Exact Matching, Part 2

photophosphorescent

Kyle Gorman (filling in for Steven Bedrick)
CS/EE 5/655, 11/17/14

Plan for today:

Z-algorithm review

Knuth-Morris-Pratt

Boyer-Moore

Definitions: for a string S:

$S[i,j]$ = contiguous substring starting at i and ending at j. $S(i) = S[i,i]$

Definitions: for a string S:

$S[i,j]$ = contiguous substring starting at i and ending at j. $S(i) = S[i,i]$

S = aardvark

Definitions: for a string S:

$S[i,j]$ = contiguous substring starting at i and ending at j. $S(i) = S[i,i]$

S = aardvark

S[2,4] = ard

Definitions: for a string S:

$S[i,j]$ = contiguous substring starting at i and ending at j. $S(i) = S[i,i]$

S = aardvark

S[2,4] = ard   S(4) = d

Definitions: for a string S:

$S[i,j]$ = contiguous substring starting at i and ending at j. $S(i) = S[i,i]$

S = aardvark

S[2,4] = ard   S(4) = d

For $i > 1$, $Z_i(S)$ is the length of the longest prefix of $S[i,|S|]$ that is also a prefix of S.

For $i > 1$, $Z_i(S)$ is the length of the longest prefix of $S[i,|S|]$ that is also a prefix of S.

For $i > 1$, $Z_i(S)$ is the length of the longest prefix of $S[i,|S|]$ that is also a prefix of S.

S = xtpxtd

For $i > 1$, $Z_i(S)$ is the length of the longest prefix of $S[i,|S|]$ that is also a prefix of S.

S = xtpxtd

$S[4,|S|]$ = xtd   xtp$\underline{xtd}$

For $i > 1$, $Z_i(S)$ is the length of the longest prefix of $S[i,|S|]$ that is also a prefix of S.

S = xtpxtd

$S[4,|S|]$ = xtd

xtp$\underline{xtd}$

$Z_4(S) = 2$

$\underline{\text{xt}}$p$\underline{\text{xt}}$d

S = <u>a</u>ardv<u>a</u>rk

S = aardvark

S = $\underline{\text{a}}$ardv$\underline{\text{a}}$rk

$$Z_6(S) = 1$$

$S = \underline{a}ardv\underline{a}rk$

$Z_6(S) = 1$

S = aardvark

$Z_6(S) = 1$

$Z_2(S) = 1$

S = aardvark $\qquad$ S = alfalfa

$Z_6(S) = 1$

$Z_2(S) = 1$

S = <u>a</u>ardv<u>a</u>rk

$Z_6(S) = 1$

$Z_2(S) = 1$

S = <u>alfalfa</u>

$Z_4(S) = 4$

$S = \underline{a}ardv\underline{a}rk$ $\qquad$ $S = \underline{alf}\underline{alfa}$

$Z_6(S) = 1$ $\qquad$ $Z_4(S) = 4$

$Z_2(S) = 1$

$S = \underline{pho}to\underline{pho}s\underline{pho}rescent$

$S = \text{aardvark}$

$Z_6(S) = 1$

$Z_2(S) = 1$

$S = \text{alfalfa}$

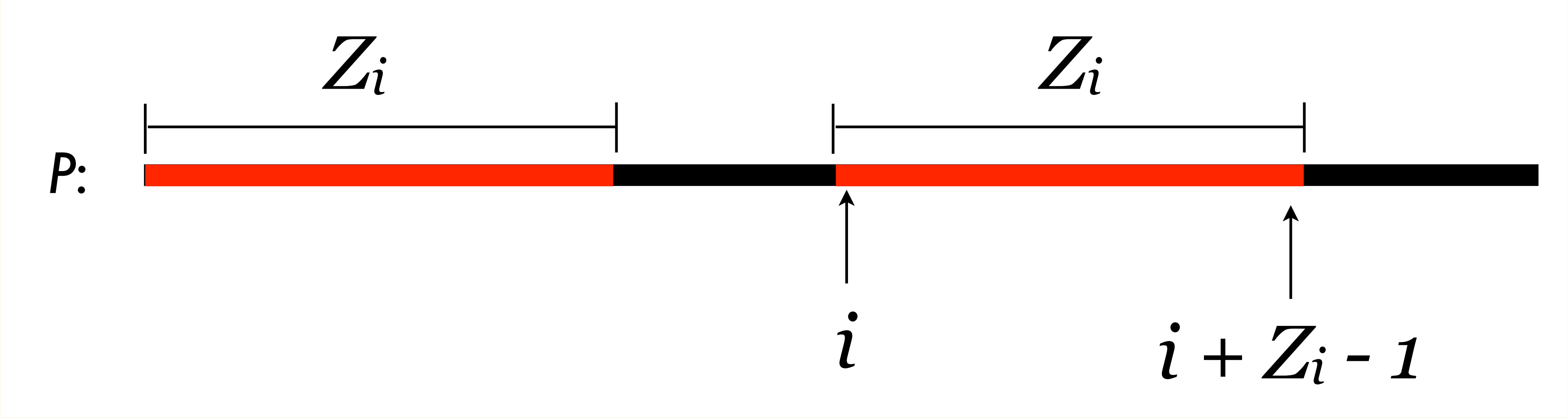
$Z_4(S) = 4$

$S = \text{photophosphorescent}$

$Z_6(S) = Z_{10}(S) = 3$

These regions of prefix-overlap are called *z-boxes*.

P = photo<u>pho</u>s<u>pho</u>rescent



https://www.cs.umd.edu/class/fall2011/cmsc858s/Lec02-zalg.pdf

By being clever about constructing our string, we can easily find exact pattern matches:

p	h	o	$	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
23				3					3				3									

Question: How to calculate Z_i?

By being clever about constructing our string, we can easily find exact pattern matches:

Make the pattern (*P*) the prefix...

p	h	o	$	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
23				3					3				3									

Question: How to calculate Z_i?

By being clever about constructing our string, we can easily find exact pattern matches:

Make the pattern (P) the prefix…

p	h	o	$	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
23				3					3				3									

Question: How to calculate Z_i?

By being clever about constructing our string, we can easily find exact pattern matches:

Make the pattern (*P*) the prefix...

p	h	o	$	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
23				3					3				3									

... and now, any occurrence of *P* is a repeat of the string's prefix, and so has $Z_i = |P|$

Question: How to calculate Z_i?

Question: How to calculate Z_i?

Question: How to calculate Z_i?

The naïve way:

Question: How to calculate Z_i?

The naïve way:

For every position *i*, compute the longest common prefix between *S* and *S[i,|S|]*

Question: How to calculate Z_i?

The naïve way:

For every position *i*, compute the longest common prefix between *S* and *S[i,|S|]*

Problem: This is $O(n^2)$!

Question: How to calculate Z_i?

The naïve way:

For every position *i*, compute the longest common prefix between *S* and *S[i,|S|]*

Problem: This is $O(n^2)$!

The solution involves thinking about the properties of Z-boxes.

At any given i with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:

p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Also, we know that any position k inside a Z-box must correspond to a position k' somewhere in the prefix of S.

At any given *i* with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:

p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
19					3				3									

Also, we know that any position *k* inside a Z-box must correspond to a position *k′* somewhere in the prefix of *S*.

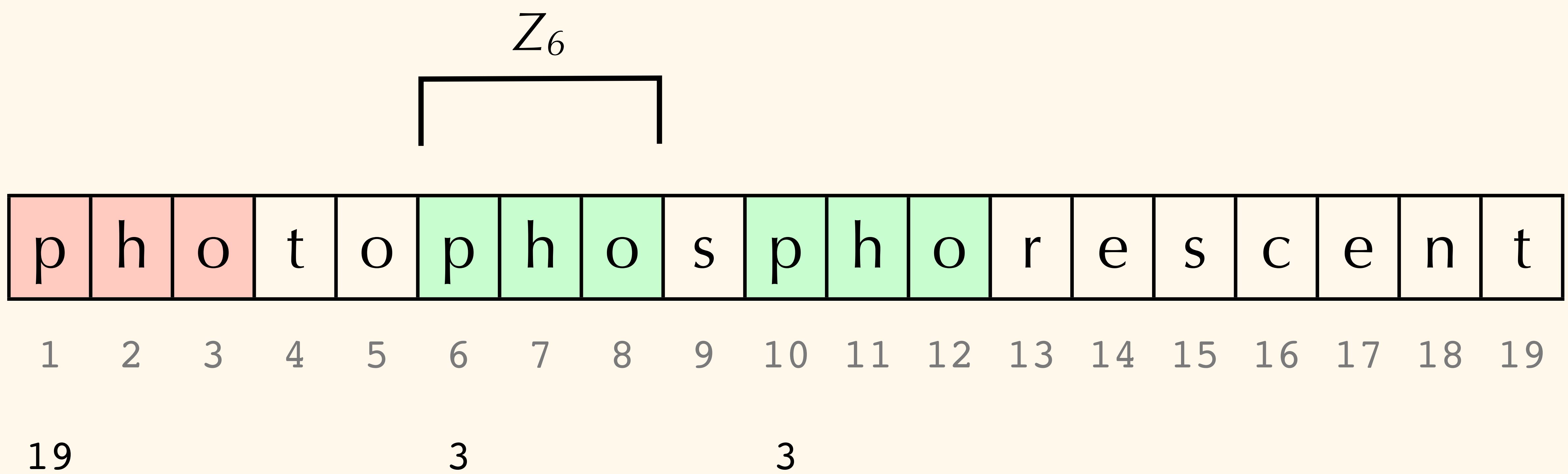
At any given i with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:

p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
19					3				3									

For i = 6: $Z_6(S) = 3$

Also, we know that any position k inside a Z-box must correspond to a position k' somewhere in the prefix of S.

