String Searching

String Searching

- String searching is to find all the occurrences of a pattern in a text.

- **String Searching Domains**
  - Text editing
  - DNA (Deoxyribo Nucleic Acid) sequence matching
  - Web searching
  - Spelling and Grammar checking
  - Compression etc.

Notations and Terminology

- T[1 .. n]; array of text having length n
- P[1 .. m]; array of pattern to be searched in text T having length m, where $m \le n$
- $\sum$ is the finite set of alphabets.
  e.g. $\sum = \{0, 1\}$ or $\sum = \{a, b, \ldots z\}$, etc.
- P and T are often called strings of characters

Notations and Terminology

- **Valid & Invalid Shift**

  If P occurs in T beginning at position s+1, where $0 \le s \le n - m$ and T[s+1.. s+m], then we call 's' a Valid Shift otherwise 's' is called an Invalid Shift.

T	a	b	c	**a**	**b**	**a**	**a**	b	c	a	b	a	c

P	$\xrightarrow{s=3}$	**a**	**b**	**a**	**a**

Notations and Terminology

- $\sum^*$ (sigma star) denote the set of all finite length strings formed using characters from the alphabet set $\sum$
- ε denotes the Empty (zero length) String, also belongs to $\sum^*$
- $|x|$ denotes the Length of string x
- xy denotes the Concatenation of two strings x and y having length $|x| + |y|$

Notations and Terminology

- Prefix of a String

  ω is a prefix of a string x, denoted by $\omega \sqsubset x$, if $x = \omega y$ for some string $y \in \sum^*$
- If $\omega \sqsubset x$ then $|\omega| \leq |x|$
- Suffix of a String

  ω is a suffix of a string x, denoted by $\omega \sqsupset x$, if $x = y\omega$ for some string $y \in \sum^*$
- If $\omega \sqsupset x$ then $|\omega| \leq |x|$

Notations and Terminology

- The empty string ε is both a suffix and a prefix of every string
- For any strings x and y and any character a, we have $x \sqsupset y$ if and only if $xa \sqsupset ya$
- $\sqsupset$ and $\sqsubset$ are transitive operations.

Notations and Terminology

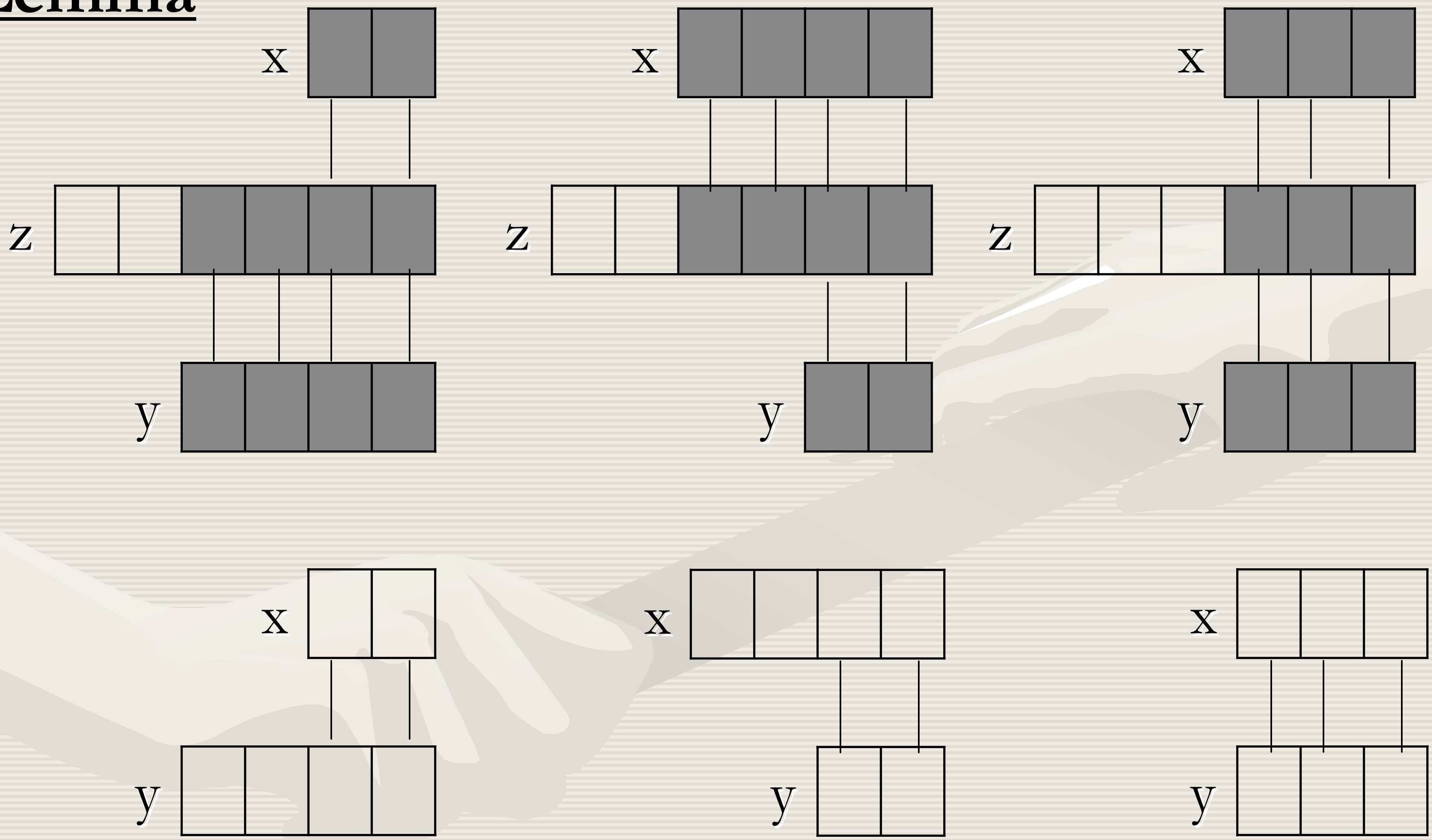
- **Lemma**

  Suppose that x, y and z are strings such that $x \sqsupseteq z$ and $y \sqsupseteq z$

  a) If $|x| \leq |y|$ then $x \sqsupseteq y$
  b) If $|x| \geq |y|$ then $y \sqsupseteq x$
  c) If $|x| = |y|$ then $x = y$

