

Algorithmic Challenges: Knuth-Morris-Pratt Algorithm

Michael Levin

Higher School of Economics

Algorithms on Strings
Data Structures and Algorithms

Outline

- 1 Exact Pattern Matching
- 2 Safe Shift
- 3 Prefix Function
- 4 Computing Prefix Function
- 5 Knuth-Morris-Pratt Algorithm

Exact Pattern Matching

Input: Strings T (Text) and P (Pattern).

Output: All such positions in T (Text)
where P (Pattern) appears as a
substring.

(For all strings in this module we use 0-based indices)

Brute Force Algorithm

- Slide the Pattern down Text

Brute Force Algorithm

- Slide the Pattern down Text
- Running time $\Theta(|T||P|)$

Brute Force Algorithm

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

Brute Force Algorithm

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

Output: []

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0]

Brute Force Algorithm

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

Output: [0,7]

Skipping Positions

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

Skipping Positions

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

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	r	a	c	a	d	a	b	r	a
---	---	---	---	---	---	---	---	---	---	---

a	b	r	a
---	---	---	---

a	b	r	a
---	---	---	---

Skipping Positions

a	b	c	d	a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	c	d	a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	c	d	a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	c	d	a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	c	d	a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	c	d	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Skipping Positions

a	b	a	b	a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	a	b	e	f
---	---	---	---	---	---	---	---

Definitions

Definition

Border of string S is a prefix of S which is equal to a suffix of S , but not equal to the whole S .

Example

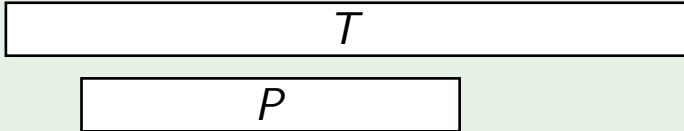
“a” is a **border** of “arba”

“ab” is a **border** of “abcdab”

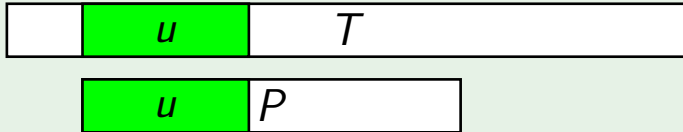
“abab” is a **border** of “ababab”

“ab” is **not** a **border** of “ab”

Shifting Pattern

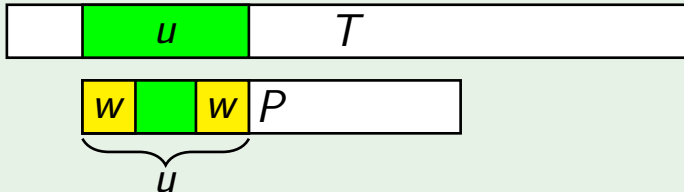


Shifting Pattern



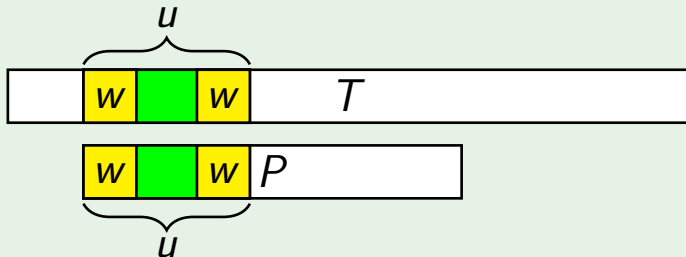
- Find longest common prefix u

Shifting Pattern



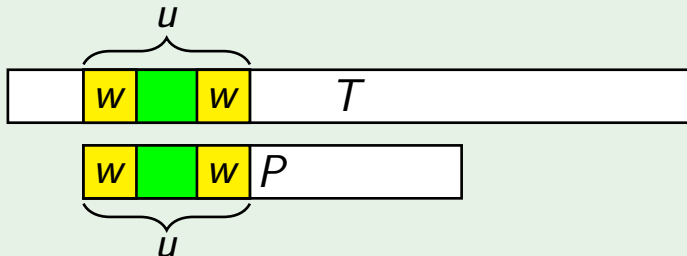
- Find longest common prefix u
- Find w — the longest border of u

Shifting Pattern



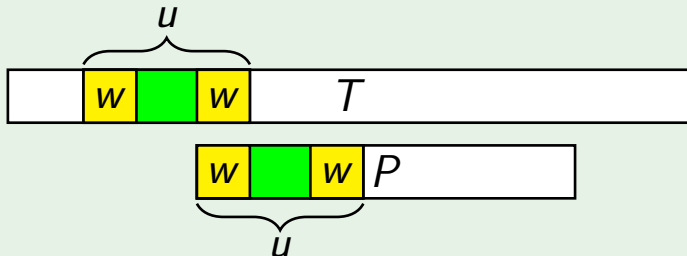
- Find longest common prefix u
- Find w — the longest border of u

Shifting Pattern



- Find longest common prefix u
- Find w — the longest border of u
- Move P such that prefix w in P aligns with suffix w of u in T

Shifting Pattern



- Find longest common prefix u
- Find w — the longest border of u
- Move P such that prefix w in P aligns with suffix w of u in T

- Now you know we can skip some of the comparisons

- Now you know we can skip some of the comparisons
- But we shouldn't miss any of the pattern occurrences in the text

- Now you know we can skip some of the comparisons
- But we shouldn't miss any of the pattern occurrences in the text
- Is it **safe** to shift the pattern this way?

Outline

- 1 Exact Pattern Matching
- 2 Safe Shift
- 3 Prefix Function
- 4 Computing Prefix Function
- 5 Knuth-Morris-Pratt Algorithm

Suffix notation

Definition

Denote by S_k suffix of string S starting at position k .