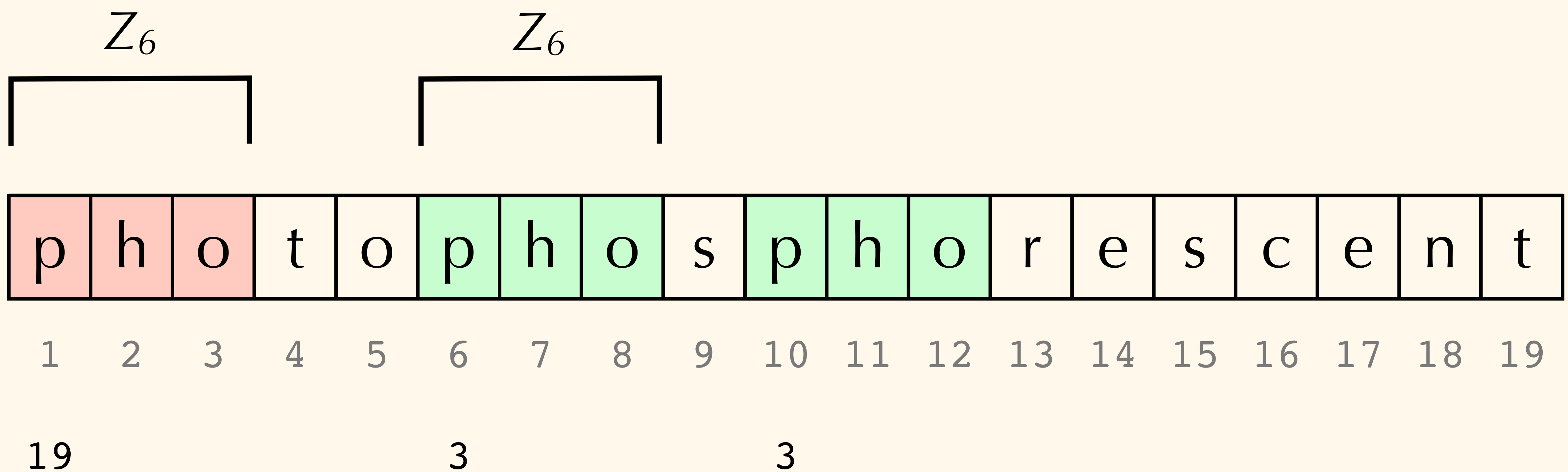
At any given i with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:



For $i = 6$: $Z_6(S) = 3$

Also, we know that any position k inside a Z-box must correspond to a position k' somewhere in the prefix of S.

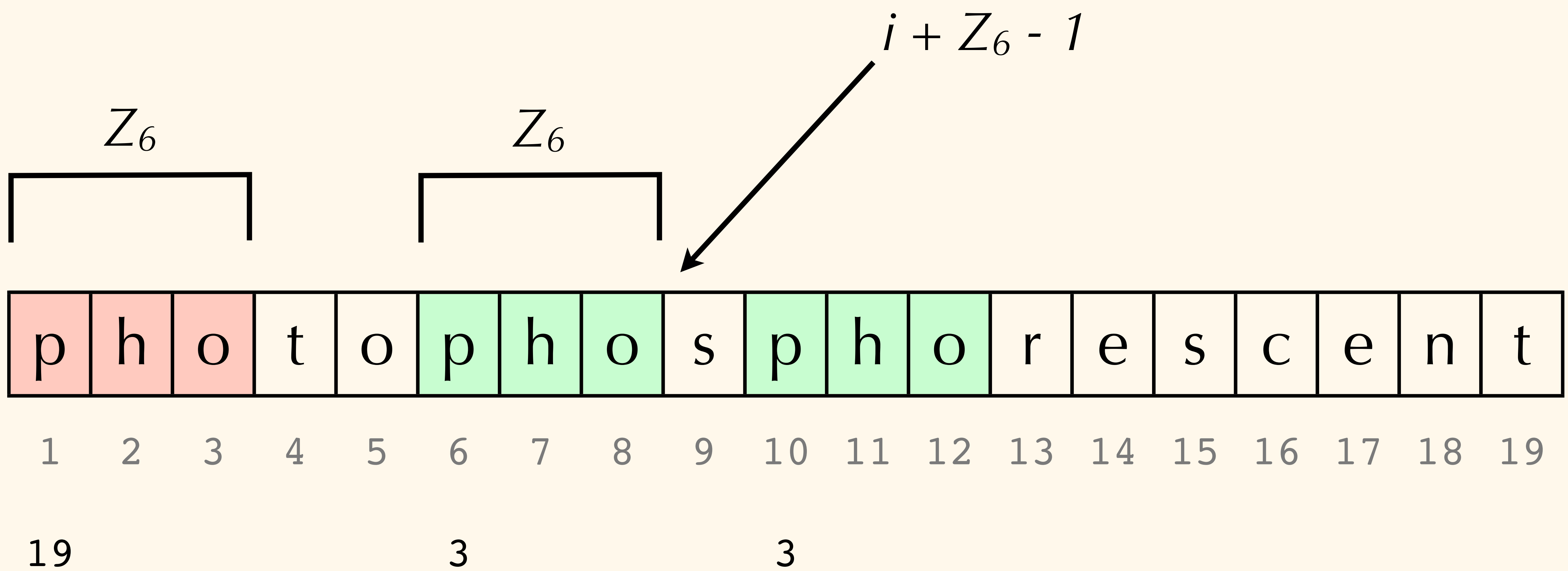
At any given i with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:



For $i = 6$: $Z_6(S) = 3$

Also, we know that any position k inside a Z-box must correspond to a position k' somewhere in the prefix of S.

At any given i with $Z_i > 0$, we can use Z_i to determine the length of its enclosing Z-box:



For i = 6: $Z_6(S) = 3$

Also, we know that any position k inside a Z-box must correspond to a position k' somewhere in the prefix of S.

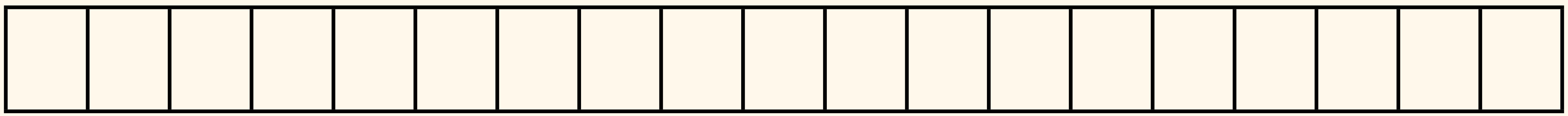
Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

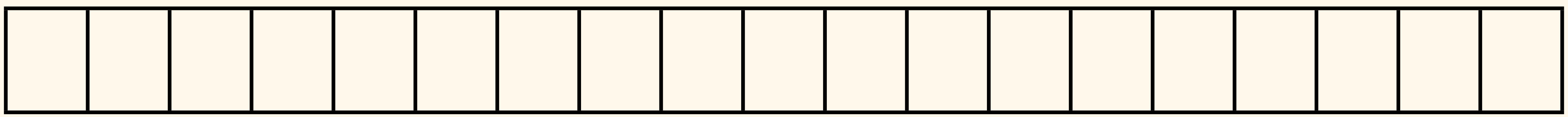
Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$



Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$



i

Z_i

r_i

l_i

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p																		
i	1																		
Z_i	19																		
r_i																			
l_i																			

Additional definitions:

$$r_i = \max_{\substack{1 < j \le i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \le i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h																	
i	1	2																	
Z_i	19																		
r_i		0																	
l_i		0																	

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

i	1	2	3																
	p	h	o																
Z_i	19																		
r_i		0	0																
l_i		0	0																

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t																
i	1	2	3	4																
Z_i	19																			
r_i		0	0	0																
l_i		0	0	0																

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o														
i	1	2	3	4	5														
Z_i	19																		
r_i		0	0	0	0														
l_i		0	0	0	0														

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p
i	1	2	3	4	5	6
Z_i	19					3
r_i		0	0	0	0	8
l_i		0	0	0	0	6

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h												
i	1	2	3	4	5	6	7												
Z_i	19					3													
r_i		0	0	0	0	8	8												
l_i		0	0	0	0	6	6												

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o											
i	1	2	3	4	5	6	7	8											
Z_i	19					3													
r_i		0	0	0	0	8	8	8											
l_i		0	0	0	0	6	6	6											

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o
i	1	2	3	4	5	6	7	8
Z_i	19					3		
r_i		0	0	0	0	8	8	8
l_i		0	0	0	0	6	6	6

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s
i	1	2	3	4	5	6	7	8	9
Z_i	19					3			
r_i		0	0	0	0	8	8	8	8
l_i		0	0	0	0	6	6	6	6