Notations and Terminology

Lemma



(a) (b) (c)

Algorithm Categories

String Matching Algorithms

- Non-Preprocessing
  - Naïve
  - Boyer-Moore
- Preprocessing
  - Finite Automaton
  - Knuth-Morris-Pratt

Naïve String Matching Algorithm

- The naïve string-matching procedure can be interpreted graphically as sliding a “template” containing the pattern over the text.
- Noting for which shifts all of the characters on the template equal the corresponding characters in the text.
- The whole string is scanned character by character without any optimization or pre-processing.
- Also called **Brute Force Pattern Matching**.

Naïve String Matching Algorithm

NAÏVE-STRING-MATCHER (T, P)

n ← length[T]

m ← length[P]

for s ← 0 **to** n – m

  do if P[1 .. m] = T[s+1 .. s+m]

    then print "Pattern occurs at shift" s

Execution of Naïve Algorithm

- T = abababacaba (Text)
- P = ababaca (Pattern)
- n = 11 (Length of Text)
- m = 7 (Length of Pattern)

Execution of Naïve Algorithm

n-m = 4, so loop will execute (0 – 4) 5 times

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

#	s	P[1 .. m] = T[s+1 .. s+m]	Output
1	0	ababaca = abababa (**false**)	
2	1	ababaca = bababac (**false**)	
3	2	ababaca = ababaca (**true**)	Pattern Occurs at Shift **2**
4	3	ababaca = babacab (**false**)	
5	4	ababaca = abacaba (**false**)	

Properties of Naïve Algorithm

- No Pre-processing is involved
- Total running time is $O((n-m+1)m)$
- No track of previously read characters
- May be suitable for smaller texts
- Performance decreases as the text grows larger
- No character set dependence
- No optimization has been performed

Finite Automata

- Build and use a finite-automaton that scans the text string T for all occurrences of the pattern P.
- String automata are efficient: it examines each text character exactly once, taking constant time per text character. But the time to build the automaton, however, can be large if $\sum$ is large.
- The automaton must be constructed from the pattern in a preprocessing step before it can be used to search the text string.

Finite Automata

- A **Finite Automata M** is a 5-tuple $(Q, q_0, A, \sum, \delta)$, where
- **Q** is a finite set of **states**
- $\mathbf{q_0} \in Q$ is the **start state**
- $\mathbf{A} \subseteq Q$ is a distinguished set of **accepting states**
- $\sum$ is finite **input alphabet (character set)**
- $\boldsymbol{\delta}$ is a function from $Q \times \sum$ into Q, called the **transition function of M**.
- The finite automaton begins is state q_0 and reads the characters of its input string one at a time. If the automaton is in state q and reads input character a, it moves ("makes a transition") from state q to state $\delta(q, a)$

Finite Automata

COMPUTE-TRANSITION-FUNCTION (P, ∑)

m $\leftarrow$ length[P]

for q $\leftarrow$ 0 **to** m

do for each character a ∈ ∑

do k $\leftarrow$ min (m+1, q+2)

repeat

k $\leftarrow$ k - 1

until $P_k \sqsupset P_q a$

$\delta(q, a) \leftarrow k$

return δ

Finite Automata

FINITE-AUTOMATON-MATCHER (T, δ, m)

```
n ←—— length[T]
q ←—— 0
for i ←—— 1 to n
    do q ←—— δ(q, T[i])
        if q = m
            then print "Pattern occurs with shift" i - m
```

Finite Automata

Computing the Transition Function

- P = ababaca (Pattern)
- m = 7 (Length of Pattern)
- $\sum = \{a, b, c\}$ (Character Set)
- At start computing the transition function which gives us the state transitions depending upon the input character and current state.

Finite Automata

Computing the Transition Function

m = 7, so the outer loop will execute (0 – 7) 8 times. **P_x = P[1 .. x]**

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
0	a	k=min(7+1, 0+2)=2	2-1=1	a⊐a (**T**)	δ(0, a)=1
	b	k=min(7+1, 0+2)=2	2-1=1	a⊐b (**F**)	
			1-1=0	ϵ⊐b (**T**)	δ(0, b)=0
	c	k=min(7+1, 0+2)=2	2-1=1	a⊐c (**F**)	
			1-1=0	ϵ⊐c (**T**)	δ(0, c)=0
1	a	k=min(7+1, 1+2)=3	3-1=2	ab⊐aa (**F**)	
			2-1=1	a⊐aa (**T**)	δ(1, a)=1

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times.