Examples

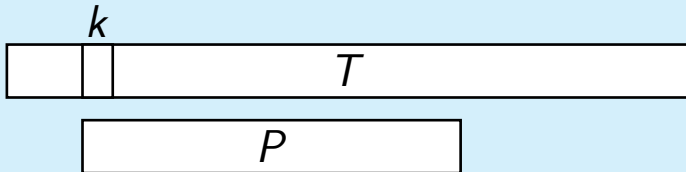
$$S = \text{"abcd"} \Rightarrow S_2 = \text{"cd"}$$

$$T = \text{"abc"} \Rightarrow T_0 = \text{"abc"}$$

$$P = \text{"aa"} \Rightarrow P_1 = \text{"a"}$$

Safe shift

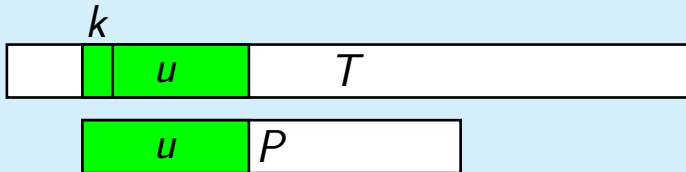
Lemma



Let u be the longest common prefix of P and T_k . Let w be the longest border of u . Then there are no occurrences of P in T starting between positions k and $(k + |u| - |w|)$ — the start of suffix w in the prefix u of T_k .

Safe shift

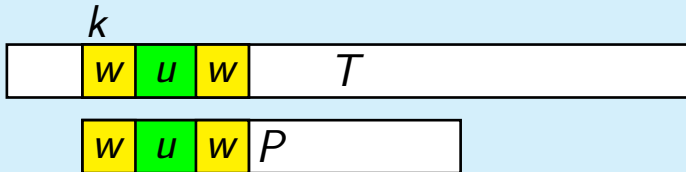
Lemma



Let u be the longest common prefix of P and T_k . Let w be the longest border of u . Then there are no occurrences of P in T starting between positions k and $(k + |u| - |w|)$ — the start of suffix w in the prefix u of T_k .

Safe shift

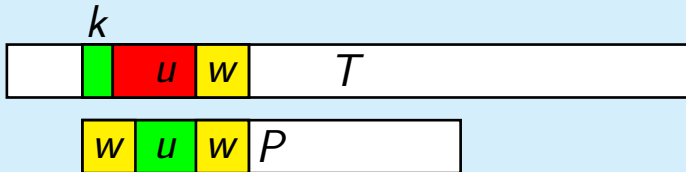
Lemma



Let u be the longest common prefix of P and T_k . Let w be the longest border of u . Then there are no occurrences of P in T starting between positions k and $(k + |u| - |w|)$ — the start of suffix w in the prefix u of T_k .

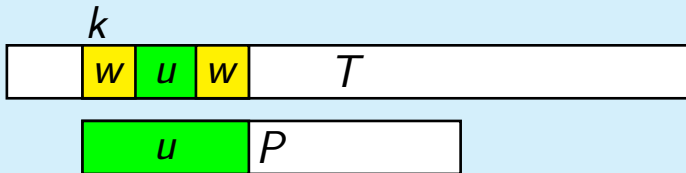
Safe shift

Lemma

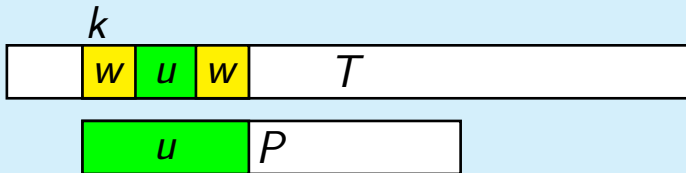


Let u be the longest common prefix of P and T_k . Let w be the longest border of u . Then there are no occurrences of P in T starting between positions k and $(k + |u| - |w|)$ — the start of suffix w in the prefix u of T_k .

Proof

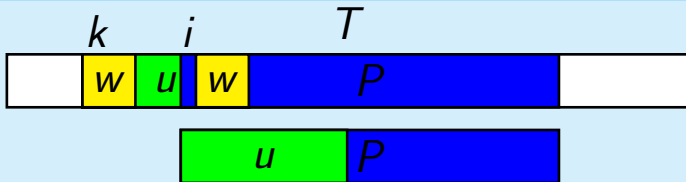


Proof



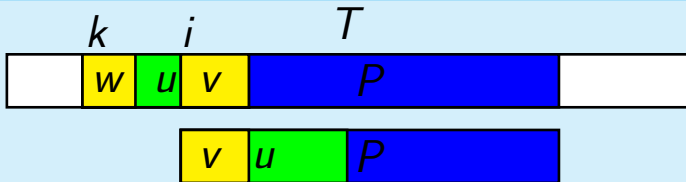
- Suppose P occurs in T in position i between k and start of suffix w

Proof



- Suppose P occurs in T in position i between k and start of suffix w

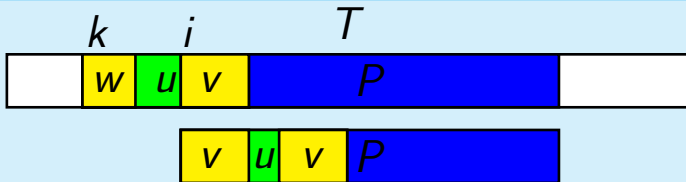
Proof



- Suppose P occurs in T in position i between k and start of suffix w
- Then there is prefix v of P equal to suffix in u , and v is longer than w



Proof



- Then there is prefix v of P equal to suffix in u , and v is longer than w
- v is a border longer than w , but w is longest border of u — contradiction □

- Now you know it is possible to avoid many of the comparisons which Brute Force algorithm does

- Now you know it is possible to avoid many of the comparisons which Brute Force algorithm does
- But how to determine the best pattern shifts?