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p
i	1	2	3	4	5	6	7	8	9	10
Z_i	19					3				3
r_i		0	0	0	0	8	8	8	8	12
l_i		0	0	0	0	6	6	6	6	10

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h
i	1	2	3	4	5	6	7	8	9	10	11
Z_i	19					3				3	
r_i		0	0	0	0	8	8	8	8	12	12
l_i		0	0	0	0	6	6	6	6	10	10

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o
i	1	2	3	4	5	6	7	8	9	10	11	12
Z_i	19					3				3		
r_i		0	0	0	0	8	8	8	8	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o
i	1	2	3	4	5	6	7	8	9	10	11	12
Z_i	19					3				3		
r_i		0	0	0	0	8	8	8	8	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

i	1	2	3	4	5	6	7	8	9	10	11	12	13						
	p	h	o	t	o	p	h	o	s	p	h	o	r						
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12						
l_i		0	0	0	0	6	6	6	6	10	10	10	10						

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o	r	e					
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14					
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12					
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10					

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12				
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10				

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

$$l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12			
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10			

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e		
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12		
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10		

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname*{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	

Additional definitions:

$$r_i = \max_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1) \qquad l_i = \operatorname{argmax}_{\substack{1 < j \leq i \\ \text{s.t. } Z_j > 0}} (j + Z_j - 1)$$

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

For any position k where $k > l_k$ (i.e., is inside a z-box), we know that there exists a position k' s.t. $S(k) = S(k')$. $k' = k - l_k + 1$

E.g.: $k = 7$

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

For any position k where $k > l_k$ (i.e., is inside a z-box), we know that there exists a position k' s.t. $S(k) = S(k')$. $k' = k - l_k + 1$

E.g.: $k = 7$ $S(k) =$ "h"

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

For any position k where $k > l_k$ (i.e., is inside a z-box), we know that there exists a position k' s.t. $S(k) = S(k')$. $k' = k - l_k + 1$

E.g.: $k = 7$ $S(k) =$ "h" $l_k = 6$

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

For any position k where $k > l_k$ (i.e., is inside a z-box), we know that there exists a position k' s.t. $S(k) = S(k')$. $k' = k - l_k + 1$

E.g.: $k = 7$ $S(k) =$ “h” $l_k = 6$ $k' = 7\text{-}6\text{+}1 = 2$

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

For any position k where $k > l_k$ (i.e., is inside a z-box), we know that there exists a position k' s.t. $S(k) = S(k')$. $k' = k - l_k + 1$

E.g.: $k = 7$ $S(k) = $ “h” $l_k = 6$ $k' = 7\text{-}6\text{+}1 = 2$

k' is the k's equivalent position in the matching prefix of the string!

What does this get us?

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

k' is the k's equivalent position in the matching prefix of the string!

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

k' is the k's equivalent position in the matching prefix of the string!

Therefore, $Z_{k'}$ can tell us something about the structure of the prefix of the string.

What does this get us?

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z_i	19					3				3									
r_i		0	0	0	0	8	8	8	8	12	12	12	12	12	12	12	12	12	12
l_i		0	0	0	0	6	6	6	6	10	10	10	10	10	10	10	10	10	10

k' is the k's equivalent position in the matching prefix of the string!

Therefore, $Z_{k'}$ can tell us something about the structure of the prefix of the string.

If $Z_{k'} > 0$, there must be repeating elements!

Putting it all together into an algorithm:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ← *i.e., not inside a previously-found matching region*

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ← *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ⟵ *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ⟵ *k is the beginning of a match of length Z_k*

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ← *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ← *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ⟵ *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ⟵ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

We know that β must match $S[k',Z_l]$…

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

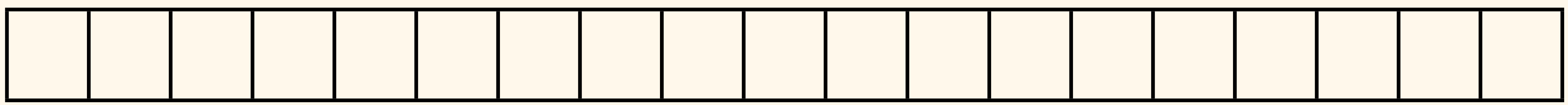
We know that β must match S[k',Z_l]...

β = S[k,r] = S[7,8] = “ho”

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]$

We know that β must match $S[k',Z_l]$...

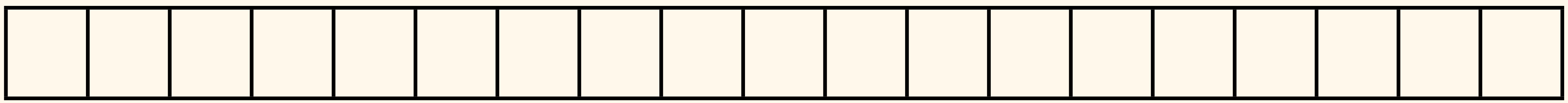


$\beta = S[k,r] = S[7,8] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

We know that β must match $S[k', Z_l]$...



i

Z_i

r_i

l_i

β = S[k,r] = S[7,8] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p																		
i	1																		
Z_i	19																		
r_i																			
l_i																			

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h																	
i	1	2																	
z_i	19																		
r_i		0																	
l_i		0																	

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o
i	1	2	3
z_i	19		
r_i		0	0
l_i		0	0

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t															
i	1	2	3	4															
Z_i	19																		
r_i		0	0	0															
l_i		0	0	0															

β = S[k,r] = S[7,8] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o
i	1	2	3	4	5
Z_i	19				
r_i		0	0	0	0
l_i		0	0	0	0

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p													
i	1	2	3	4	5	6													
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$\beta = S[k,r] = S[7,8] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h
i	1	2	3	4	5	6	7
z_i	19					3	
r_i		0	0	0	0	8	
l_i		0	0	0	0	6	

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

						↓		
	p	h	o	t	o	p	h	o
i	1	2	3	4	5	6	7	8
Z_i	19					3		
r_i		0	0	0	0	8		
l_i		0	0	0	0	6		

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s
i	1	2	3	4	5	6	7	8	9
Z_i	19					3			
r_i		0	0	0	0	8			
l_i		0	0	0	0	6			

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p
i	1	2	3	4	5	6	7	8	9	10
Z_i	19					3				
r_i		0	0	0	0	8				
l_i		0	0	0	0	6				

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h								
i	1	2	3	4	5	6	7	8	9	10	11								
z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o							
i	1	2	3	4	5	6	7	8	9	10	11	12							
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

i	1	2	3	4	5	6	7	8	9	10	11	12	13						
	p	h	o	t	o	p	h	o	s	p	h	o	r						
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e					
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14					
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e		
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

↓

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

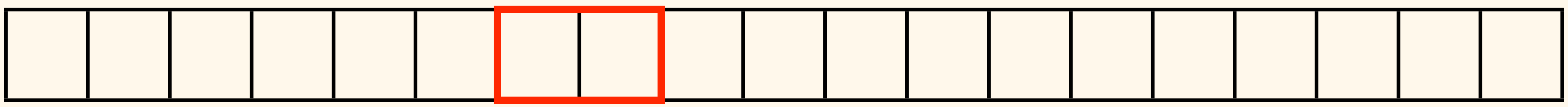
We know that β must match $S[k', Z_l]$...

β = S[k,r] = S[7,8] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]$

We know that β must match $S[k',Z_l]$...

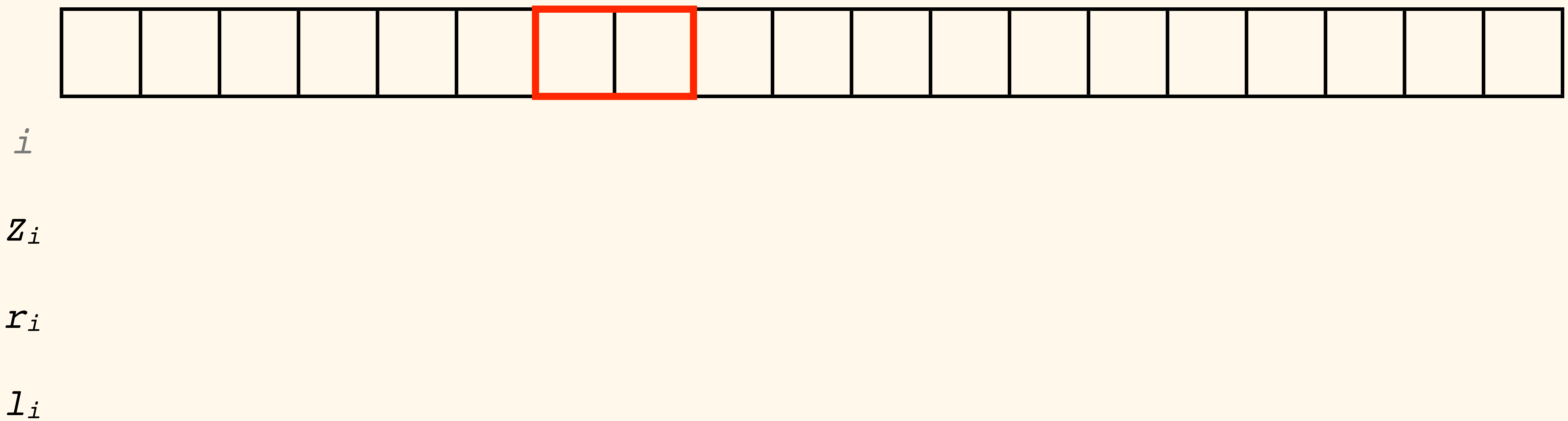


$\beta = S[k,r] = S[7,8] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

We know that β must match S[k',Z_l]...