P_x = P[1 .. x]

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

q	**a**	**k=min(m+1, q+2)**	**k=k-1**	**$P_k \sqsupset P_q$ a**	**δ(q, a)**
1	b	k=min(7+1, 1+2)=3	3-1=2	ab$\sqsupset$ab (**T**)	δ(1, b)=2
	c	k=min(7+1, 1+2)=3	3-1=2	ab$\sqsupset$ac (**F**)	
			2-1=1	a$\sqsupset$ac (**F**)	
			1-1=0	ϵ$\sqsupset$ac (**T**)	δ(1, c)=0
2	a	k=min(7+1, 2+2)=4	4-1=3	aba$\sqsupset$aba (**T**)	δ(2, a)=3
	b	k=min(7+1, 2+2)=4	4-1=3	aba$\sqsupset$abb (**F**)	
			3-1=2	ab$\sqsupset$abb (**F**)	
			2-1=1	a$\sqsupset$abb (**F**)	
			1-1=0	ϵ$\sqsupset$abb (**T**)	δ(2, b)=0

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. $P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
2	c	k=min(7+1, 2+2)=4	4-1=3	aba⊐abc (**F**)	
			3-1=2	ab⊐abc (**F**)	
			2-1=1	a⊐abc (**F**)	
			1-1=0	ϵ⊐abc (**T**)	δ(2, c)=0
3	a	k=min(7+1, 3+2)=5	5-1=4	abab⊐abaa (**F**)	
			4-1=3	aba⊐abaa (**F**)	
			3-1=2	ab⊐abaa (**F**)	
			2-1=1	a⊐abaa (**T**)	δ(3, a)=1
	b	k=min(7+1, 3+2)=5	5-1=4	abab⊐abab (**T**)	δ(3, b)=4

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times.

P_x = P[1 .. x]

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

q	**a**	**k=min(m+1, q+2)**	**k=k-1**	**$P_k \sqsupset P_q a$**	**δ(q, a)**
3	c	k=min(7+1, 3+2)=5	5-1=4	abab⊐abac (**F**)	
			4-1=3	aba⊐abac (**F**)	
			3-1=2	ab⊐abac (**F**)	
			2-1=1	a⊐abac (**F**)	
			1-1=0	ϵ⊐abac (**T**)	δ(3, c)=0
4	a	k=min(7+1, 4+2)=6	6-1=5	ababa⊐ababa (**T**)	δ(4, a)=5
	b	k=min(7+1, 4+2)=6	6-1=5	ababa⊐ababb (**F**)	
			5-1=4	abab⊐ababb (**F**)	
			4-1=3	aba⊐ababb (**F**)	

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. **P_x = P[1 .. x]**

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
			3-1=2	ab⊐ababb (**F**)	
			2-1=1	a⊐ababb (**F**)	
			1-1=0	ϵ⊐ababb (**F**)	δ(4, b)=0
	c	k=min(7+1, 4+2)=6	6-1=5	ababa⊐ababc (**F**)	
			5-1=4	abab⊐ababc (**F**)	
			4-1=3	aba⊐ababc (**F**)	
			3-1=2	ab⊐ababc (**F**)	
			2-1=1	a⊐ababc (**F**)	
			1-1=0	ϵ⊐ababc (**F**)	δ(4, c)=0

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. $P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
5	a	k=min(7+1, 5+2)=7	7-1=6	ababac⊐ababaa (**F**)	
			6-1=5	ababa⊐ababaa (**F**)	
			5-1=4	abab⊐ababaa (**F**)	
			4-1=3	aba⊐ababaa (**F**)	
			3-1=2	ab⊐ababaa (**F**)	
			2-1=1	a⊐ababaa (**T**)	δ(5, a)=1
	b	k=min(7+1, 5+2)=7	7-1=6	ababac⊐ababab (**F**)	
			6-1=5	ababa⊐ababab (**F**)	
			5-1=4	abab⊐ababab (**T**)	δ(5, b)=4

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. $P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
5	c	k=min(7+1, 5+2)=7	7-1=6	ababac⊐ababac (**T**)	δ(5, c)=6
6	a	k=min(7+1, 6+2)=8	8-1=7	abababaca⊐abababaca (**T**)	δ(6, a)=7
	b	k=min(7+1, 6+2)=8	8-1=7	abababaca⊐abababacb (**F**)	
			7-1=6	ababac⊐abababacb (**F**)	
			6-1=5	ababa⊐abababacb (**F**)	
			5-1=4	abab⊐abababacb (**F**)	
			4-1=3	aba⊐abababacb (**F**)	
			3-1=2	ab⊐abababacb (**F**)	
			2-1=1	a⊐abababacb (**F**)	

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. $P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
			1-1=0	ϵ⊐ababacb (**T**)	δ(6, b)=0
	c	k=min(7+1, 6+2)=8	8-1=7	ababaca⊐ababacc (**F**)	
			7-1=6	ababac⊐ababacc (**F**)	
			6-1=5	ababa⊐ababacc (**F**)	
			5-1=4	abab⊐ababacc (**F**)	
			4-1=3	aba⊐ababacc (**F**)	
			3-1=2	ab⊐ababacc (**F**)	
			2-1=1	a⊐ababacc (**F**)	
			1-1=0	ϵ⊐ababacc (**T**)	δ(6, c)=0

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times.

$P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
7	a	k=min(7+1, 7+2)=8	8-1=7	ababaca⊐ababacaa (**F**)	
			7-1=6	ababac⊐ababacaa (**F**)	
			6-1=5	ababa⊐ababacaa (**F**)	
			5-1=4	abab⊐ababacaa (**F**)	
			4-1=3	aba⊐ababacaa (**F**)	
			3-1=2	ab⊐ababacaa (**F**)	
			2-1=1	a⊐ababacaa (**T**)	δ(7, a)=1
	b	k=min(7+1, 7+2)=8	8-1=7	ababaca⊐ababacab (**F**)	
			7-1=6	ababac⊐ababacab (**F**)	

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times.