Outline

- 1 Exact Pattern Matching
- 2 Safe Shift
- 3 Prefix Function
- 4 Computing Prefix Function
- 5 Knuth-Morris-Pratt Algorithm

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Prefix function

Definition

Prefix function of a string P is a function $s(i)$ that for each i returns the length of the longest border of the prefix $P[0..i]$.

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Lemma

$P[0..i]$ has a border of length $s(i+1) - 1$

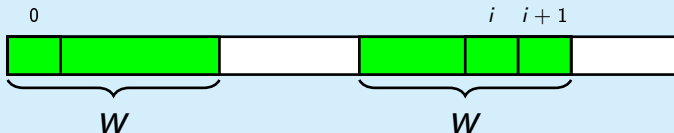
Proof



Lemma

$P[0..i]$ has a border of length $s(i+1) - 1$

Proof

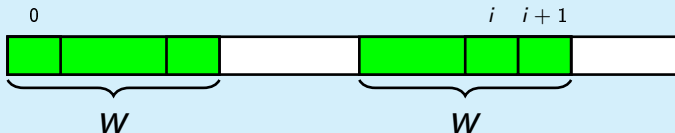


- Take the longest border w of $P[0..i+1]$

Lemma

$P[0..i]$ has a border of length $s(i+1) - 1$

Proof



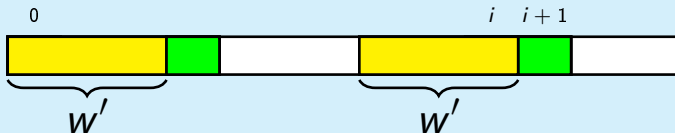
- Take the longest border w of $P[0..i+1]$
- Cut the last character from w — it's a border of $P[0..i]$ now



Lemma

$P[0..i]$ has a border of length $s(i+1) - 1$

Proof



- Take the longest border w of $P[0..i+1]$
- Cut the last character from w — it's a border of $P[0..i]$ now



Corollary

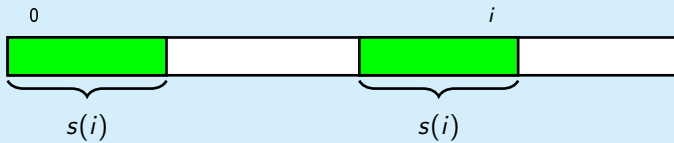
$$s(i + 1) \leq s(i) + 1$$

Enumerating borders

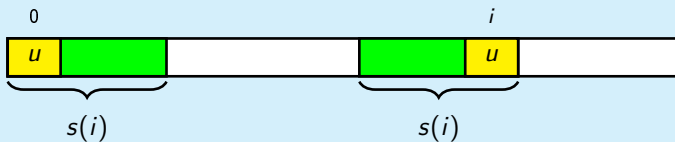
Lemma

If $s(i) > 0$, then all borders of $P[0..i]$ but for the longest one are also borders of $P[0..s(i) - 1]$.

Proof

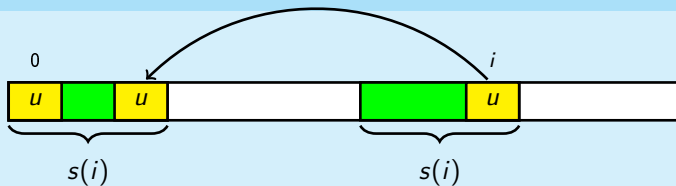


Proof



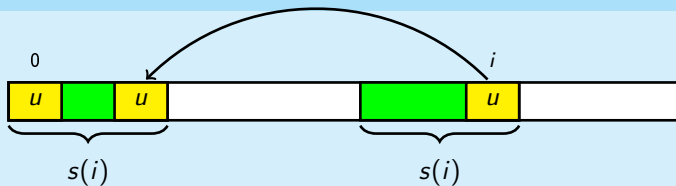
- Let u be a border of $P[0..i]$ such that $|u| < s(i)$

Proof



- Let u be a border of $P[0..i]$ such that $|u| < s(i)$
- Then u is both a prefix and a suffix of $P[0..s(i) - 1]$

Proof



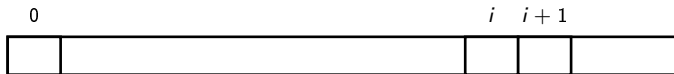
- Let u be a border of $P[0..i]$ such that $|u| < s(i)$
- Then u is both a prefix and a suffix of $P[0..s(i) - 1]$
- $u \neq P[0..s(i) - 1]$, so u is a border of $P[0..s(i) - 1]$

Enumerating borders

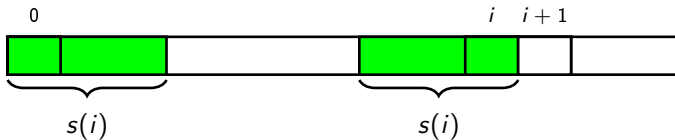
Corollary

All borders of $P[0..i]$ can be enumerated by taking the longest border b_1 of $P[0..i]$, then the longest border b_2 of b_1 , then the longest border b_3 of b_2 , \dots , and so on.