β = S[k,r] = S[7,8] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p																		
i	1																		
z_i	19																		
r_i																			
l_i																			

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h																	
i	1	2																	
z_i	19																		
r_i		0																	
l_i		0																	

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o																
i	1	2	3																
z_i	19																		
r_i		0	0																
l_i		0	0																

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

We know that β must match S[k',Z_l]...

	p	h	o	t															
i	1	2	3	4															
z_i	19																		
r_i		0	0	0															
l_i		0	0	0															

β = S[k,r] = S[7,8] = “ho”

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o														
i	1	2	3	4	5														
z_i	19																		
r_i		0	0	0	0														
l_i		0	0	0	0														

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p
i	1	2	3	4	5	6
z_i	19					3
r_i		0	0	0	0	8
l_i		0	0	0	0	6

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \text{ ; } \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h
i	1	2	3	4	5	6	7
z_i	19					3	
r_i		0	0	0	0	8	
l_i		0	0	0	0	6	

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o											
i	1	2	3	4	5	6	7	8											
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s										
i	1	2	3	4	5	6	7	8	9										
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

i	1	2	3	4	5	6	7	8	9	10
	p	h	o	t	o	p	h	o	s	p
Z_i	19					3				
r_i		0	0	0	0	8				
l_i		0	0	0	0	6				

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	$\boxed{h}$	$\boxed{o}$	s	p	h
i	1	2	3	4	5	6	7	8	9	10	11
z_i	19					3					
r_i		0	0	0	0	8					
l_i		0	0	0	0	6					

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o
i	1	2	3	4	5	6	7	8	9	10	11	12
z_i	19					3						
r_i		0	0	0	0	8						
l_i		0	0	0	0	6						

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r
i	1	2	3	4	5	6	7	8	9	10	11	12	13
Z_i	19					3							
r_i		0	0	0	0	8							
l_i		0	0	0	0	6							

β = S[k,r] = S[7,8] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14				
z_i	19					3												
r_i		0	0	0	0	8												
l_i		0	0	0	0	6												

$\beta = S[k,r] = S[7,8] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16			
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e		
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \;;\; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

↓

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = S[7,8] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

We know that β must match S[k',Z_l]...

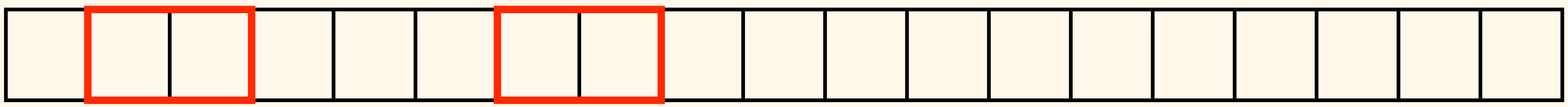
β = S[k,r] = "ho"

S[k',Z_l] = S[7-6+1, 3] = S[2,3] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]$

We know that β must match $S[k',Z_l]$...



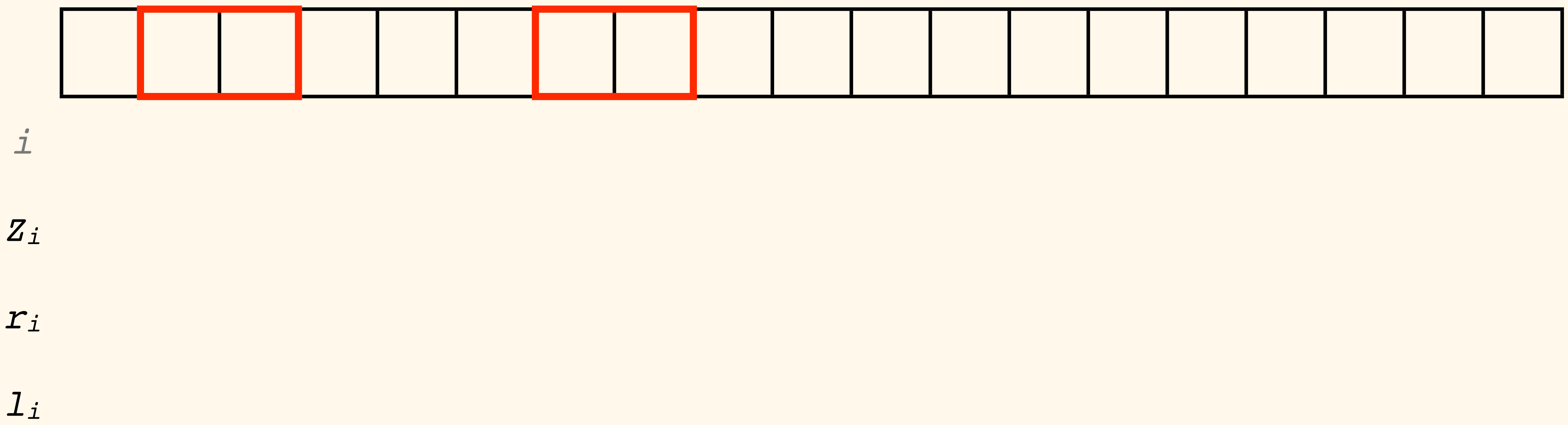
$\beta = S[k,r] =$ "ho"

$S[k',Z_l] = S[7-6+1, 3] = S[2,3] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...



$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7\text{-}6\text{+}1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]

We know that β must match S[k',Z_l]...

	p																		
i	1																		
Z_i	19																		
r_i																			
l_i																			

β = S[k,r] = "ho"

S[k',Z_l] = S[7-6+1, 3] = S[2,3] = "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h																	
i	1	2																	
Z_i	19																		
r_i		0																	
l_i		0																	

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7\text{-}6\text{+}1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o
i	1	2	3
Z_i	19		
r_i		0	0
l_i		0	0

β = S[k,r] = “ho”

S[k',Z_l] = S[7-6+1, 3] = S[2,3] = “ho”

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t															
i	1	2	3	4															
Z_i	19																		
r_i		0	0	0															
l_i		0	0	0															

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7-6+1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o
i	1	2	3	4	5
Z_i	19				
r_i		0	0	0	0
l_i		0	0	0	0

$\beta = S[k,r] =$ "ho"

$S[k',Z_l] = S[7-6+1, 3] = S[2,3] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p													
i	1	2	3	4	5	6													
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k', Z_l] = S[7\text{-}6\text{+}1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h
i	1	2	3	4	5	6	7
Z_i	19					3	
r_i		0	0	0	0	8	
l_i		0	0	0	0	6	

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k', Z_l] = S[7-6+1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p	h	o
i	1	2	3	4	5	6	7	8
Z_i	19					3		
r_i		0	0	0	0	8		
l_i		0	0	0	0	6		

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7\text{-}6\text{+}1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s										
i	1	2	3	4	5	6	7	8	9										
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

β = S[k,r] = “ho”

$S[k',Z_l]$ = S[7-6+1, 3] = S[2,3] = “ho”

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k', Z_l]$...

	p	h	o	t	o	p	h	o	s	p									
i	1	2	3	4	5	6	7	8	9	10									
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k', Z_l] = S[7\text{-}6\text{+}1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h
i	1	2	3	4	5	6	7	8	9	10	11
Z_i	19					3					
r_i		0	0	0	0	8					
l_i		0	0	0	0	6					

$\beta = S[k,r] =$ "ho"

$S[k',Z_l] = S[7\text{-}6+1,\ 3] = S[2,3] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o							
i	1	2	3	4	5	6	7	8	9	10	11	12							
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$\beta = S[k,r] =$ "ho"

$S[k',Z_l] = S[7-6+1, 3] = S[2,3] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r						
i	1	2	3	4	5	6	7	8	9	10	11	12	13						
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

β = S[k,r] = “ho”

$S[k',Z_l]$ = S[7-6+1, 3] = S[2,3] = “ho”

If $k \le r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e					
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14					
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

β = S[k,r] = “ho”

$S[k',Z_l] = S[7\text{-}6+1, 3] = S[2,3]$ = “ho”

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15				
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7\text{-}6\text{+}1,\, 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c
Z_i	19					3										
r_i		0	0	0	0	8										
l_i		0	0	0	0	6										

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7\text{-}6+1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \ ; \ \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	$\boxed{h}$	$\boxed{o}$	t	o	p	$\boxed{h}$	$\boxed{o}$	s	p	h	o	r	e	s	c	e		
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$\beta = S[k,r] =$ "ho"

$S[k',Z_l] = S[7\text{-}6+1,\ 3] = S[2,3] =$ "ho"

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$…

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Z_i	19					3												
r_i		0	0	0	0	8												
l_i		0	0	0	0	6												

$$\beta = S[k,r] = \text{“ho”}$$

$$S[k',Z_l] = S[7-6+1,\ 3] = S[2,3] = \text{“ho”}$$

If $k \leq r$, we *are* inside of an already-found z-box, so:

$$k' = k - l + 1 \; ; \; \beta = S[k,r]$$

We know that β must match $S[k',Z_l]$...

	p	h	o	t	o	p	h	o	s	p	h	o	r	e	s	c	e	n	t
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Z_i	19					3													
r_i		0	0	0	0	8													
l_i		0	0	0	0	6													

$\beta = S[k,r] =$ “ho”

$S[k',Z_l] = S[7\text{-}6+1,\ 3] = S[2,3] =$ “ho”

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = |S[k,r]|$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ← *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ← *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

Two possibilities:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ⟵ *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ⟵ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

Two possibilities:

S(k) could simply be part of a matching substring...