$P_x = P[1 .. x]$

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
			6-1=5	ababa⊐abababacab (**F**)	
			5-1=4	abab⊐ababacab (**F**)	
			4-1=3	aba⊐ababacab (**F**)	
			3-1=2	ab⊐ababacab (**T**)	δ(7, b)=2
	c	k=min(7+1, 7+2)=8	8-1=7	ababaca⊐ababacac (**F**)	
			7-1=6	ababac⊐ababacac (**F**)	
			6-1=5	ababa⊐ababacac (**F**)	
			5-1=4	abab⊐ababacac (**F**)	
			4-1=3	aba⊐ababacac (**F**)	

Finite Automata

m = 7, so the outer loop will execute (0 – 7) 8 times. **P_x = P[1 .. x]**

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

q	a	k=min(m+1, q+2)	k=k-1	$P_k \sqsupset P_q a$	δ(q, a)
			3-1=2	ab⊐abababacac (**F**)	
			2-1=1	a⊐abababacac (**F**)	
			1-1=0	ϵ⊐abababacac (**T**)	δ(7, c)=0

Finite Automata

This matrix is showing the results of execution of transition function. These results will be used to execute the Finite-Automaton-Matcher to find any valid shifts.

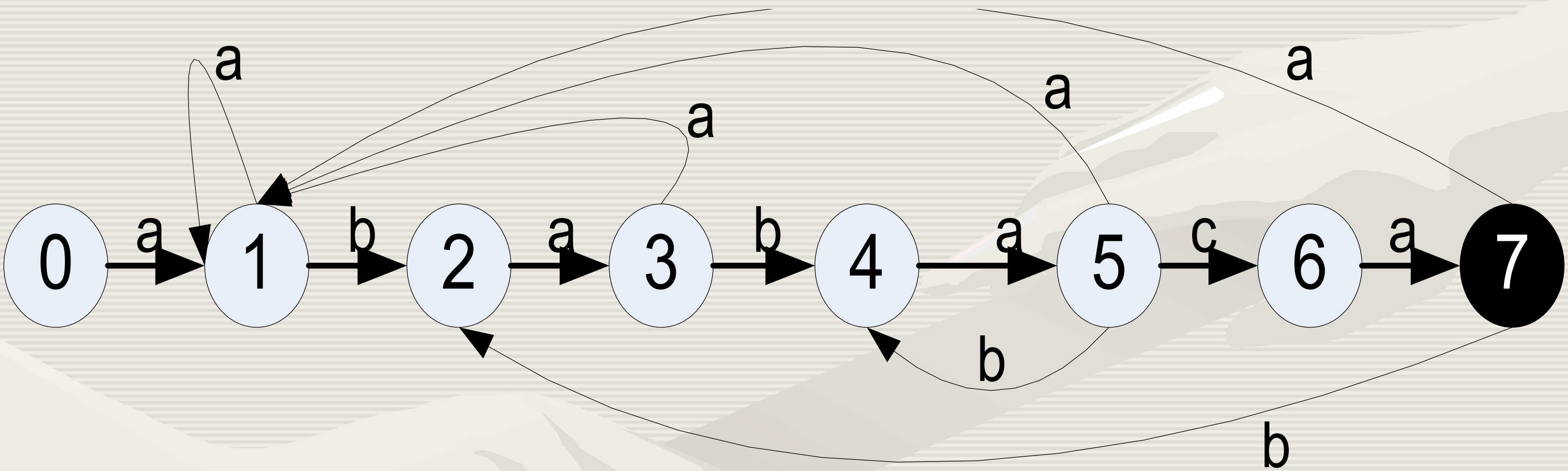
e.g. $\delta(3, b)=4$, $\delta(5, c)=6$ etc.

	input			
state	**a**	**b**	**c**	**P**
0	1	0	0	**a**
1	1	2	0	**b**
2	3	0	0	**a**
3	1	4	0	**b**
4	5	0	0	**a**
5	1	4	6	**c**
6	7	0	0	**a**
7	1	2	0	

Finite Automata

State Transition Diagram

A state transition diagram for the string matching automaton that accepts all strings ending in the string ababaca.



State 0 is the start state and state 7 is the only accepting state. A directed edge from state i to state j labeled a represents **δ(i, a) = j**. Some edges corresponding to failing matches are not shown; by convention, if a state i has no outgoing edge labeled a for some a ϵ ∑, then **δ(i, a) = 0**.

Finite Automata

Executing Finite-Automaton-Matcher

- T = abababacaba (Text)
- n = 11 (Length of Text)
- P = ababaca (Pattern)
- m = 7 (Length of Pattern)
- $\sum$ = {a, b ,c} (Character Set)
- q = 0 (Start State initially 0)

Finite Automata

n = 11, so loop will execute (1 – 11) 11 times

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

i	**q = δ(q, T[i])**	**q = m**	**Output**
1	q = δ(0, a) = 1	1 = 7 (**False**)	
2	q = δ(1, b) = 2	2 = 7 (**False**)	
3	q = δ(2, a) = 3	3 = 7 (**False**)	
4	q = δ(3, b) = 4	4 = 7 (**False**)	
5	q = δ(4, a) = 5	5 = 7 (**False**)	
6	q = δ(5, b) = 4	4 = 7 (**False**)	
7	q = δ(4, a) = 5	5 = 7 (**False**)	
8	q = δ(5, c) = 6	6 = 7 (**False**)	

Finite Automata

n = 11, so loop will execute (1 – 11) 11 times

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

i	q = δ(q, T[i])	q = m	Output
9	q = δ(6, a) = 7	7 = 7 (**True**)	Pattern occurs with shift (9 – 7) = **2**
10	q = δ(7, b) = 2	2 = 7 (**False**)	
11	q = δ(2, a) = 3	3 = 7 (**False**)	

