Learn DAA : From B K Sharma

TCS-503: Design and Analysis of Algorithms

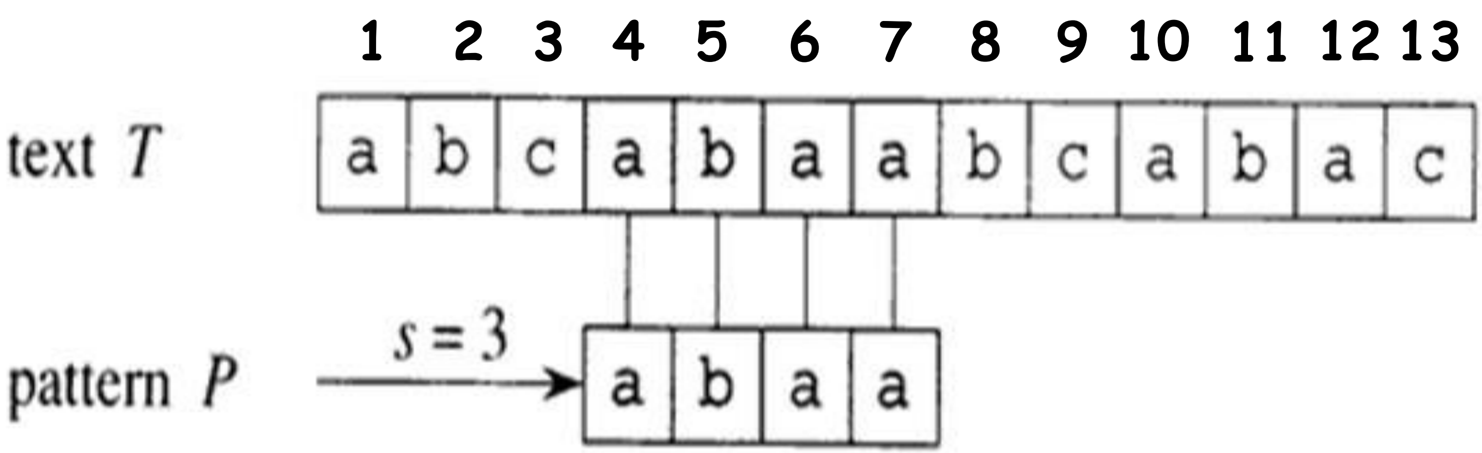
Unit V: Selected Topics: String Matching

Unit V

- Selected Topics:
  - NP Completeness
  - Approximation Algorithms
  - Randomized Algorithms
  - String Matching

What ?

- **Input**
  - a string T -- the text
  - a string P -- the pattern
- **Output**
  - all the occurrences of *P* in T.



The pattern P occurs only once in the text T, at shift s=3

Illustration

	1	2	3	4	5	6	7	8	9	0	1	2	3
T =	t	a	t	a	t	t	a	t	a	t	a	t	a
P =	t	a	t	a									

- Output

  1

  6

  8

  10

Why?

In Computer Science:

Dictionary, database

Search engines: Yahoo!, Google, …

How ?

- There are many exact string matching algorithms. Nearly all of them are concerned with how to slide the pattern.
- The naïve string matching Algorithm
- Backward Algorithm
- Boyer and Moore Algorithm
- Colussi Algorithm
- Crochemore and Perrin Algorithm
- Galil Gianardo Algorithm

How ?

- Galil and Seiferas Algorithm
- Horsepool Algorithm
- Knuth Morris and Pratt Algorithm
- KMP Skip Algorithm
- Max-Suffix Matching Algorithm
- Morris and Pratt Algorithm
- Quick Searching Algorithm

How ?

- The Rabin-Karp Algorithm
- Raita Algorithm
- Reverse Factor Algorithm
- Reverse Colussi Algorithm
- Self Max-Suffix Algorithm
- Simon Algorithm

We will learn…….

- The naïve string matching Algorithm
- The Knuth-Morris-Pratt Algorithm
- The Rabin-Karp Algorithm
- The Boyer and Moore Algorithm

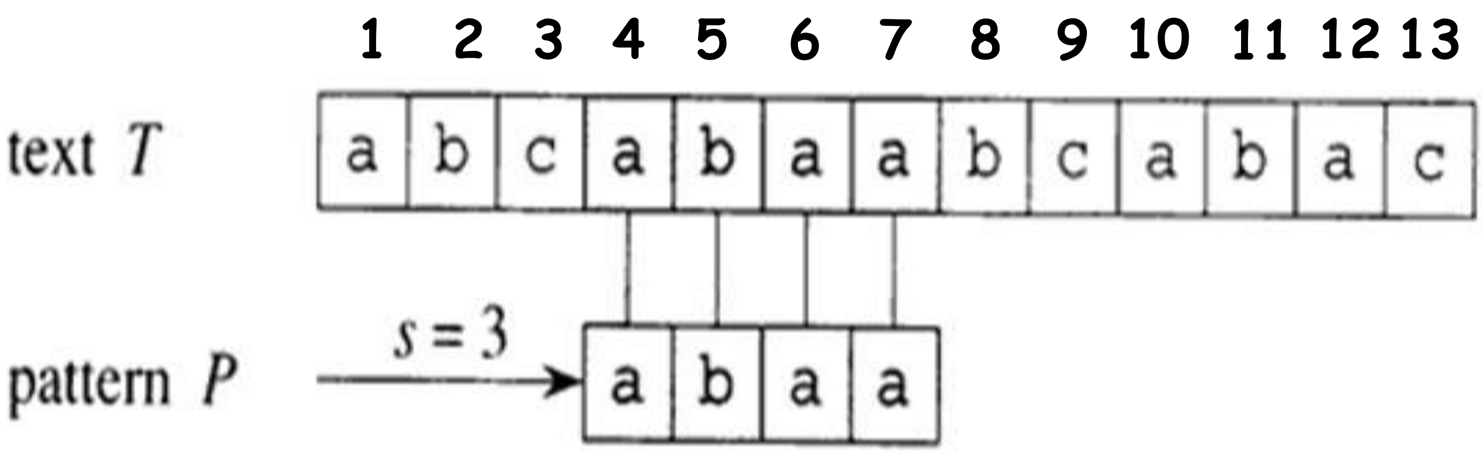
String Matching

Let

the text be an array T[1….n] of length n,

the pattern is an array P[1….m] of length m≤ n and

0 ≤ s ≤ n – m



If P occurs with shift *s* in T , the we call s a valid shift, otherwise, we call s an invalid shift.