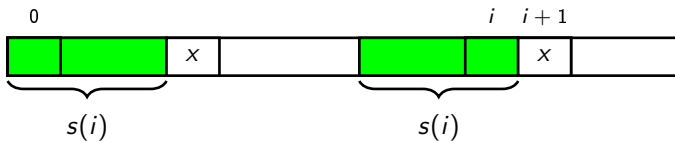
Computing $s(i + 1)$



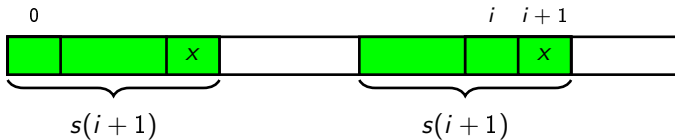
Computing $s(i + 1)$



Computing $s(i + 1)$

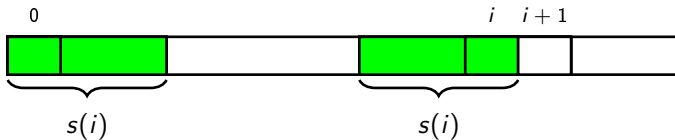


Computing $s(i + 1)$

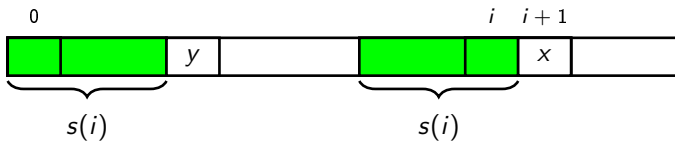


$$s(i+1) = s(i) + 1$$

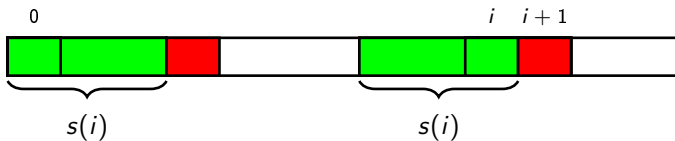
Computing $s(i + 1)$



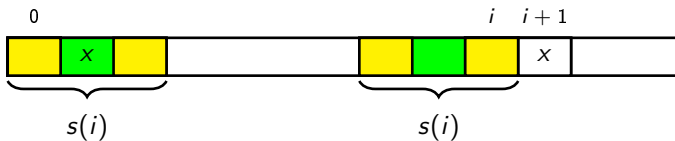
Computing $s(i + 1)$



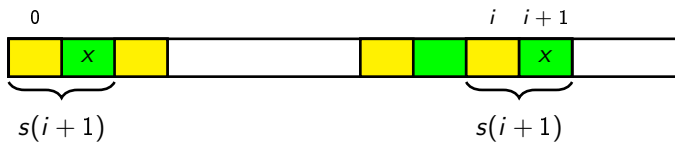
Computing $s(i + 1)$



Computing $s(i + 1)$



Computing $s(i + 1)$



$$s(i + 1) = |\text{some border of } P[0..s(i) - 1]| + 1$$

- Now you know lots of properties of prefix function

- Now you know lots of properties of prefix function
- But how to compute all of its values??

Outline

- 1 Exact Pattern Matching
- 2 Safe Shift
- 3 Prefix Function
- 4 Computing Prefix Function
- 5 Knuth-Morris-Pratt Algorithm

Example

[illegible]

Example

[illegible]

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0								

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0								

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0								

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1							

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1							

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1							

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2						

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2						

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2						

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3					

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3					

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3					

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4				

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0			

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0			

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0			

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1		

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1		

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1		

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1		

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1		

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

Example

P	a	b	a	b	a	b	c	a	a	b
s	0	0	1	2	3	4	0	1	1	2

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```


ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```

ComputePrefixFunction(P)

```
 $s \leftarrow$  array of integers of length  $|P|$   
 $s[0] \leftarrow 0, border \leftarrow 0$   
for  $i$  from 1 to  $|P| - 1$ :  
    while ( $border > 0$ ) and ( $P[i] \neq P[border]$ ):  
         $border \leftarrow s[border - 1]$   
    if  $P[i] == P[border]$ :  
         $border \leftarrow border + 1$   
    else:  
         $border \leftarrow 0$   
     $s[i] \leftarrow border$   
return  $s$ 
```


Lemma

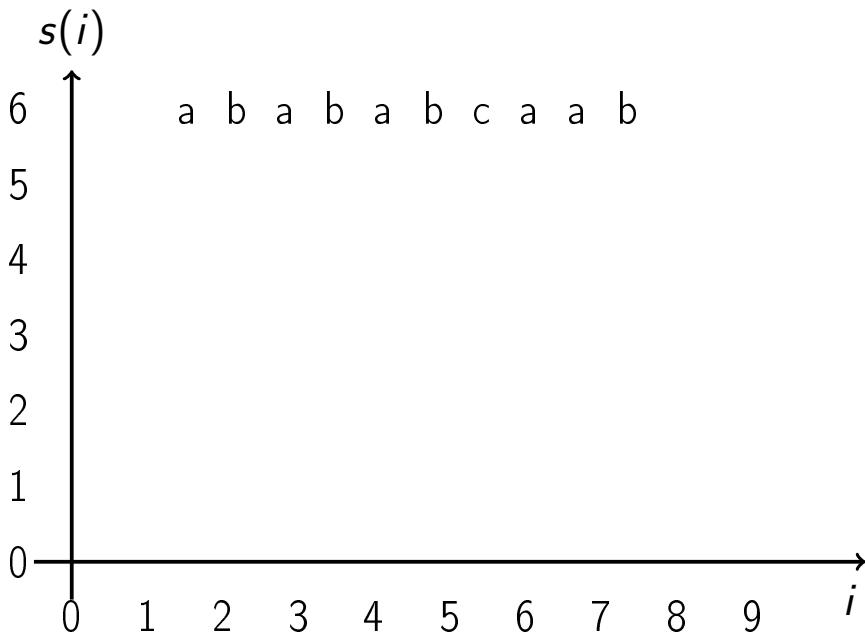
The running time of
ComputePrefixFunction is $O(|P|)$.