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

Two possibilities:

S(k) could simply be part of a matching substring...

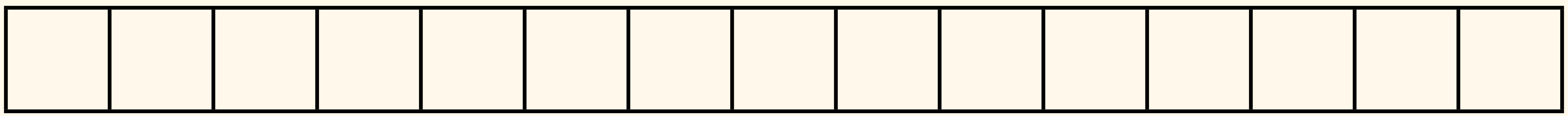
Alternatively, S[k,r] could itself be a matching prefix of S!

Two possibilities:

$S(k)$ could simply be part of a matching substring…

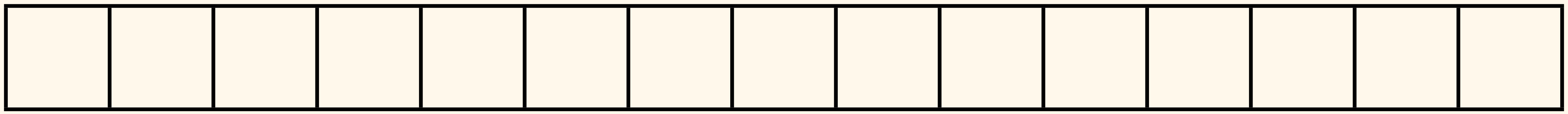
Two possibilities:

$S(k)$ could simply be part of a matching substring…



Two possibilities:

$S(k)$ could simply be part of a matching substring...



i

z_i

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a														
i	1														
z_i	15														

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b													
i	1	2													
z_i	15														

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a												
i	1	2	3												
z_i	15		2												

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a	b											
i	1	2	3	4											
z_i	15		2												

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x										
i	1	2	3	4	5										
z_i	15		2												

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x	a									
i	1	2	3	4	5	6									
z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x	a	b								
i	1	2	3	4	5	6	7								
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a	b	x	a	b	a							
i	1	2	3	4	5	6	7	8							
z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring…

↓ (above position 7)

	a	b	a	b	x	a	b	a	b						
i	1	2	3	4	5	6	7	8	9						
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a					
i	1	2	3	4	5	6	7	8	9	10					
z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y				
i	1	2	3	4	5	6	7	8	9	10	11				
z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y	a			
i	1	2	3	4	5	6	7	8	9	10	11	12			
Z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y	a	b		
i	1	2	3	4	5	6	7	8	9	10	11	12	13		
z_i	15		2			4									

Two possibilities:

S(*k*) could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Z_i	15		2			4									

Two possibilities:

$S(k)$ could simply be part of a matching substring...

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring...

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

In this case, we know that the characters between S(k) and S(r) are all part of a single match...

Two possibilities:

S(k) could simply be part of a matching substring...

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		2			4									

In this case, we know that the characters between S(k) and S(r) are all part of a single match...

... and so:

- Z_k must be equal to $Z_{k'}$, and therefore...
- we don't need to update l and r

Two possibilities:

$S(k)$ could simply be part of a matching substring...

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring…

							↓								
	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

Two possibilities:

$S(k)$ could simply be part of a matching substring…

↓

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		2			4									

Alternatively, $S[k,r]$ could itself be a matching prefix of S!

↓

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		2			4									

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

In this case, we are in an *overlapping* z-box...

Alternatively, S[k,r] could itself be a matching prefix of S!

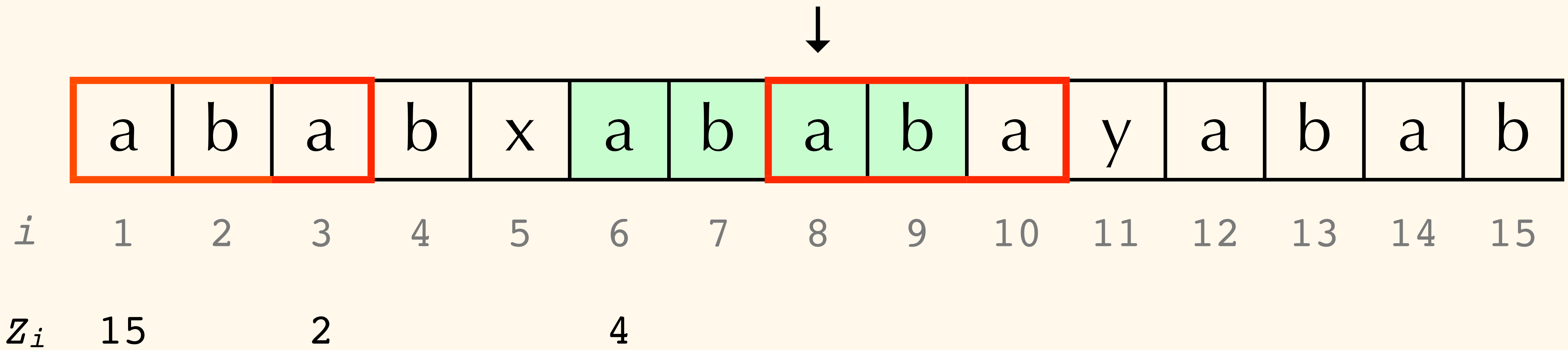
↓

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

In this case, we are in an *overlapping* z-box...

... so Z_k might be potentially end up being different from $Z_{k'}$...

Alternatively, S[k,r] could itself be a matching prefix of S!



In this case, we are in an *overlapping* z-box...

... so Z_k might be potentially end up being different from $Z_{k'}$...

... and we might need to update l and r (to reflect the boundaries of the *new* z-box).

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ⟵ *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ⟵ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

Two possibilities:

S(k) could simply be part of a matching substring...

Alternatively, S[k,r] could itself be a matching prefix of S!

How to tell which condition? Compare $Z_{k'}$ to $|\beta|$:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ← *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ← *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = |S[k,r]|$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: ⟵ *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ ⟵ *k is the beginning of a match of length* Z_k

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

If $Z_{k'} \geq |\beta|$:

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

If $Z_{k'} \geq |\beta|$:

We know that Z_k must be at least $Z_{k'}$ - but it could be longer...

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

  Calculate Z_k the normal way

  If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

  $k' = k - l + 1$; $\beta = S[k,r]|$

  If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

  If $Z_{k'} \geq |\beta|$:

    We know that Z_k must be at least $Z_{k'}$ - but it could be longer...

    Start looking for a match between $S[r + 1,]$ and $S[|\beta| + 1,]$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; $\beta = S[k,r]|$

k is inside a z-box, but is not the start of an overlapping box

If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

If $Z_{k'} \geq |\beta|$:

We know that Z_k must be at least $Z_{k'}$ - but it could be longer...

Start looking for a match between $S[r + 1,]$ and $S[|\beta| + 1,]$

Putting it all together into an algorithm:

Initialize l and r to 0; for each k, $1 < k \leq |S|$:

If $k > r$, we are not in a z-box, so: *i.e., not inside a previously-found matching region*

Calculate Z_k the normal way

If $Z_k > 0$, set $l = k$ and $r = k + Z_k - 1$ *k is the beginning of a match of length Z_k*

If $k \leq r$, we *are* inside of an already-found z-box, so:

$k' = k - l + 1$; β = S[k,r]| *k is inside a z-box, but is not the start of an overlapping box*

If $Z_{k'} < |\beta|$, $Z_k = Z_{k'}$ and l, r are unchanged;

If $Z_{k'} \geq |\beta|$: *k must itself be the beginning of a new, overlapping z-box*

We know that Z_k must be at least $Z_{k'}$ - but it could be longer…

Start looking for a match between S[r + 1,] and S[|β| + 1,]

Two possibilities:

S(k) could simply be part of a matching substring...

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		2			4									

Two possibilities:

S(k) could simply be part of a matching substring…

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		2			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		***2***			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

In this case, we know that Z_k will be at least 2…

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

In this case, we know that Z_k will be at least 2…

… and so we'll need to do prefix matching, but we *don't* have to start at the beginning of the string.

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

In this case, we know that Z_k will be at least 2...

... and so we'll need to do prefix matching, but we *don't* have to start at the beginning of the string.

We know that S[*k*,*r*] *must* match S[1,$Z_{k'}$], so can use our current *r* offset to tell us where to start looking for a continued match.

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

We know that S[*k*,*r*] *must* match S[1,$Z_{k'}$], so can use our current *r* offset to tell us where to start looking for a continued match.

Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

We know that S[k,r] *must* match S[1,$Z_{k'}$], so can use our current r offset to tell us where to start looking for a continued match.

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		***2***			4									

We know that S[*k*,*r*] *must* match S[1,$Z_{k'}$], so can use our current *r* offset to tell us where to start looking for a continued match.

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
z_i	15		***2***			4									

Alternatively, S[k,r] could itself be a matching prefix of S!

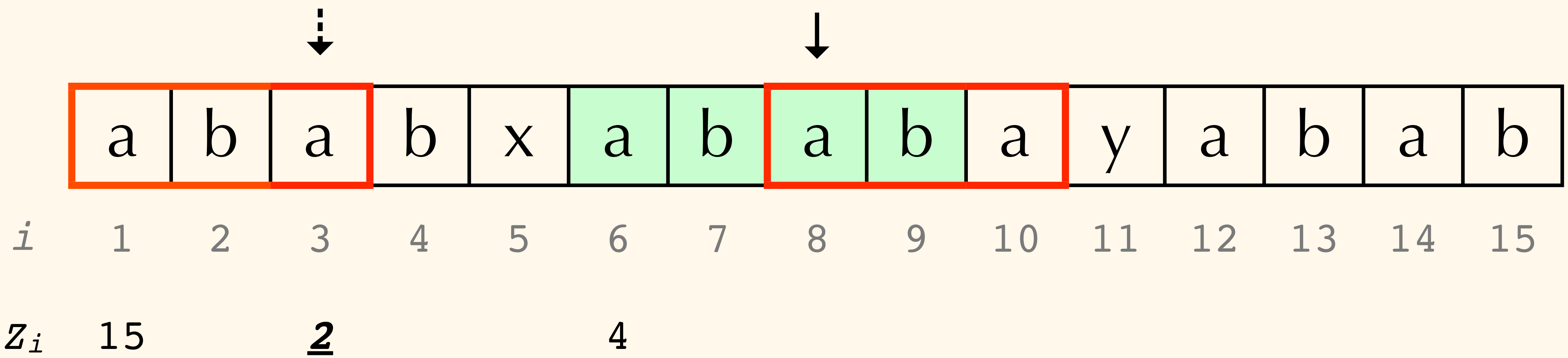
	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		***2***			4									

We know that S[k,r] *must* match S[1,$Z_{k'}$], so can use our current r offset to tell us where to start looking for a continued match.

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.

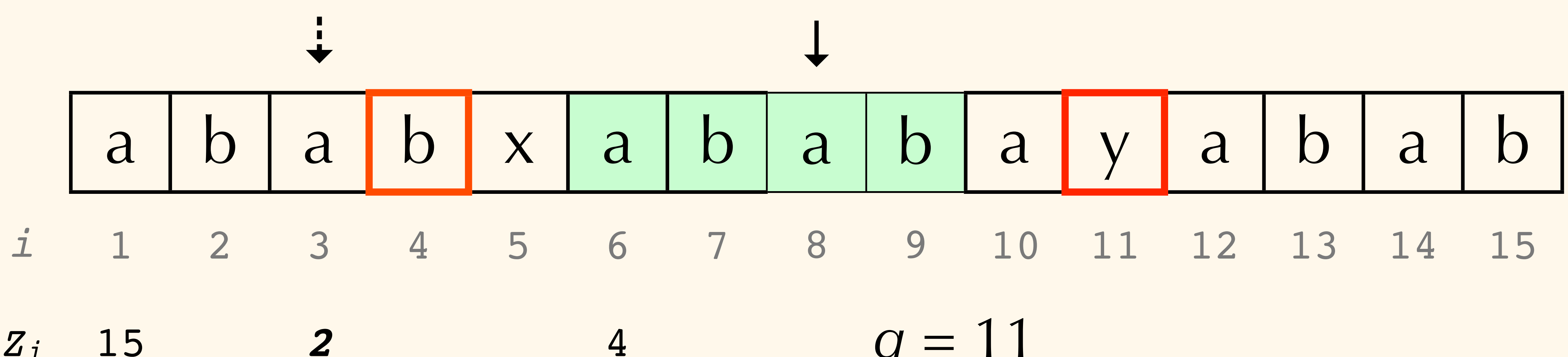
	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		***2***			4									

Alternatively, S[*k*,*r*] could itself be a matching prefix of S!



We know that S[*k*,*r*] *must* match S[1,$Z_{k'}$], so can use our current *r* offset to tell us where to start looking for a continued match.

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.



Alternatively, S[k,r] could itself be a matching prefix of S!

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

We know that S[k,r] *must* match S[1,$Z_{k'}$], so can use our current r offset to tell us where to start looking for a continued match.

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.

$q = 11$

We can now set $Z_k = q - k$; the new $r = q - 1$; and $l = k$

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4									

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs q.

$q = 11$

We can now set $Z_k = q - k$; the new $r = q - 1$; and $l = k$

	a	b	a	b	x	a	b	a	b	a	y	a	b	a	b
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z_i	15		**2**			4		3							

Start comparing S[r+1,] and S[|β|+1,]; call the position in S where the first mismatch occurs *q*.

$q = 11$

We can now set $Z_k = q - k$; the new $r = q - 1$; and $l = k$

Now, continue on to the next *k*...

What does this get us?

What does this get us?

A lot, in the *worst* case:

What does this get us?

A lot, in the *worst* case:

(a long, repetitive pattern, with a string full of matches or near-matches)

What does this get us?

A lot, in the *worst* case:

(a long, repetitive pattern, with a string full of matches or near-matches)

We get to “skip ahead” quite often in that case.

What does this get us?

A lot, in the *worst* case:

(a long, repetitive pattern, with a string full of matches or near-matches)

We get to “skip ahead” quite often in that case.

In a more “normal” case.... meh.

What does this get us?

A lot, in the *worst* case:

(a long, repetitive pattern, with a string full of matches or near-matches)

We get to “skip ahead” quite often in that case.

In a more “normal” case.... meh.

Less skipping == less (relative) benefit

Plan for today:

Z-algorithm review

Knuth-Morris-Pratt

Boyer-Moore

Historical notes:

Donald Knuth
1938 –

Vaughan Pratt
1944 –

James Morris
1941 –

The KMP algorithm was discovered in 1974 by K & P at Stanford, and independently by M at CMU in that same year.

The three authors formally published together in 1977.

D. E. Knuth, J. H. Morris, and V. B. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6:323-50, 1977.

The Knuth-Morris-Pratt algorithm builds on top of the Z-algorithm…

… the main innovation being that rather than moving through S one character at a time…

… we use information about repetitive segments of P to help us move more quickly.

For example, (from Gusfield):

P =

	a	b	c	x	a	b	c	d	e
i	1	2	3	4	5	6	7	8	9

Consider a mismatch between P and S occurring at position 8 in P:

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P					a	b	c	x	a	b	c	d	e							
i					1	2	3	4	5	6	7	8	9							

The previous algorithm would shift P down by one position, and then resume searching for a match:

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P					a	b	c	x	a	b	c	d	e							
				i	1	2	3	4	5	6	7	8	9							

The previous algorithm would shift P down by one position, and then resume searching for a match:

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P						a	b	c	x	a	b	c	d	e						
					i	1	2	3	4	5	6	7	8	9						
						↑														

S

P

The previous algorithm would shift *P* down by one position, and then resume searching for a match:

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P						a	b	c	x	a	b	c	d	e						
					i	1	2	3	4	5	6	7	8	9						
						↑														

However, looking at the pattern, we can see that we *could* have shifted further without missing anything!

S

P

The previous algorithm would shift *P* down by one position, and then resume searching for a match:

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P						a	b	c	x	a	b	c	d	e						
					i	1	2	3	4	5	6	7	8	9						
						↑														

However, looking at the pattern, we can see that we *could* have shifted further without missing anything!

S	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
P									a	b	c	x	a	b	c	d	e			
								i	1	2	3	4	5	6	7	8	9			
									↑											

Definitions:

$sp_i(P)$: The length of the longest *suffix* of P[1,*i*] that is also a *prefix* of P. (and let $sp_0 = 0$)

	x	t	p	x	t	d
i	1	2	3	4	5	6
Z_i	19	0	0	2	0	0
sp_i	0	0	0	0	2	0

"Failure Function": $F(i)$: $sp_{i-1} + 1$

Definitions:

$sp_i(P)$: The length of the longest *suffix* of P[1,*i*] that is also a *prefix* of P. (and let $sp_0 = 0$)

	x	t	p	x	t	d	
i	1	2	3	4	5	6	7
Z_i	19	0	0	2	0	0	
sp_i	0	0	0	0	2	0	
$F(i)$	1	1	1	1	1	3	1

"Failure Function": $F(i)$: $sp_{i-1} + 1$

Computing $\text{sp}_i(P)$ can be done in linear time, using a modification of the Z-algorithm:

Initialize $\text{sp}_i(P) = 0$ for all $P(i)$

From left-to-right over P, hence at position k we have already calculated Z_{k-1} and have current values for l and r

- If $k > r$, begin comparing with beginning of P. Length of match is Z_k. If $Z_k > 0$, then $r = k + Z_k - 1$ and $l = k$.
  - If $\text{sp}_r(P) = 0$ then set $\text{sp}_r(P) = Z_k$
- If $k \leq r$, then $P(k) = P(k')$ where $k' = k - l - 1$