String Matching Problem

Formal Definition

Find all valid shifts with which a given P occurs in a given text T.

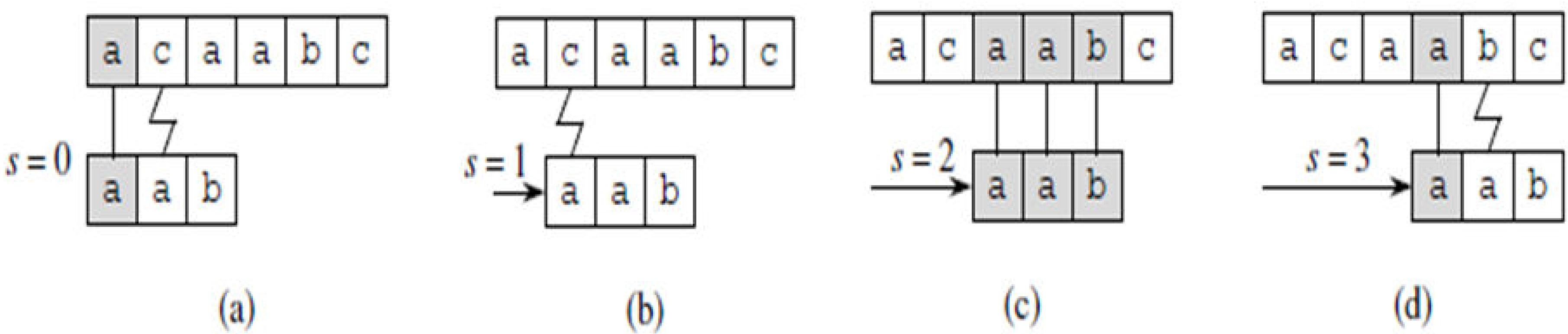
The naïve String Matching Algorithms (Brute- Force Algorithm): $O((mn)$

Alg. NAÏVE-STRING-MATCHER(T,P)

1. $n \leftarrow$ length[T]
2. $m \leftarrow$ length[P]
3. For $s \leftarrow 0$ to n-m do
4. If P[1…..m]=T[s+1,s=2,……,s+m] then
5. Print "Pattern occurs with shift" s

The naïve algorithm finds all valid shifts using a loop that checks the condition P[1…..m]=T[s+1,s=2,……,s+m] for each of the n-m+1 possible values of s.

The Operation of Naïve- String Matching Algorithms
(Brute- Force Algorithm)

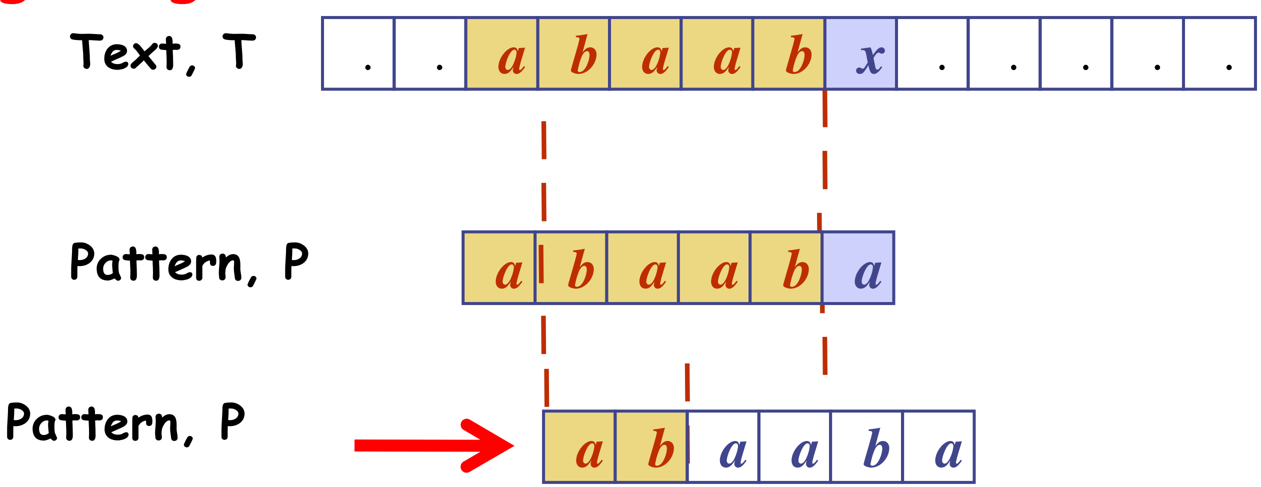


P[1…3]=T[1…3]: No P[1…3]=T[2…4]: No P[1…3]=T[3…5]: Yes P[1…3]=T[4…6]: No

Pattern occurs with shift 2

The Problem with Naïve- String Matching Algorithms (Brute- Force Algorithm)

Whenever a character mismatch occurs after matching of several characters, the comparison begins by going back in T from the character which follows the last beginning character.



Question: Can we do better: not going back in T?

Answer: Yes, we can, using The Knuth Morris and Pratt (KMP) Algorithm.