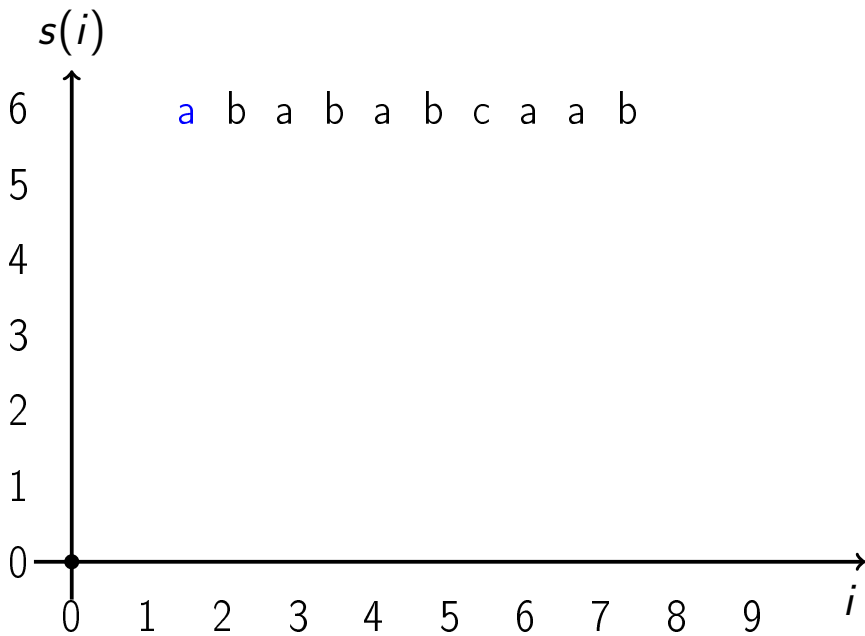
Proof

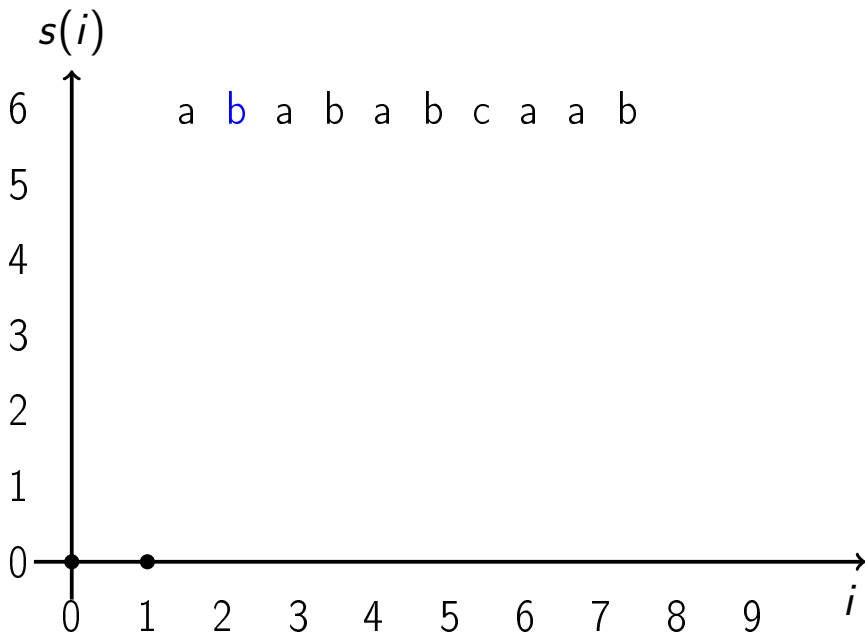
- Everything but for inner **while** loop is $O(|P|)$ initialization plus $O(|P|)$ iterations of the **for** loop with $O(1)$ assignments on each iteration