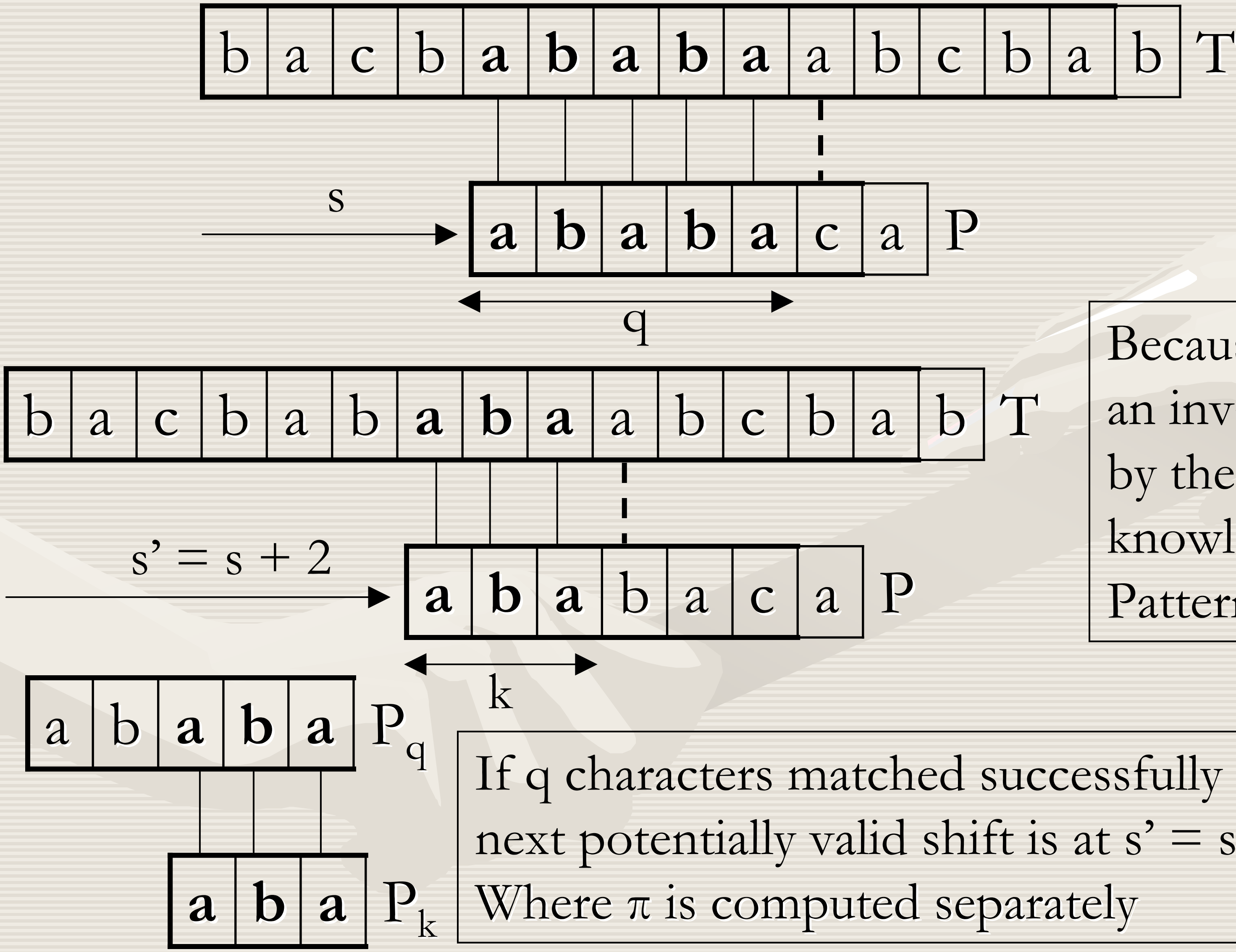
Properties of Finite Automata Algorithm

- Pre-processing is required
- Total running time is $O(m|\sum|) + \Theta(n)$
- Heavily dependent on Character Set
- Larger the Character Set more time it takes
- Performance decreases with the increase in the size of character set
- Un-necessarily stores all the valid and invalid state transitions

Knuth-Morris-Pratt Algorithm

- This algorithm avoids the computation of the transition function δ altogether, and its matching time is $\Theta(n)$ using just an auxiliary (prefix) function $\pi[1 .. m]$ pre-computed from the pattern in time $\Theta(m)$.
- The prefix function π for a pattern encapsulates knowledge about how the pattern matches against shifts to itself.
- This information can be used to avoid testing useless shifts in the naïve pattern-matching algorithm or to avoid the pre-computation of δ for a string-matching automaton.
- The information that q characters have matched successfully determines the corresponding text characters. Knowing these q text characters allows us to determine immediately that certain shifts are invalid.

Knuth-Morris-Pratt Algorithm



Because s + 1 is an invalid shift, by the knowledge of Pattern

If q characters matched successfully at shift s, the next potentially valid shift is at s’ = s + (q – π[q]), Where π is computed separately

Knuth-Morris-Pratt Algorithm

COMPUTE-PREFIX-FUNCTION (P)

```
m ← length[ P ]
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
```

Knuth-Morris-Pratt Algorithm

KMP-MATCHER (T, P)

n $\leftarrow$ length[T]
m $\leftarrow$ length[P]
π $\leftarrow$ COMPUTE-PREFIX-FUNCTION (P)
q $\leftarrow$ 0 ▹ Number of characters matched
for i $\leftarrow$ 1 **to** n **do** ▹ Scan text from left to right
while q > 0 **and** P[q + 1] $\neq$ T[i] **do**
q $\leftarrow$ π[q] ▹ Next character does not match
if P[q + 1] = T[i] **then**
q $\leftarrow$ q + 1 ▹ Next character matches
if q = m **then** ▹ Is all of P matched?
print "Pattern occurs with shift" i - m
q $\leftarrow$ π[q] ▹ Look for the next match

Knuth-Morris-Pratt Algorithm

- <u>**Computing the Prefix Function**</u>
- P = ababaca (Pattern)
- $m = 7$ (Length of Pattern)
- $\pi[1] = 0$ (Initial state is 0)
- $k = 0$
- First, computing the prefix function (π) which gives us the knowledge about how the pattern matches against shifts with itself. This information can be used to avoid testing useless shifts.