Comments

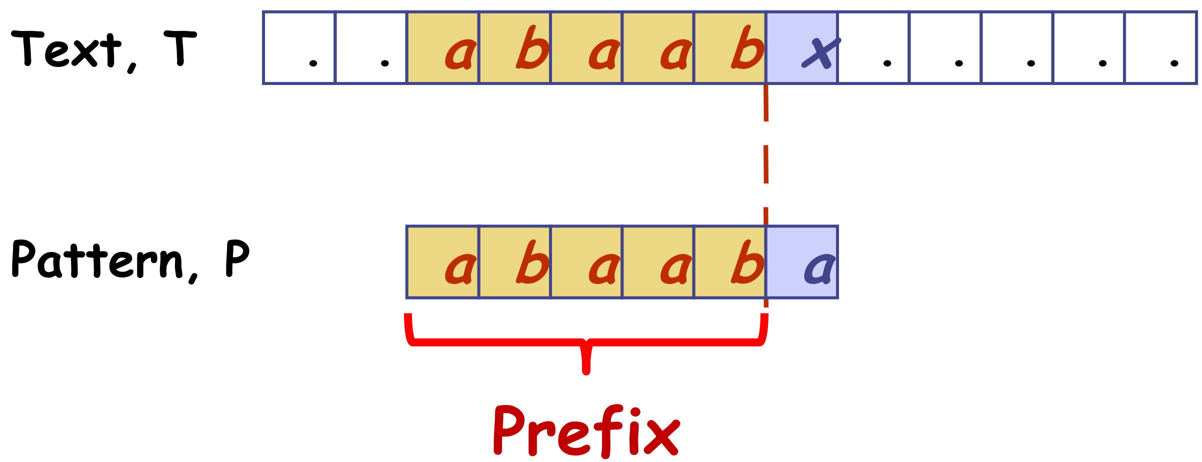
Mismatch is a signal for "jumping".

How far to jump?

Determinable merely from *P*.

The Knuth Morris and Pratt (KMP) Algorithm

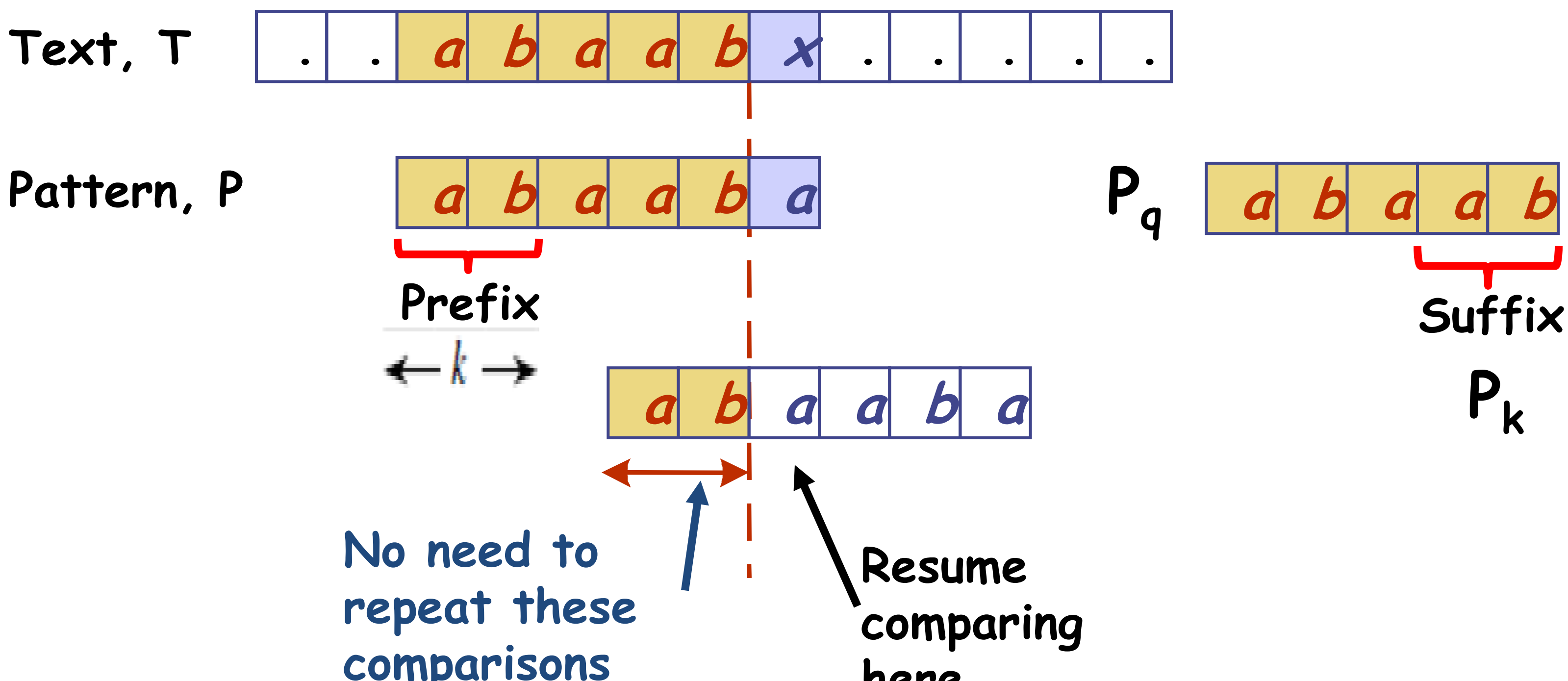
Idea: The matched characters in T are, in fact, a prefix of P, so just from P, it is OK to determine whether a shift is invalid or not. That is, T is irrelevant.



The Knuth Morris and Pratt (KMP) Algorithm

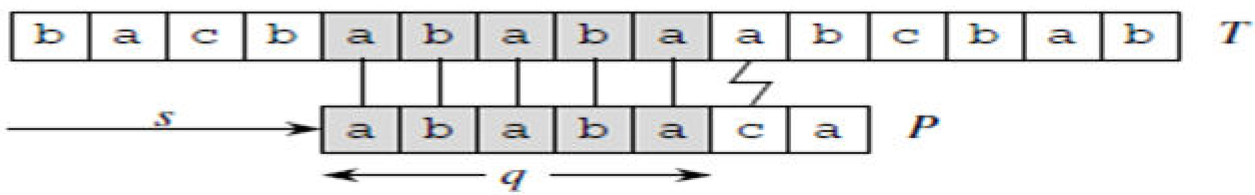
Idea: The matched q characters allows us to determine immediately that certain shifts are invalid.

So directly go to the shift which is potentially valid.