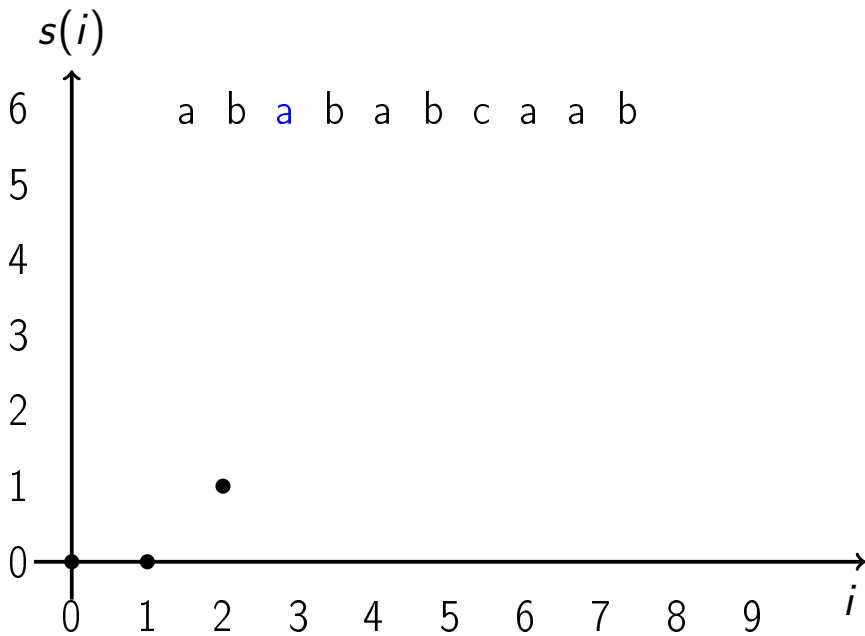
Proof

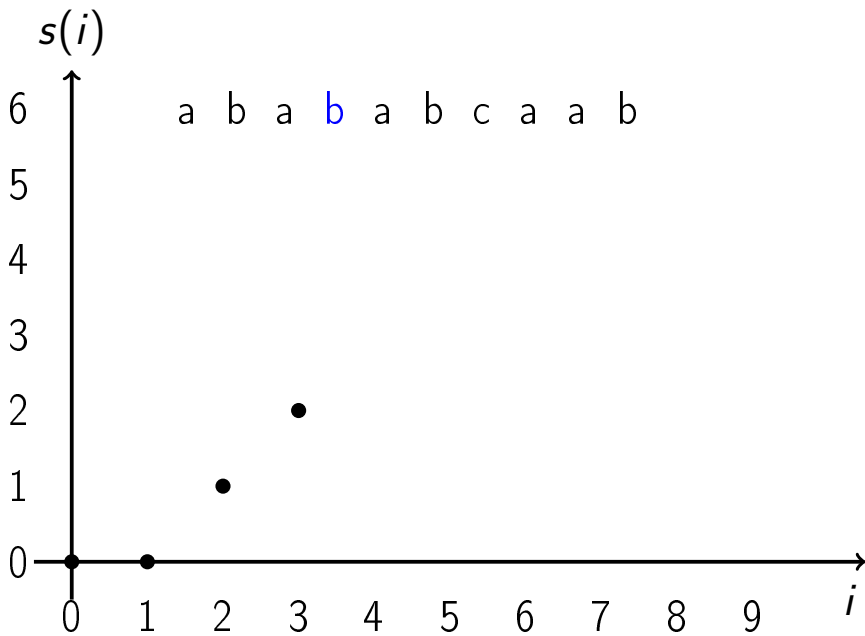
- Everything but for inner **while** loop is $O(|P|)$ initialization plus $O(|P|)$ iterations of the **for** loop with $O(1)$ assignments on each iteration
- Now we will bound the number of the **while** loop iterations by $O(|P|)$