Knuth-Morris-Pratt Algorithm

Computing the Prefix Function

m = 7, so the outer loop will execute (2 – 7) 6 times. Initially **π[1] = 0, k = 0**

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

q	k>0 AND P[k+1]≠P[q]	k=π[k]	if P[k+1]=P[q]	k=k+1	π[q]=k
2	0>0 AND a≠b (**False**)		a=b (**False**)		π[2]=0
3	0>0 AND a≠a (**False**)		a=a (**True**)	0+1=1	π[3]=1
4	1>0 AND b≠b (**False**)		b=b (**True**)	1+1=2	π[4]=2
5	2>0 AND a≠a (**False**)		a=a (**True**)	2+1=3	π[5]=3
6	3>0 AND b≠c (**True**)	1			
	1>0 AND b≠c (**True**)	0			
	0>0 AND a≠c (**False**)		a=c (**False**)		π[6]=0
7	0>0 AND a≠a (**False**)		a=a (**True**)	0+1=1	π[7]=1

Knuth-Morris-Pratt Algorithm

Computing the Prefix Function

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

This matrix is showing the results of execution of prefix function. These results will be used to execute the KMP-Matcher to find any valid shifts.

State Matrix

i	**π[i]**
1	0
2	0
3	1
4	2
5	3
6	0
7	1

Knuth-Morris-Pratt Algorithm

Executing KMP-Matcher

- T = abababacaba (Text)
- n = 11 (Length of Text)
- P = ababaca (Pattern)
- m = 7 (Length of Pattern)
- π (Computed Prefix Function for P)
- q = 0 (Number of Characters Matched)

Knuth-Morris-Pratt Algorithm

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

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

n = 11, so outer loop will execute (1 – 11) 11 times

i	**q>0 And P[q+1]≠T[i]**	**q= π[q]**	**P[q+1]= T[i]**	**q=q+1**	**q=m**	**q=π[q]**	**Output**
1	0>0 AND a≠a (**F**)		a=a (**T**)	0+1=1	1=7 (**F**)		
2	1>0 AND b≠b (**F**)		b=b (**T**)	1+1=2	2=7 (**F**)		
3	2>0 AND a≠a (**F**)		a=a (**T**)	2+1=3	3=7 (**F**)		
4	3>0 AND b≠b (**F**)		b=b (**T**)	3+1=4	4=7 (**F**)		
5	4>0 AND a≠a (**F**)		a=a (**T**)	4+1=5	5=7 (**F**)		
6	5>0 AND c≠b (**T**) 3>0 AND b≠b (**F**)	3	b=b (**T**)	3+1=4	4=7 (**F**)		

Knuth-Morris-Pratt Algorithm

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

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

n = 11, so outer loop will execute (1 – 11) 11 times

i	q>0 And P[q+1]≠T[i]	q= π[q]	P[q+1]= T[i]	q=q+1	q=m	q= π[q]	Output
7	4>0 AND a≠a (**F**)		a=a (**T**)	4+1=5	5=7 (**F**)		
8	5>0 AND c≠c (**F**)		c=c (**T**)	5+1=6	6=7 (**F**)		
9	6>0 AND a≠a (**F**)		a=a (**T**)	6+1=7	7=7 (**T**)	1	Pattern Occurs with Shift (9-7) **2**
10	1>0 AND b≠b (**F**)		b=b (**T**)	1+1=2	2=7 (**F**)		
11	2>0 AND a≠a (**F**)		a=a (**T**)	2+1=3	3=7 (**F**)		

Properties of Knuth-Morris-Pratt Algorithm

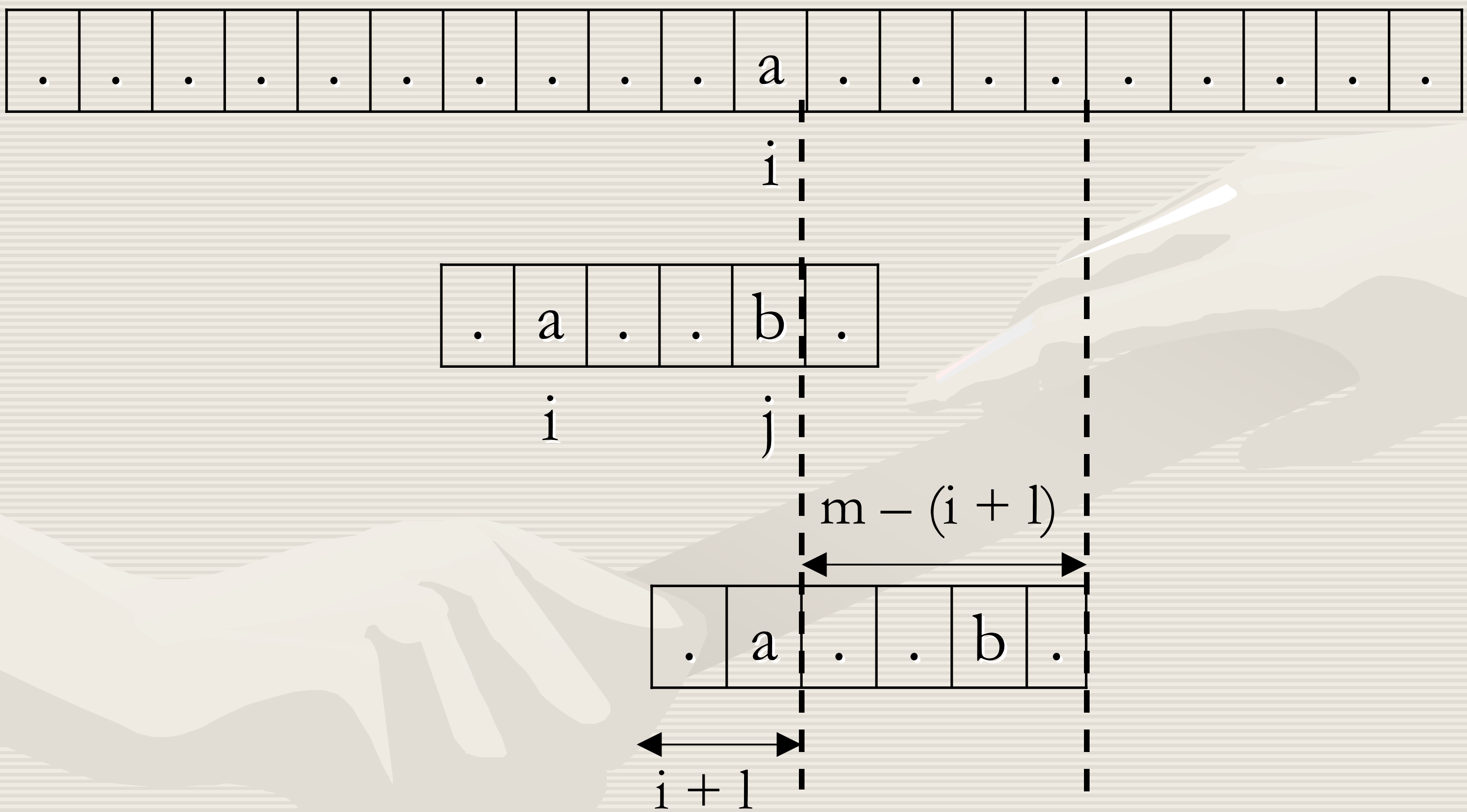
- Pre-processing is required
- Total running time is $\Theta(m) + \Theta(n)$
- No dependency on Character Set
- Linear execution time
- Efficiently use the information about the Pattern and the characters read so far.
- Most efficient of the four algorithms.