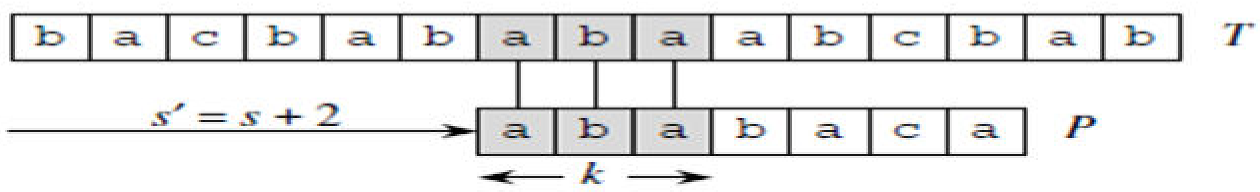
The Knuth Morris and Pratt (KMP) Algorithm

Given that q characters have matched successfully at shift s, then how much to jump?

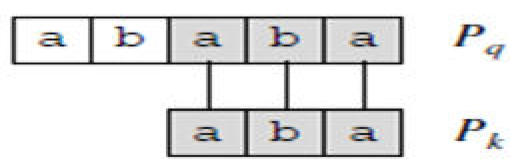


(a)

Answer: Longest prefix of P, which is a proper suffix of p_q



(b)



(c)

The Knuth Morris and Pratt (KMP) Algorithm

Answer: Longest prefix of P which a proper suffix of p_q ,which is equal to 3.

This information is pre-computed and stored in an array π, so that $\pi[5]=3$.

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

The Knuth Morris and Pratt (KMP) Algorithm

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

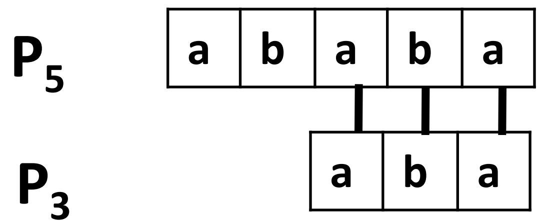
What is the meaning of $\pi[5]=3$?

$\pi[5]=3$ means that after a mismatch at index 5 of pattern, we next compare index 3 of pattern.

P_5	a	b	a	b	a
P_3			a	b	a

$\pi[5]=3$

The Knuth Morris and Pratt (KMP) Algorithm



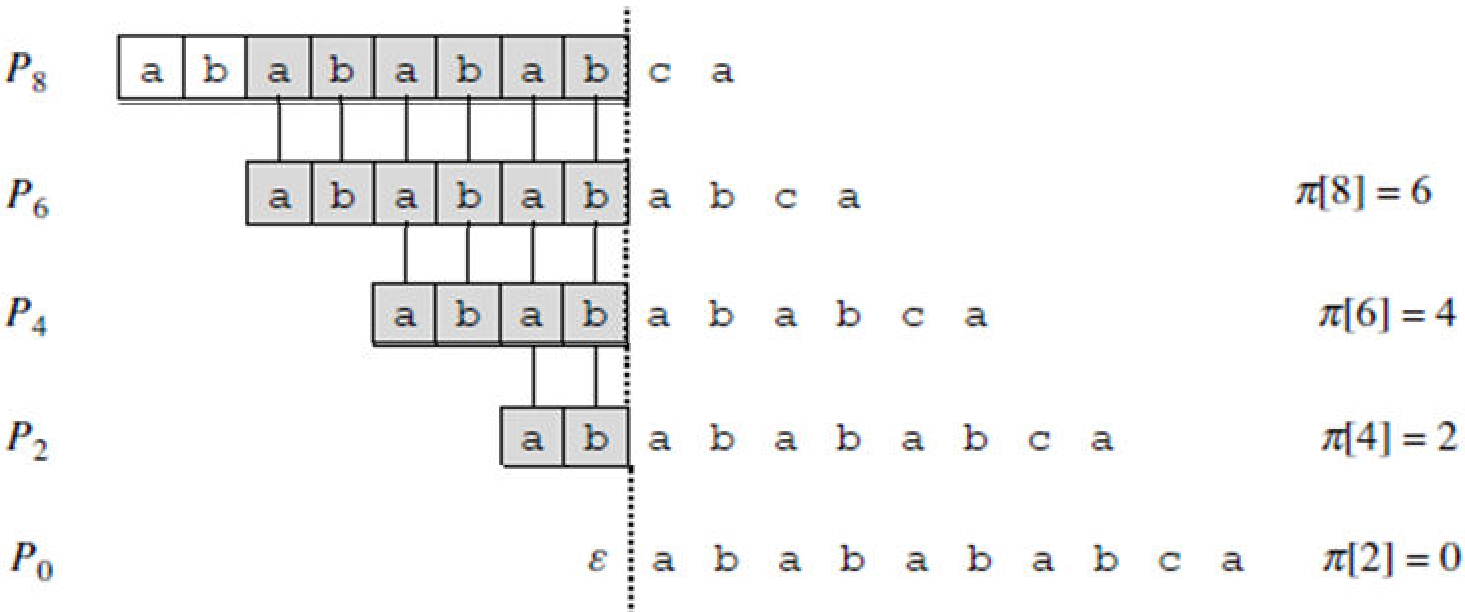
$\pi[5]=3$

$\pi[5]=3$ means that after a mismatch at index 5 of pattern, the next valid shift will be 5-3=2

The Knuth Morris and Pratt (KMP) Algorithm

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

(a)