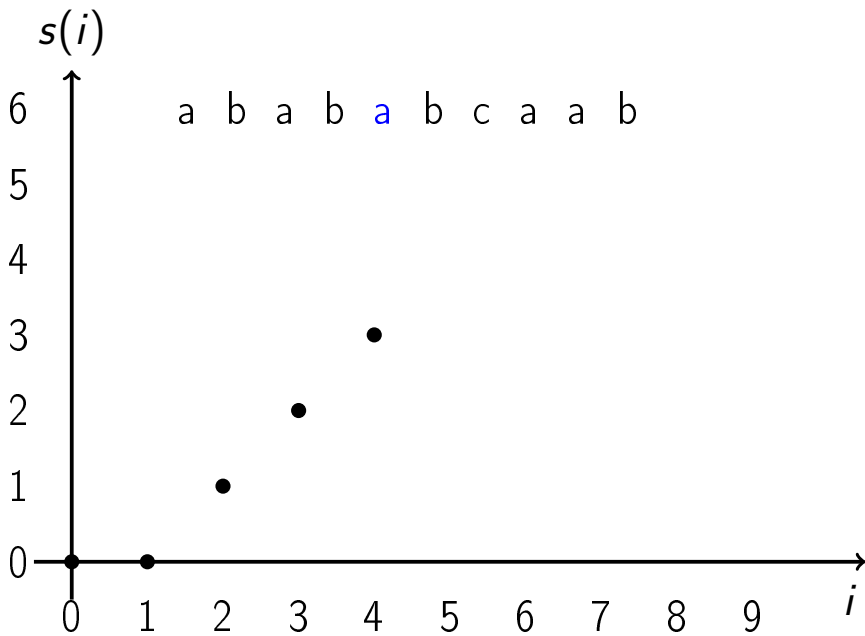


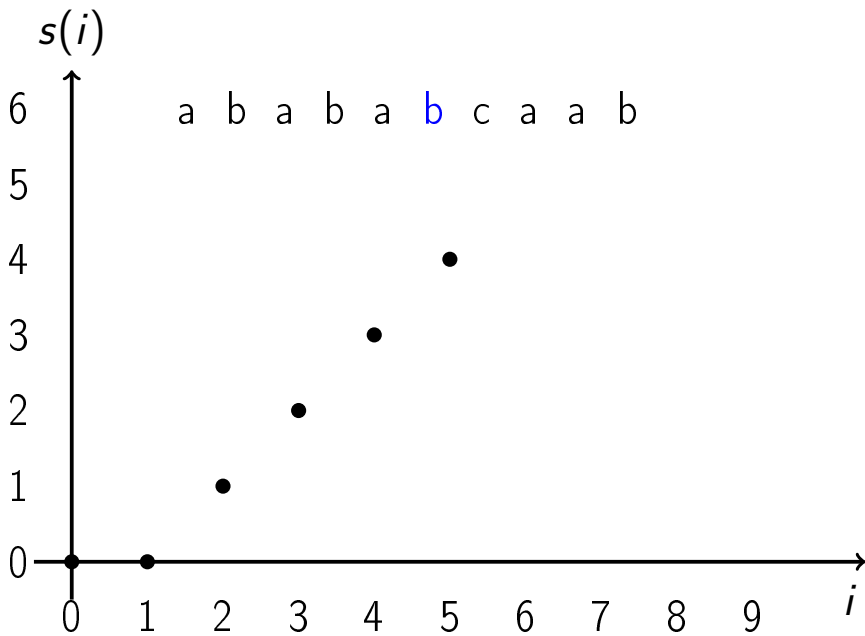


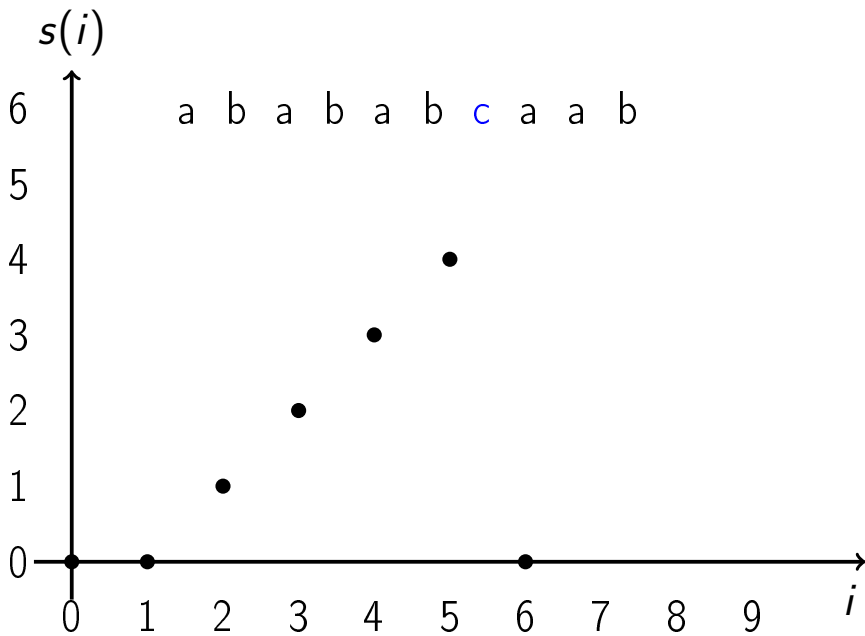


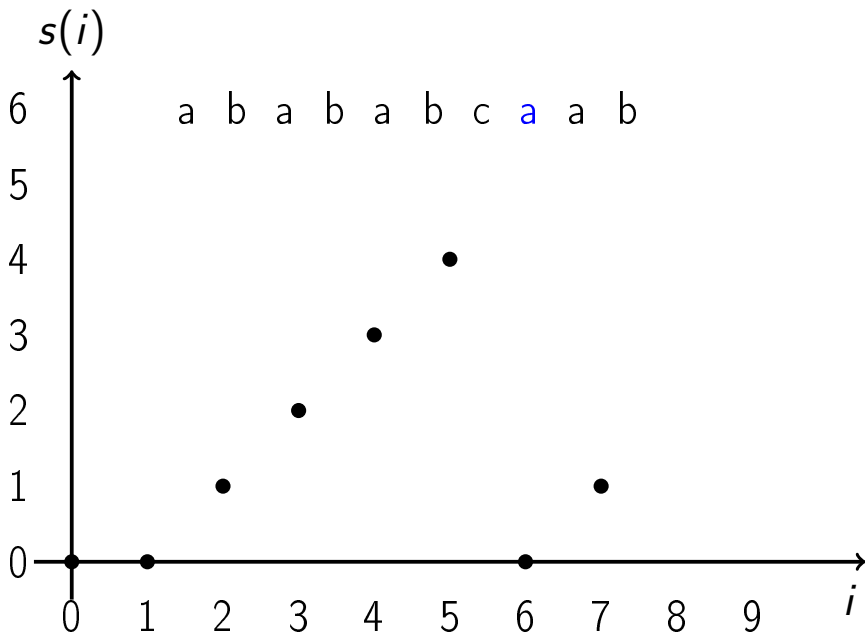


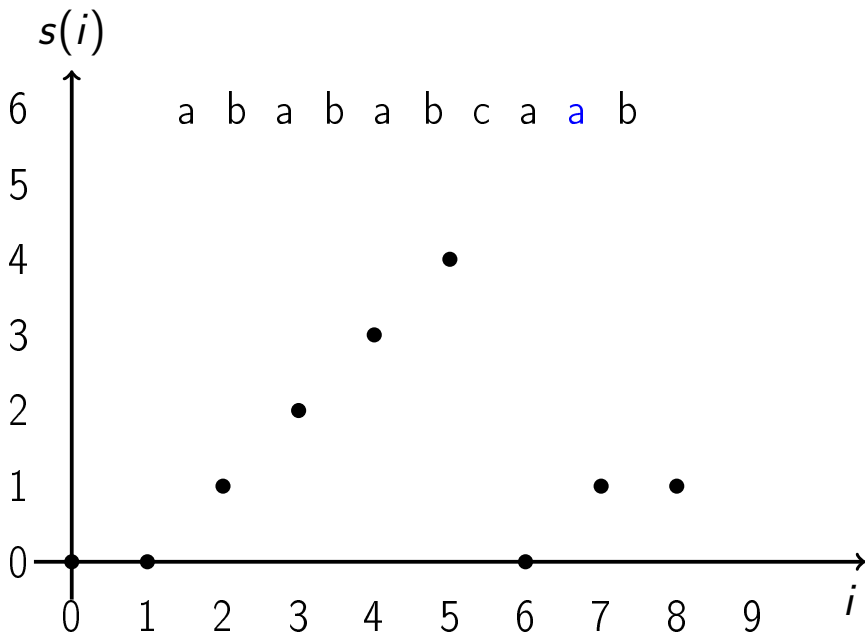


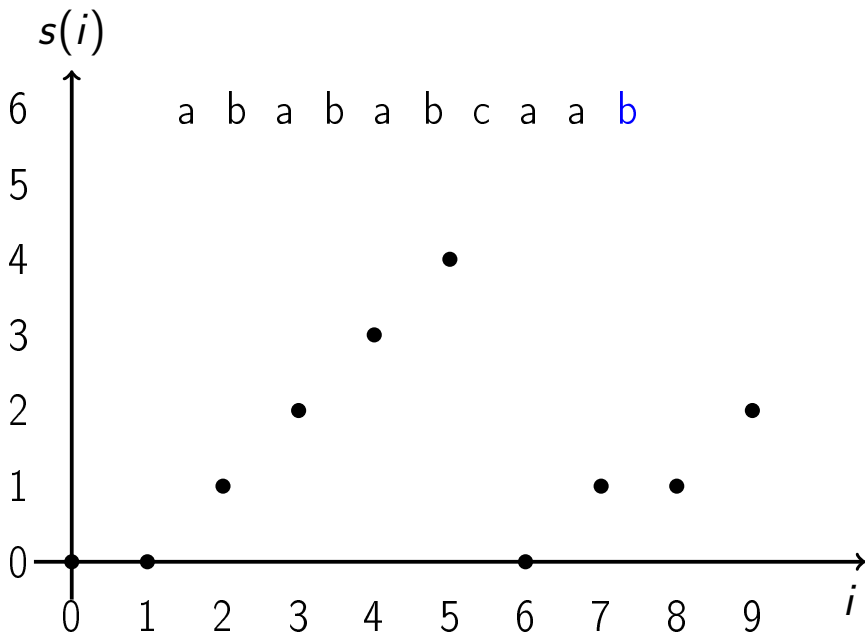












Proof

(continued)

- *border* can increase at most by 1 on each iteration of the for loop

Proof

(continued)

- *border* can increase at most by 1 on each iteration of the for loop
- In total, *border* is increased $O(|P|)$ times

Proof

(continued)

- *border* can increase at most by 1 on each iteration of the for loop
- In total, *border* is increased $O(|P|)$ times
- *border* is decreased at least by 1 on each iteration of the while loop

Proof

(continued)

- *border* can increase at most by 1 on each iteration of the for loop
- In total, *border* is increased $O(|P|)$ times
- *border* is decreased at least by 1 on each iteration of the while loop
- $\textit{border} \geq 0$

Proof

(continued)

- *border* can increase at most by 1 on each iteration of the for loop
- In total, *border* is increased $O(|P|)$ times
- *border* is decreased at least by 1 on each iteration of the while loop
- $\textit{border} \geq 0$
- So there are $O(|P|)$ iterations of the while loop



- Now you know how to compute prefix function in linear time

- Now you know how to compute prefix function in linear time
- But how to find pattern in text??

Outline

- 1 Exact Pattern Matching
- 2 Safe Shift
- 3 Prefix Function
- 4 Computing Prefix Function
- 5 Knuth-Morris-Pratt Algorithm

Algorithm

P T

S a b r a \$ a b r a c a d a b r a

To search for pattern P in text T :

- Create new string $S = P + '$' + T$,
where '\$' is a special character absent
from both P and T

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S
- For all positions i such that $i > |P|$ and $s(i) = |P|$, add $i - 2|P|$ to the output

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S
- For all positions i such that $i > |P|$ and $s(i) = |P|$, add $i - 2|P|$ to the output

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S