  Further, $P[k, r] = P[k', Z_l]$

  Thus, $Z_k \geq \min(Z_{k'}, |P[k, r]|)$
- If $Z_{k'} < |P[k, r]|$, then $Z_k = Z_{k'}$ and r, l unchanged
- If $Z_{k'} > |P[k, r]|$, then $Z_k = |P[k, r]|$ and r, l unchanged
- Otherwise, begin comparing position $r + 1$ with $|P[k, r]| + 1$

  If mismatch at position q, then $Z_k = q - k$, $l = k$, $r = q - 1$
  - If $\text{sp}_r(P) = 0$ then set $\text{sp}_r(P) = Z_k$

(See Gusfield for more gory details)

i	1	2	3	4	5	6	7
	x	t	p	x	t	d	
Z_i	19	0	0	2	0	0	
sp_i	0	0	0	0	2	0	
$F(i)$	1	1	1	1	1	3	1

We will use F(i) to tell us how far we can safely shift *P* along *S* when we encounter a mis-match.

	x	t	p	x	t	d	
i	1	2	3	4	5	6	7
Z_i	19	0	0	2	0	0	
sp_i	0	0	0	0	2	0	
$F(i)$	1	1	1	1	1	3	1

We will use F(i) to tell us how far we can safely shift *P* along *S* when we encounter a mis-match.

Basic idea: if we encounter a mismatch at character *i* of *P*, we can shift *P* down F(i) positions along *S*.

Demo (roark_kmp.pdf)

Notes on KMP:

Notes on KMP:

KMP is a classic and widely-known linear time exact-match algorithm...

Notes on KMP:

KMP is a classic and widely-known linear time exact-match algorithm...

... and gives a nice speedup over the “simple” linear-time algorithm...

Notes on KMP:

KMP is a classic and widely-known linear time exact-match algorithm...

... and gives a nice speedup over the "simple" linear-time algorithm...

... but for many situations, it is *not* the method of choice.

Notes on KMP:

KMP is a classic and widely-known linear time exact-match algorithm...

... and gives a nice speedup over the "simple" linear-time algorithm...

... but for many situations, it is *not* the method of choice.

The Boyer-Moore algorithm gives better *typical* performance.

Historical Notes:

Bob Boyer

J Strother Moore

The Boyer-Moore algorithm was developed while BB was at SRI and JSM was at Xerox PARC, and was published in 1977.

Fun fact: Moore's first name is, in fact, the alphabetic letter "J" – it's not an abbreviation.

Boyer, R. S., and Moore, J. S. A fast string searching algorithm. Commun. ACM 20, 10 (Oct. 1977), 762–772.

Boyer-Moore improves over KMP to give sub-linear typical performance, and linear worst-case.

Key ideas:

Boyer-Moore improves over KMP to give sub-linear typical performance, and linear worst-case.

Key ideas:

1. We still move the pattern (a.k.a. the “needle”) from *left* to *right* through the string (the “haystack”)...

Boyer-Moore improves over KMP to give sub-linear typical performance, and linear worst-case.

Key ideas:

1. We still move the pattern (a.k.a. the “needle”) from *left* to *right* through the string (the “haystack”)...

   ... but we start searching for matching characters from the *right* of the pattern!

Boyer-Moore improves over KMP to give sub-linear typical performance, and linear worst-case.

Key ideas:

1. We still move the pattern (a.k.a. the "needle") from *left* to *right* through the string (the "haystack")...

   ... but we start searching for matching characters from the *right* of the pattern!

2. If a mis-matched character *T* never occurs in *P*, we shift *P* completely past that character.

Boyer-Moore improves over KMP to give sub-linear typical performance, and linear worst-case.

Key ideas:

1. We still move the pattern (a.k.a. the "needle") from *left* to *right* through the string (the "haystack")...

   ... but we start searching for matching characters from the *right* of the pattern!

2. If a mis-matched character *T* never occurs in *P*, we shift *P* completely past that character.

3. We calculate an optimal shift amount (as in KMP), but we use suffixes rather than prefixes.

Caveat: Some of the Boyer-Moore pre-processing steps can be tricky to get one’s head around.

The explanation in Chapter 2 of Gusfield is very decent, and you will ***need*** to spend some time working through it to fully grok the algorithm.

Definitions:

$R(x) =$

Definitions:

“Bad character rule”:

$R(x) =$

Definitions:

"Bad character rule":

For each character x in P...

$R(x) =$

Definitions:

"Bad character rule":

For each character x in P...

$R(x) =$ the position of the right-most occurrence of x in P

Definitions:

"Bad character rule":

For each character *x* in *P*...

R(*x*) = the position of the right-most occurrence of *x* in *P*

	x	t	p	x	t	d
i	1	2	3	4	5	6

Definitions:

"Bad character rule":

For each character *x* in *P*...

R(*x*) = the position of the right-most occurrence of *x* in *P*

R(x) =	4
R(t) =	5
R(p) =	3
R(d) =	6

	x	t	p	x	t	d
i	1	2	3	4	5	6

“Bad character rule”:

$R(x) = 4$

$R(t) = 5$

$R(p) = 3$

$R(d) = 6$

	x	t	p	x	t	d
i	1	2	3	4	5	6

"Bad character rule":

$R(x) = 4$
$R(t) = 5$
$R(p) = 3$
$R(d) = 6$

i	1	2	3	4	5	6
	x	t	p	x	t	d

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by $\max(1, i - R(T(k)))$

"Bad character rule":

R(x)	=	4
R(t)	=	5
R(p)	=	3
R(d)	=	6

i	1	2	3	4	5	6
	x	t	p	x	t	d

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

Furthermore, if T(*k*) does not appear in *P*, shift *P* by |*P*| (since there's no way that a match could involve *k*.

“Bad character rule”:

$R(x) = 4$
$R(t) = 5$
$R(p) = 3$
$R(d) = 6$

	x	t	p	x	t	d
i	1	2	3	4	5	6

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by $\max(1, i - R(T(k)))$

Furthermore, if $T(k)$ does not appear in *P*, shift *P* by $|P|$ (since there’s no way that a match could involve *k*.

Intuition: Since we’re matching from right-to-left, R(x) tells us the first place there could *possibly* be a match.

R(a) = 5

R(b) = 6

R(c) = 7

R(x) = 4

R(d) = 8

R(e) = 9

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

m o q j a b c x a b c j q z r a a j o q

a	b	c	x	a	b	c	d	e

i 1 2 3 4 5 6 7 8 9

↑

“a” and “e” don’t match; the next *possible* location in *S* that a match *could* occur would involve position R(a) of the pattern being aligned with the current index.

$R(a) = 5$

$R(b) = 6$

$R(c) = 7$

$R(x) = 4$

$R(d) = 8$

$R(e) = 9$

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

	m	o	q	j	a	b	c	x	a	b	c	j	q	z	r	a	a	j	o	q
	a	b	c	x	a	b	c	d	e											
i	1	2	3	4	5	6	7	8	9											
									↑											

$R(T(k)) = R(T(9)) = R(a) = 5$

We shift the pattern (and the search index!) by *i* - *5* = 9 - 5 = 4 positions.

R(a) = 5

R(b) = 6

R(c) = 7

R(x) = 4

R(d) = 8

R(e) = 9

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

m o q j a b c x a b c j q z r a a j o q

	a	b	c	x	a	b	c	d	e
i	1	2	3	4	5	6	7	8	9
									↑

R(T(*k*)) = R(T(9)) = R(a) = 5

We shift the pattern (and the search index!) by *i* - *5* = 9 - 5 = 4 positions.

R(a) = 5

R(b) = 6

R(c) = 7

R(x) = 4

R(d) = 8

R(e) = 9

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

m o q j a b c x a b c j q z r a a j o q

	a	b	c	x	a	b	c	d	e
i	1	2	3	4	5	6	7	8	9

↑

Then, resume searching from the right-hand side of the pattern moving to the left.

R(a) = 5

R(b) = 6

R(c) = 7

R(x) = 4

R(d) = 8

R(e) = 9

If a mismatch occurs between character *i* of *P* and *k* of *T*, shift *P* by max(1, *i* - R(T(*k*)))

m o q j a b c x a b c j q z r a a j o q

	a	b	c	x	a	b	c	d	e
i	1	2	3	4	5	6	7	8	9

↑

Note that, in this case, our next shift would be quite large, since “q” does not appear in the pattern!

“Bad character rule”:

According to Gusfield, just using the bad suffix rule on its own yields good performance on “normal” English...

“Bad character rule”:

According to Gusfield, just using the bad suffix rule on its own yields good performance on “normal” English...

... but in some situations (e.g. small alphabets), it is less effective. Question for discussion: why?

"Bad character rule":

According to Gusfield, just using the bad suffix rule on its own yields good performance on "normal" English...

... but in some situations (e.g. small alphabets), it is less effective. Question for discussion: why?

To assist, Boyer-Moore defines the "Good suffix" rule.

"The original preprocessing method [278] for the strong good suffix rule is generally considered quite difficult and somewhat mysterious (although a weaker version of it is easy to understand). In fact, the preprocessing for the strong rule was given incorrectly in [278] and corrected, without much explanation, in [384]. Code based on [384] is given without real explanation in the text by Baase [32], but there are no published sources that try to fully explain the method."...