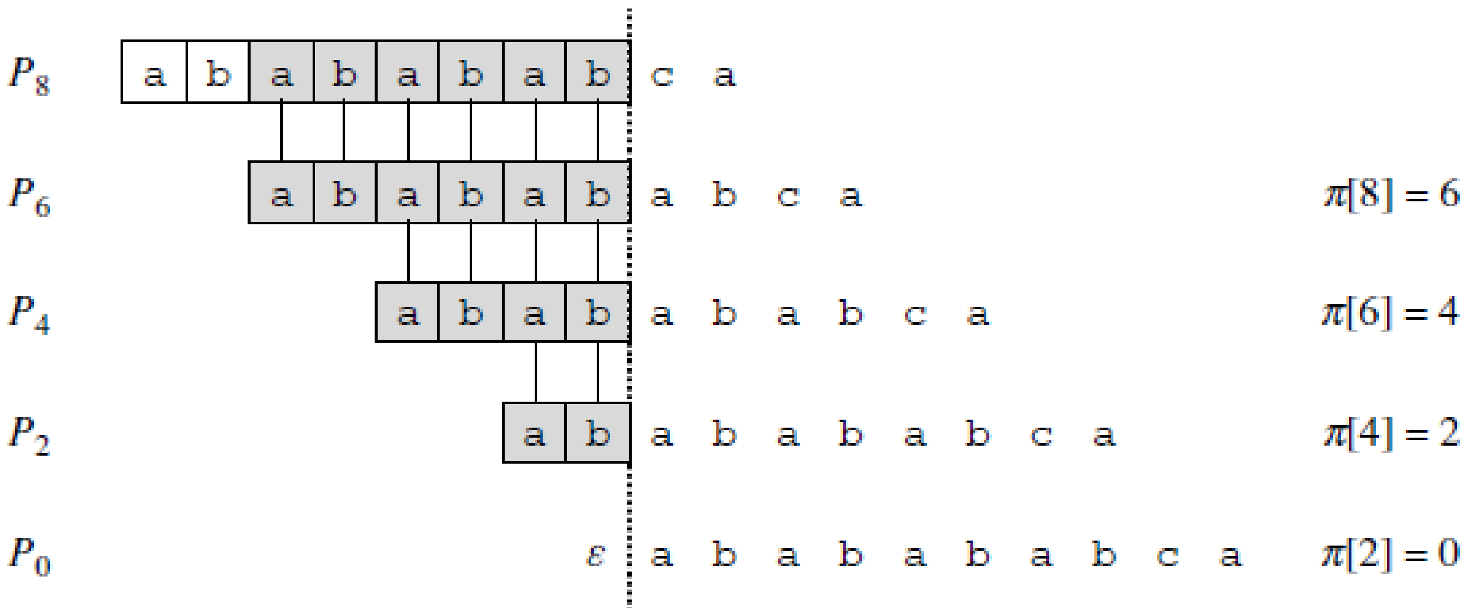


(b)

The Knuth Morris and Pratt (KMP) Algorithm

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

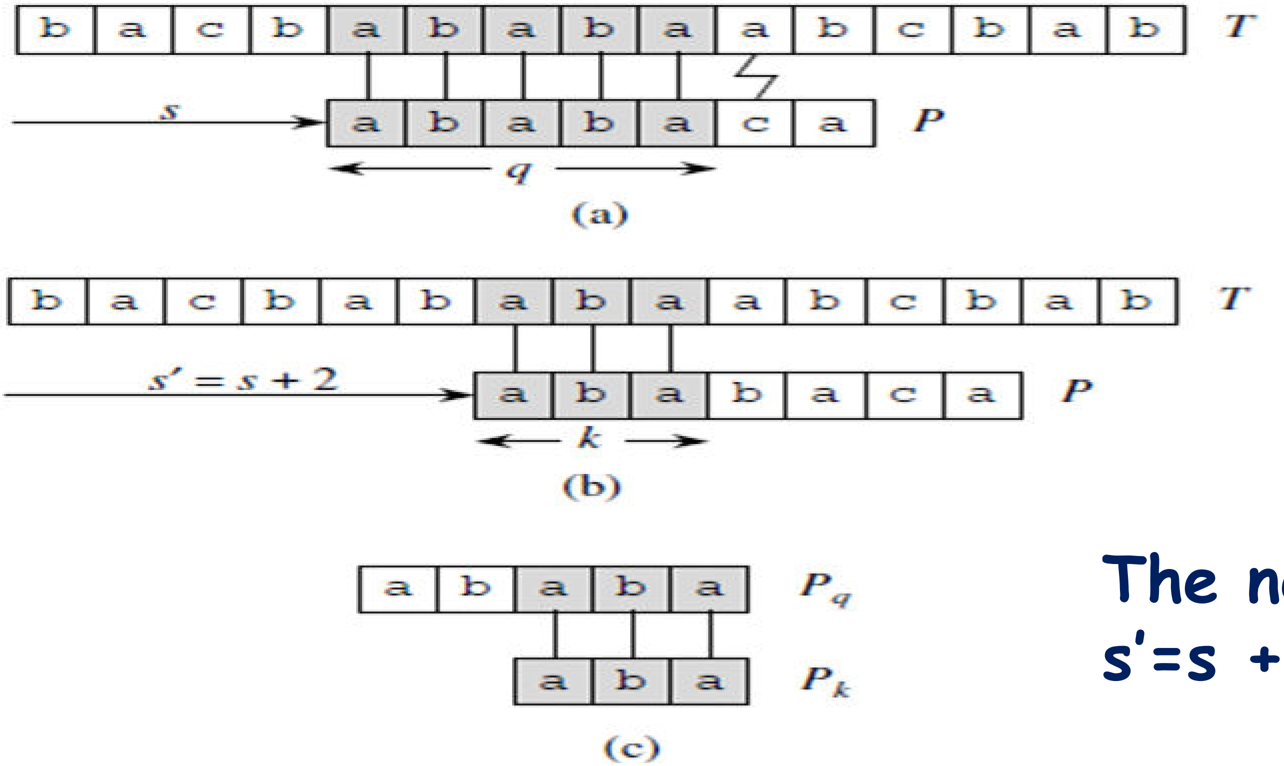
(a)



(b)

Learn DAA : From B K Sharma

The Knuth Morris and Pratt (KMP) Algorithm



The next valid shift is $s' = s + (q - \pi[q])$.

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

Learn DAA : From B K Sharma

The Knuth Morris and Pratt (KMP) Algorithm

Complexity

An inspiring example

Answer: Longest prefix of P which a proper suffix of p_7 is 3

S = x a b x y a b x y a b x z

P = a b x y a b x z

s=1 a b x y a b x z

P_7: 7 Characters have matched then mismatch occurs then how much to jump?

a b x y a b x z

Will not match:
Redundant Comparisons

a b x y a b x z

a b x y a b x z

s=5 a b x y a b x z

$\pi[q]= \pi[7]=3$

Given that q characters have matched successfully at shift s, the next valid shift is $s'=s + (q- \pi[q]) =1+(7-3)=5$

Rabin-Karp Algorithm

Calculates a hash value for the M-character long pattern, and for each M-character subsequence of text to be compared.

