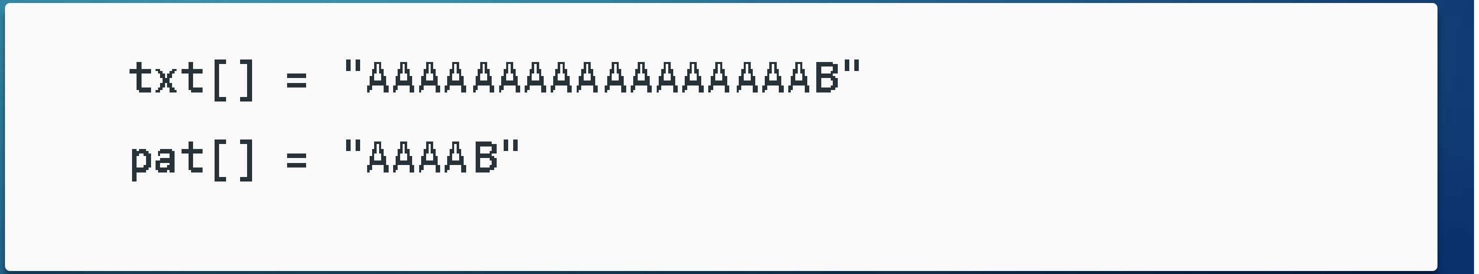
T: b a c b a b a b a b a c a a b

P: a b a b a c a

- 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

TIME COMPLEXITY

- **Time Complexity**
- The time complexity of KMP algorithm is O(n) in the worst case.



```
txt[] = "AAAAAAAAAAAAAAAAAB"
pat[] = "AAAAB"
```

THE END