Boyer Moore Algorithm

- It works fastest when alphabet is moderately sized and pattern is relatively long.
- Pattern is matched from right to left so in case of a mismatch we can skip a considerable number of characters.
- **Character Jump Heuristic**
  - During the testing of possible placement of P against T, a mismatch of text character T[i]=c with the corresponding pattern character P[j] is handled as follows.
  - If c is not contained anywhere in P, then shift P completely past T[i] (for it cannot match any character in P). Otherwise shift P until an occurrence of character c in P gets aligned with T[i]

Boyer Moore Algorithm

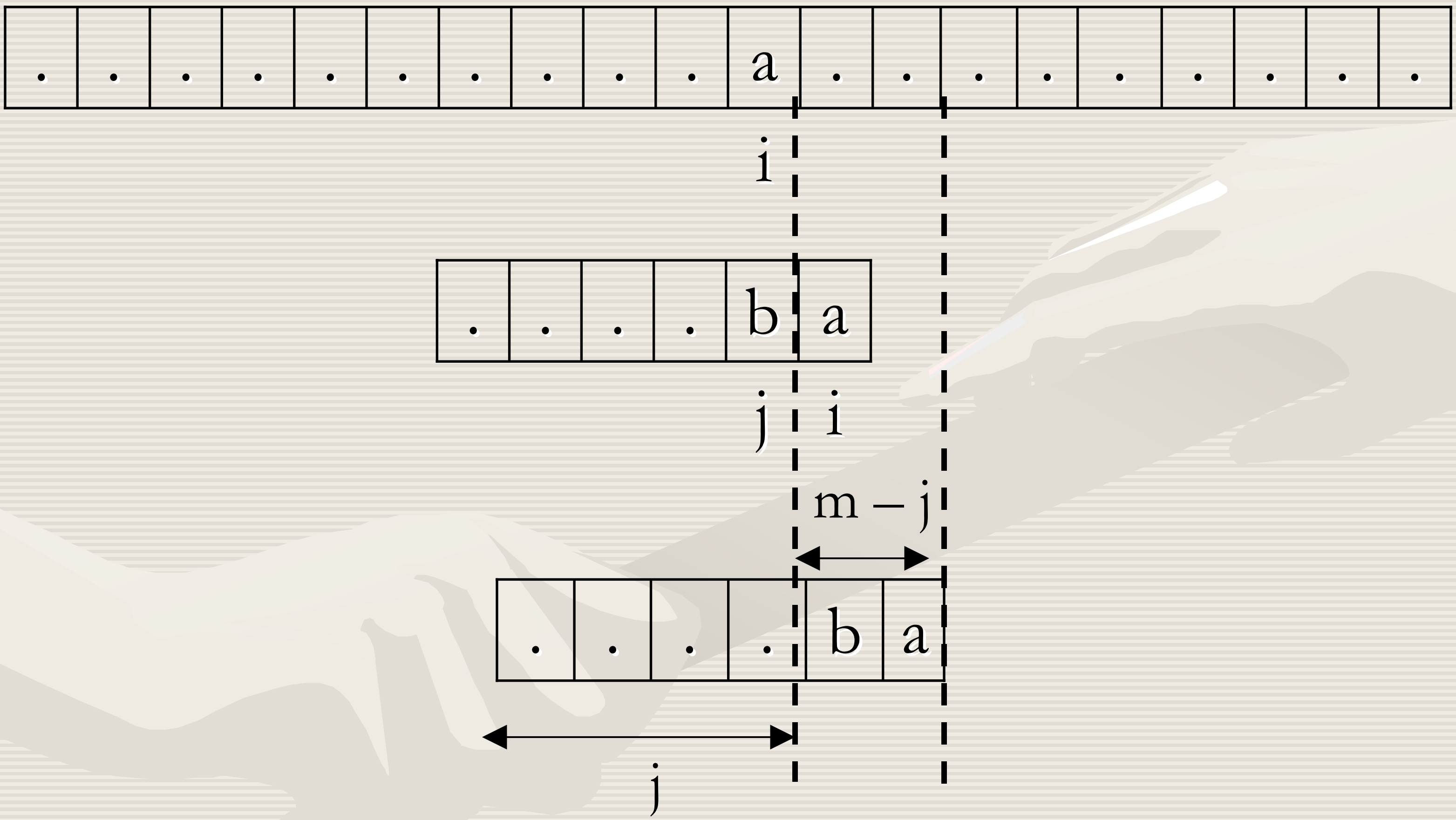
- Illustration of Jump in Boyer Moore Algorithm, where l denotes Last(P, T[i]) (to be discussed shortly)