If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence.

If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence.

Rabin-Karp Algorithm

In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

Rabin-Karp Algorithm

pattern is M characters long.

HASH_P=hash value of pattern.

HASH_T=hash value of first M letters in body of text.

1. do
2. if (HASH_P == HASH_T)
3. brute force comparison of pattern and selected section of text.
4. HASH_T= hash value of next section of text, one character over.
5. while (end of text)

Rabin-Karp Algorithm

Example

Given T = 31415926535 and P = 26

We choose q = 11, a prime number

P mod q = 26 mod 11 = 4

3	1	4	1	5	9	2	6	5	3	5

T[1,2] mod q =31 mod 11 = 9. not equal to 4

3	1	4	1	5	9	2	6	5	3	5

T[2,3]mod q=14 mod 11 = 3, not equal to 4

3	1	4	1	5	9	2	6	5	3	5

T[3,4]mod q=41 mod 11 = 8, not equal to 4

Rabin-Karp Algorithm

Example

3	1	4	1	5	9	2	6	5	3	5

T[4,5]mod 11=15 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

T[5,6] mod 11=59 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

92 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

26 mod 11 = 4 equal to 4 -> an exact match!!

3	1	4	1	5	9	2	6	5	3	5

65 mod 11 = 10 not equal to 4

Rabin-Karp Algorithm

Example

3	1	4	1	5	9	2	6	5	3	5

53 mod 11 = 9 not equal to 4

3	1	4	1	5	9	2	6	5	3	5

35 mod 11 = 2 not equal to 4

Boyer Moore Algorithm

The most efficient string-matching algorithm in usual applications.

Based on Three Rules:

The first is Bad-Symbol Shift Rule:

(Based on symbols that caused mismatch)

The second is called Good Suffix Shift Rule:

(Based on symbols that caused match)

Third is Alignment and Comparison Rule:

Align Pattern with the Text from left to right and

Compare text with the pattern from right to left

Boyer Moore Algorithm

What is bad symbol shift Rule?

If we mismatch, use knowledge of the mismatched text character to skip alignments:

This is called Bad symbol shift Rule.

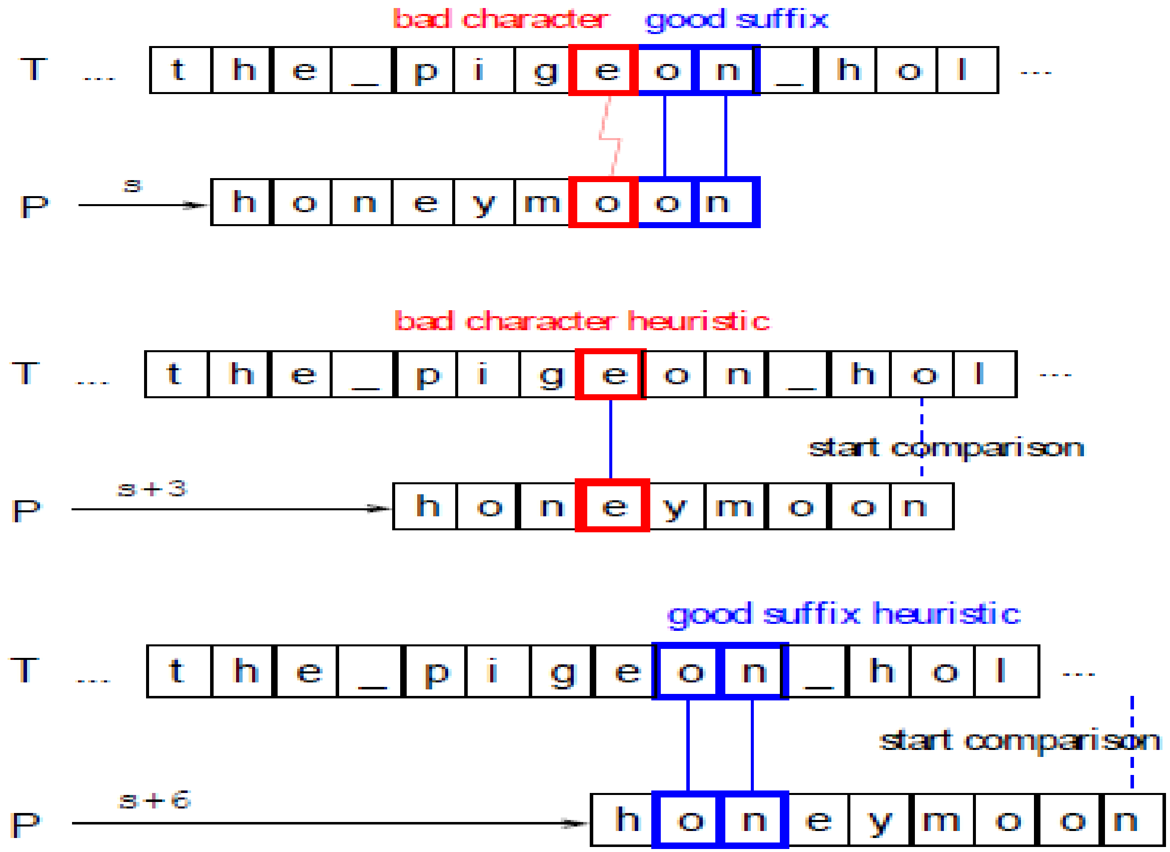
What is Good symbol shift Rule?

If we match some characters, use knowledge of the matched characters to skip alignments:

This is called Good Suffix Rule.

Boyer Moore Algorithm

The BM algorithm takes the larger shift amount computed by Bad Symbol Rule and Good Suffix Rule.



Boyer Moore Algorithm

Bad Symbol Shift Rule: Case 1

Example

	1	2	3	4	5	6	7	8	9	10	11	12
T:	a	b	b	a	**d**	a	b	a	c	b	a	b
P:	b	a	b	a	c							

First mismatch occurs at T[5]≠P[5].