- For all positions i such that $i > |P|$ and $s(i) = |P|$, add $i - 2|P|$ to the output

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S
- For all positions i such that $i > |P|$ and $s(i) = |P|$, add $i - 2|P|$ to the output

Algorithm

	P						T									
S	a	b	r	a	\$	a	b	r	a	c	a	d	a	b	r	a
s	0	0	0	1	0	1	2	3	4	0	1	0	1	2	3	4

To search for pattern P in text T :

- Compute prefix function s for string S
- For all positions i such that $i > |P|$ and $s(i) = |P|$, add $i - 2|P|$ to the output

Explanation

- For all i , $s(i) \leq |P|$ because of the special character '\$'
- If $i > |P|$ and $s(i) = |P|$, then
$$P = S[0..|P| - 1] = S[i - |P| + 1..i] = T[i - 2|P|..i - |P| - 1]$$
- If $s(i) < |P|$, no full occurrence of $|P|$ ends in position i

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

FindAllOccurrences(P, T)

$S \leftarrow P + '$\$' + T$

$s \leftarrow \text{ComputePrefixFunction}(S)$

result \leftarrow empty list

for i from $|P| + 1$ to $|S| - 1$:

 if $s[i] == |P|$:

 result.Append($i - 2|P|$)

return result

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

FindAllOccurrences(P, T)

```
 $S \leftarrow P + '\$' + T$   
 $s \leftarrow \text{ComputePrefixFunction}(S)$   
result  $\leftarrow$  empty list  
for  $i$  from  $|P| + 1$  to  $|S| - 1$ :  
    if  $s[i] == |P|$ :  
        result.Append( $i - 2|P|$ )  
return result
```

Lemma

The running time of Knuth-Morris-Pratt algorithm is $O(|P| + |T|)$.

Proof

- Building string S is $O(|P| + |T|)$

Lemma

The running time of Knuth-Morris-Pratt algorithm is $O(|P| + |T|)$.

Proof

- Building string S is $O(|P| + |T|)$
- Computing prefix function is $O(|S|) = O(|P| + |T|)$

Lemma

The running time of Knuth-Morris-Pratt algorithm is $O(|P| + |T|)$.

Proof

- Building string S is $O(|P| + |T|)$
- Computing prefix function is $O(|S|) = O(|P| + |T|)$
- The for loop runs $O(|S|) = O(|P| + |T|)$ iterations



Conclusion

- Can search pattern in text in linear time
- Can compute prefix function of a string in linear time
- Can enumerate all borders of a string