- **Case 1: if $i + l \leq j$, then we shift the pattern by $j - l$ units**

Boyer Moore Algorithm

- Illustration of Jump in Boyer Moore Algorithm, where i denotes Last(P, T[i]) (to be discussed shortly)



- **Case 2: if j < i + 1, then we shift the pattern by one unit**

Boyer Moore Algorithm

a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b	
a	b	a	c	a	b															Since i+l ≤ j so shift by j - l characters
a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b	
	a	b	a	c	a	b														Since j < i+l so shift by 1 character
a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b	
		a	b	a	c	a	b													Since i+l ≤ j so shift by j - l characters
a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b	
			a	b	a	c	a	b												No match so shift by m characters
a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b	
									a	b	a	c	a	b						Since i+l ≤ j so shift by j-l characters

Boyer Moore Algorithm

a	b	a	c	a	a	b	a	d	c	a	b	a	c	a	b	a	a	b	b

a	b	a	c	a	b

………..

Boyer Moore Algorithm

BOYER-MOORE-MATCHER (T, P)

```
m ← length[ P ]
n ← length[ T ]
i ← m - 1
j ← m - 1
repeat
    if P[ j ] = T [ i ] then
        if j = 0 then
            print "Pattern Occurs with Shift" i
            i ← i + m
            j ← m – 1
        else
            i ← i – 1
            j ← j – 1
    else
        i ← i + m – min(j, 1 + last(P, T[ i ]))
        j ← m – 1
until i > n - 1
```

Boyer Moore Algorithm

LAST(P, c)

$i \leftarrow$ length[P] - 1

do while $i \geq 0$

if P[i] = c **then**

return i

$i \leftarrow i - 1$

return -1

Since index start from 0 therefore

e.g. Last (abxyabax, b) will return 5

Last (abxyabax, a) will return 6

Last (abxyabax, c) will return -1

- This function takes the Pattern P and a character c as input and return the index of the first character from the right side of pattern, if exists otherwise returns -1.

Boyer Moore Algorithm

- **Executing Boyer-Moore-Matcher**
- T = ababababacaba (Text)
- n = 11 (Length of Text)
- P = abababaca (Pattern)
- m = 7 (Length of Pattern)
- i=6 (m-1)
- j=6 (m-1)

Boyer Moore Algorithm

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

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

n=11 so loop will execute until i>10 (i>n-1), Initially i=6, and j=6 because i=j=m-1

P[j]= T[i]	**j=0**	**i=i-1**	**j=j-1**	**i = i + m – min (j, 1 + last(P, T[i]))**	**j= m-1**	**Result**
a=a (**T**)	F	6-1=5	6-1=5			
c=b (**F**)				=5+7 – min(5, 1+ last (P, b)) =12 – min(5, 1+3) =12 – 4 =8	7-1=6	
a=a (**T**)	F	8-1=7	6-1=5			
c=c (**T**)	F	7-1=6	5-1=4			
a=a (**T**)	F	6-1=5	4-1=3			

Boyer Moore Algorithm

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

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

n=11 so loop will continue to execute while i<=10 (i.e. until i>10 (i>n-1))

P[j]= T[i]	**j=0**	**i=i-1**	**j=j-1**	**i = i + m – min (j, 1 + last(P, T[i]))**	**j= m-1**	**Result**
b=b (**T**)	F	5-1=4	3-1=2			
a=a (**T**)	F	4-1=3	2-1=1			
b=b (**T**)	F	3-1=2	1-1=0			
a=a (**T**)	T	"Pattern occurs with shift" 2 and due to match i and j will be i = i + m = 2 + 7 = 9 j = m – 1 = 7 - 1 = 6				
a=b (**F**)				=9+7 – min(6, 1+ last (P, b)) =16 – min(6, 1+3) =16 – 4 = 12	7-1=6	

Boyer Moore Algorithm

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

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

n=11 so loop will continue to execute while i<=10 (i.e. until i>10 (i>n-1))

P[j]= T[i]	**j=0**	**i=i-1**	**j=j-1**	**i = i + m – min (j, 1 + last(P, T[i]))**	**j= m-1**	**Result**
Loop will terminate since i becomes 12 (>10) and because there are no more characters left in the string which can be completely matched with the pattern						

Properties of Boyer-Moore Algorithm

- No Pre-processing is required
- Total running time is $O(nm + |\sum|)$ (worst case)
- No track of previously read characters
- Scans the text from right to left
- No character set dependence
- Makes use of the pattern and index-character for Character Jumps (Character-Jump Heuristic)
- Good performance since a large portion of text is skipped

Algorithm Time Comparison

Algorithm	Pre-Processing Time	Matching Time	Total Running Time
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$	$\Theta(m) + \Theta(n)$
Finite Automaton	$O(m\|\sum\|)$	$\Theta(n)$	$O(m\|\sum\|) + \Theta(n)$
Naïve	0	$O((n-m+1)m)$	$O((n-m+1)m)$
Boyer-Moore	0	$O(nm + \|\sum\|)$	$O(nm + \|\sum\|)$