Since T[5]=d does not occur in P at all, can shift P[1] to P[6].

	1	2	3	4	5	6	7	8	9	10	11	12
T:	a	b	b	a	**d**	a	b	a	c	b	a	b
P:	b	a	b	a	c							
P:						b	a	b	a	c		

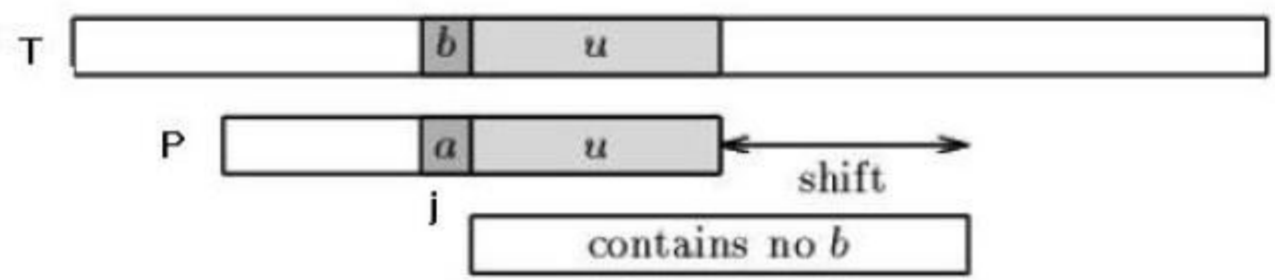
Boyer Moore Algorithm

Bad Symbol Shift Rule: Case 1

In general:

If there is a mismatch between P[j] and T[i] and T[i] does not appear in P,

P should be advanced by j



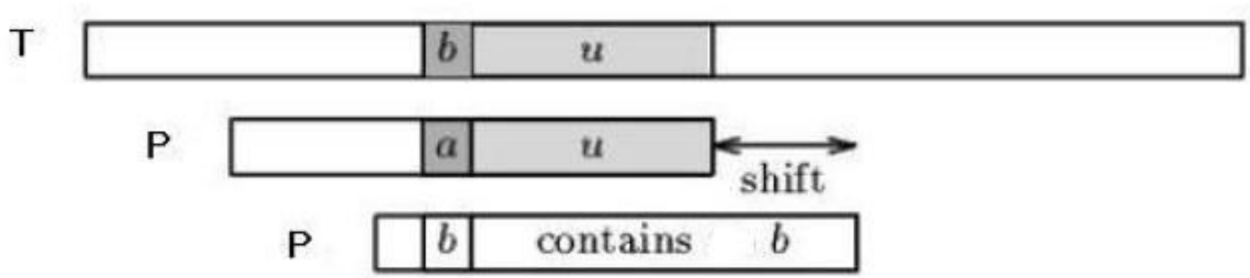
Boyer Moore Algorithm

Bad Symbol Shift Rule: Case 2

If there is a mismatch between P[j] and T[i] and

if T[i] appears in P,

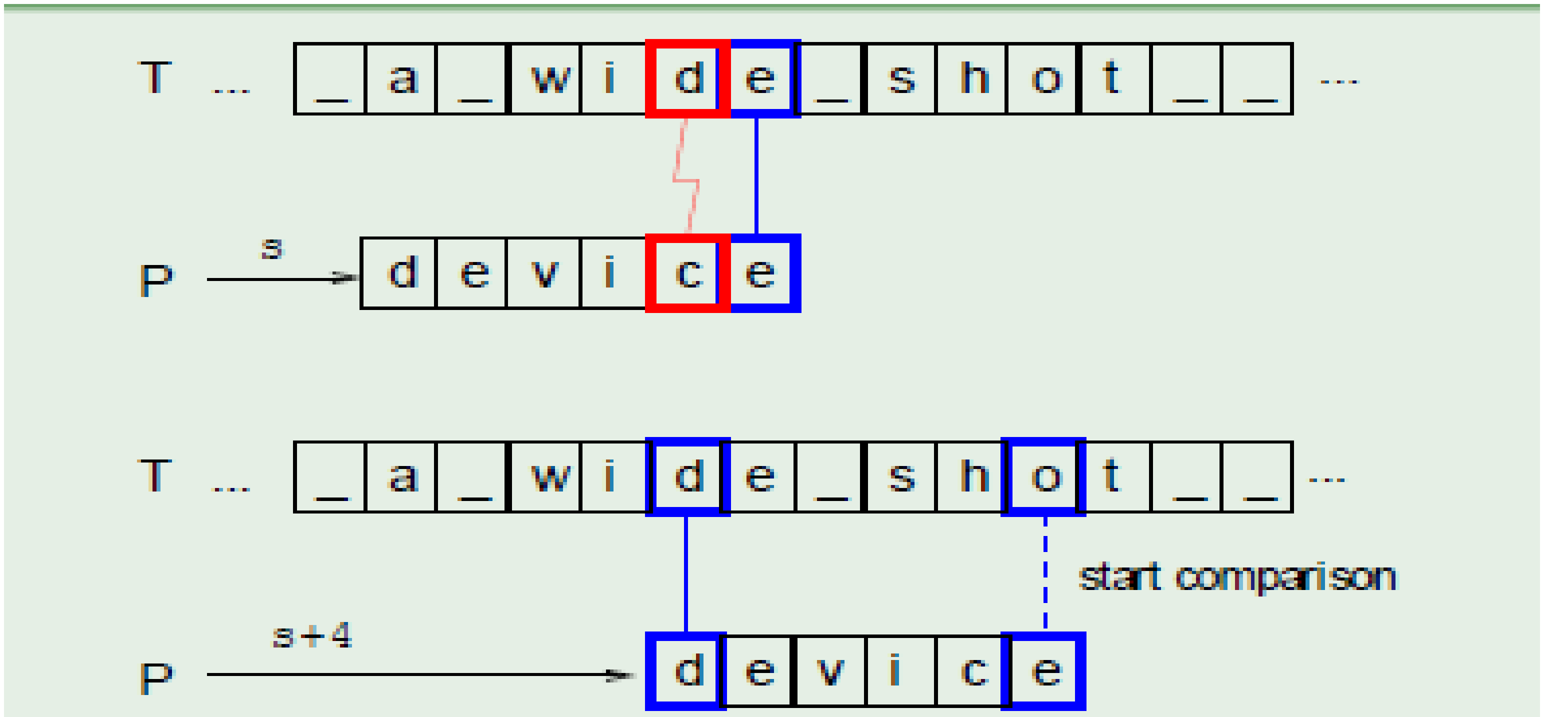
Shift P such that T[i] is aligned with the rightmost occurrence of T[i] in P



Learn DAA : From B K Sharma

Boyer Moore Algorithm

Bad Symbol Shift Rule: Case 2



Boyer Moore Algorithm

Bad Symbol Shift Rule

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:	F	I	Z	B	I	Z										

F and B don't match, shift to align F

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:				F	I	Z	B	I	Z							

$\boxed{\text{BadSymbolTable(F,2)=3}}$

Boyer Moore Algorithm

Bad Symbol Shift

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:				F	I	Z	B	I	Z							

I and Z don't match, shift to align I

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:					F	I	Z	B	I	Z						

$\boxed{\text{BadSymbolTable}(I,0)=1}$

Learn DAA : From B K Sharma

Boyer Moore Algorithm

Bad Symbol Shift

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:					F	I	Z	B	I	Z						

I and Z don't match, shift to align F

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:						F	I	Z	B	I	Z					

BadSymbolTable(I,3)=1

Boyer Moore Algorithm

Bad Symbol Shift

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:						F	I	Z	B	I	Z					

F and Z don't match, shift to align F

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:											F	I	Z	B	I	Z

FOUND!!!

BadSymbolTable(F,0)=5

Boyer Moore Algorithm

Bad Symbol Table

F and B don't match $\boxed{\text{BadSymbolTable(F,2)=3}}$

I and Z don't match $\boxed{\text{BadSymbolTable(I,0)=1}}$

I and Z don't match $\boxed{\text{BadSymbolTable(I,3)=1}}$

F and Z don't match $\boxed{\text{BadSymbolTable(F,0)=5}}$

Bad symbol table for FIZBIZ:

k^{th} row contains shift amount if mismatch occurred at index k

	A	B	C	...	F	...	I	...	Y	Z
0	6	2	6	...	5	...	1	...	6	3
1	5	1	5	...	4	...	-	...	5	2
2	4	-	4	...	3	...	2	...	4	1
3	3	3	3	...	2	...	1	...	3	-
4	2	2	2	...	1	...	-	...	2	2

Boyer Moore Algorithm

Good Suffix Rule

Find the longest suffix that matches:

Good Suffix Rule: Case 1

Suffix matched with the pattern and suffix appears to the left in P, preceded by a different char,

Then pattern can be shifted until the next occurrence of suffix in the pattern is aligned with the text symbols.

	1	2	3	4	5	6	7	8	9	10	11	12
T:	a	b	**a**	a	b	a	b	a	c	b	a	b
P:	c	a	b	a	b							
P:			c	a	b	a	b					

Boyer Moore Algorithm

Good Suffix Rule

Find the longest suffix that matches:

Good Suffix Rule: Case 2(a)

Suffix matched with the pattern but there is no occurrence of suffix in the pattern:

Then pattern can be shifted behind the suffix in the text

	1	2	3	4	5	6	7	8	9	10	11	12
T:	a	b	**a**	a	b	a	b	a	c	b	a	b
P:	c	b	c	a	b							
P:						c	b	c	a	b		

Boyer Moore Algorithm

Good Suffix Rule

Find the longest suffix that matches:

Good Suffix Rule: Case 2(b)

Suffix matched with the pattern and there is no occurrence of suffix in the pattern but a prefix of pattern matches the suffix of the text in the end.

	1	2	3	4	5	6	7	8	9	10	11	12
T:	a	a	b	a	b	b	a	b	c	b	a	b
P:	a	b	b	a	b							
P:				a	b	b	a	b				

Boyer Moore Algorithm

Good Suffix Shift

Find the longest suffix that matches:

if that suffix appears to the left in P, preceded by a different char, shift to align.

if not, then shift the entire length of the word 1 spot.

Boyer Moore Algorithm

Good Suffix Shift

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:	F	I	Z	B	I	Z										

IZ suffix matches, IZ appears to left in P, preceded by a different character, so shift to align

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z
P:				F	I	Z	B	I	Z							

Learn DAA : From B K Sharma

Boyer Moore Algorithm

Good Suffix Shift

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z

P:	F	I	Z	B	I	Z

No suffix match, so shift 1 spot

T:	B	I	Z	F	I	Z	I	B	I	Z	F	I	Z	B	I	Z

P:	F	I	Z	B	I	Z

Boyer Moore Algorithm

Good Suffix Table

IZ suffix matches, IZ appears to left → goodSuffixTable(IZ) = 3

No suffix match → goodSuffixTable() = 1

BIZ suffix matches, doesn't appear agai → goodSuffixTable(BIZ) = 6

IZBIZ	ZBIZ	BIZ	IZ	Z	
6	6	6	3	6	1

Boyer Moore Algorithm

1. Calculate the bad symbol and good suffix shift tables.
2. While match not found and not off the edge:
   a) compare pattern with string section
   b) shift1 = bad symbol shift of rightmost non-matching char
   c) shift2 = good suffix shift of longest matching suffix
   d) shift string section for comparison by max(shift1, shift2)

Boyer Moore Algorithm

Complexity

$O(m)$

Boyer More Algorithm

- The (BM) algorithm
  - Slides P from left to right;
  - At each shift, it compares P and T from right to left
    - First compare P[m] with T[i].
    - If match, compare P[m – 1] with T[i – 1].
    - If match, compare P[m – 2] with T[i – 2].
    - . . .
- When mismatch occurs, use two heuristics to determine the number of positions that P can be shifted to the right:
  - Bad-Character Heuristic
  - Good Suffix Heuristic.