Gusfield, p. 19

"The original preprocessing method [278] for the strong good suffix rule is generally considered quite difficult and somewhat mysterious (although a weaker version of it is easy to understand). In fact, the preprocessing for the strong rule was given incorrectly in [278] and corrected, without much explanation, in [384]. Code based on [384] is given without real explanation in the text by Baase [32], but there are no published sources that try to fully explain the method."...

"In contrast, the fundamental preprocessing of P discussed in Chapter 1 makes the needed preprocessing very simple."

Gusfield, p. 19

Intuition of good suffix rule:

Intuition of good suffix rule:

When we hit a mismatch, we want to shift our pattern further along the string.

Intuition of good suffix rule:

When we hit a mismatch, we want to shift our pattern further along the string.

If we've found a partial match, and our pattern contains repeated elements...

Intuition of good suffix rule:

When we hit a mismatch, we want to shift our pattern further along the string.

If we've found a partial match, and our pattern contains repeated elements...

... we can shift our pattern down until the right-most repeated element* to the left of our current position aligns with the current partial match.

Intuition of good suffix rule:

When we hit a mismatch, we want to shift our pattern further along the string.

If we've found a partial match, and our pattern contains repeated elements...

... we can shift our pattern down until the right-most repeated element* to the left of our current position aligns with the current partial match.

In many cases, this will be a further shift than the bad-character rule would have given us!

Intuition of good suffix rule:

When we hit a mismatch, we want to shift our pattern further along the string.

If we've found a partial match, and our pattern contains repeated elements...

... we can shift our pattern down until the right-most repeated element* to the left of our current position aligns with the current partial match.

In many cases, this will be a further shift than the bad-character rule would have given us!

*: Some conditions apply, stay tuned...

Illustration from Gusfield

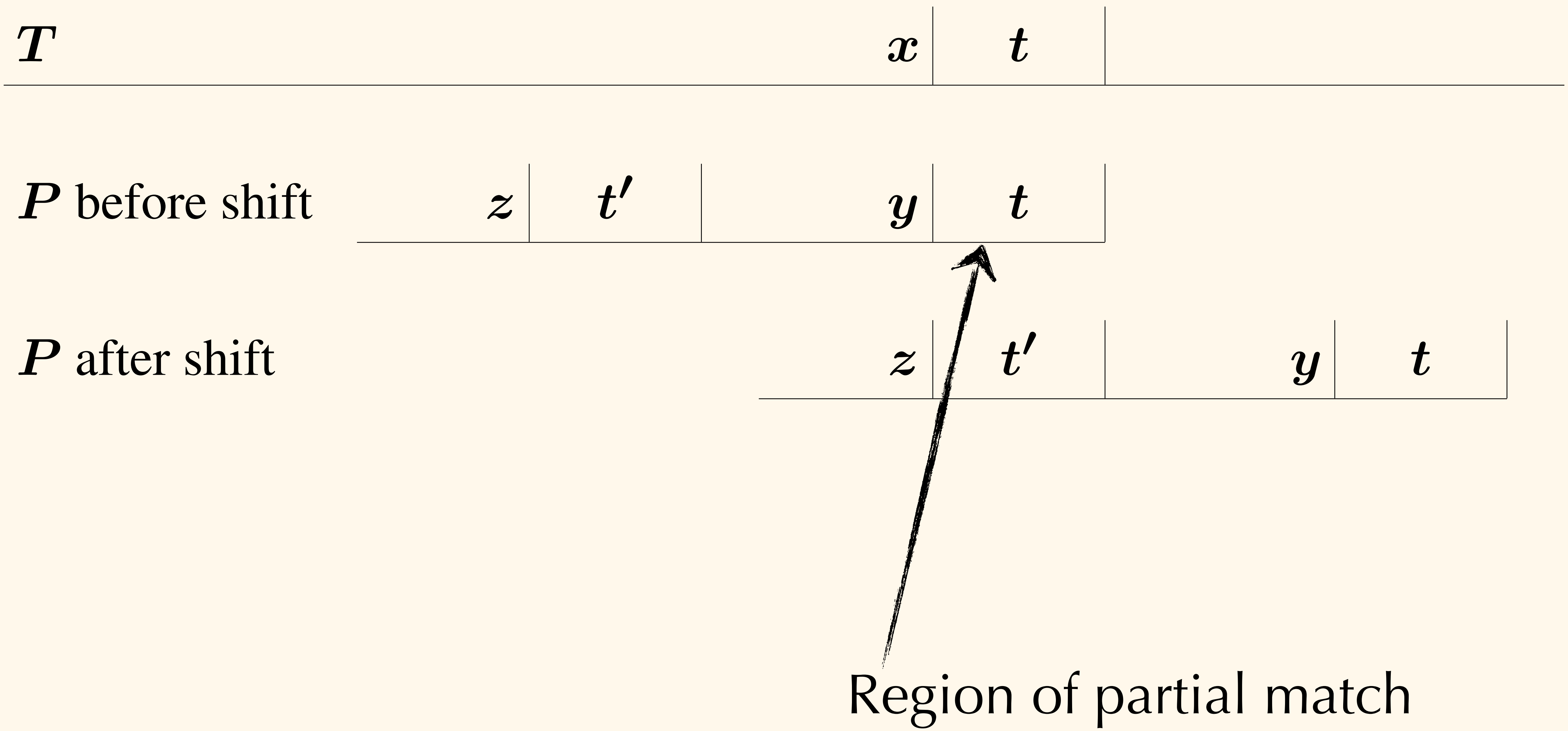


Illustration from Gusfield

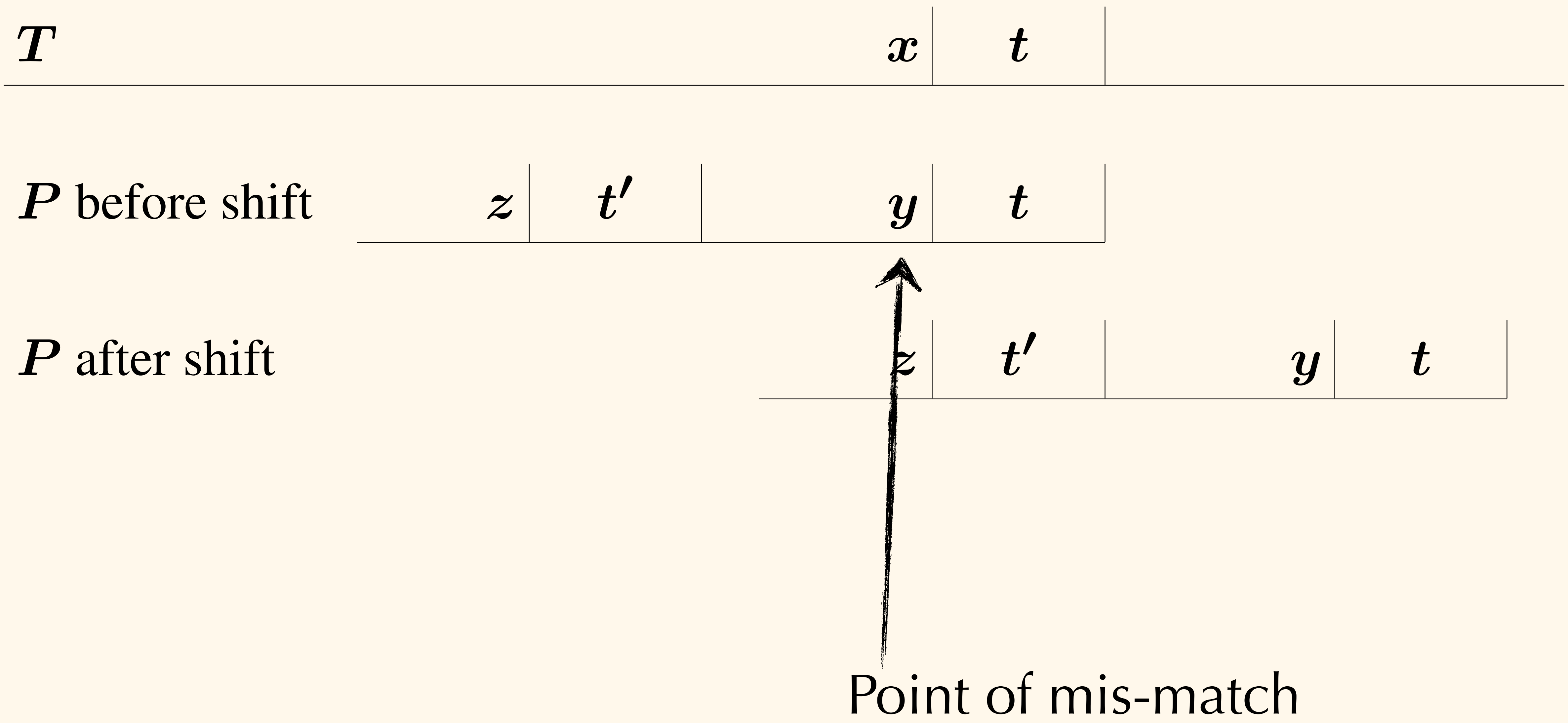


Illustration from Gusfield

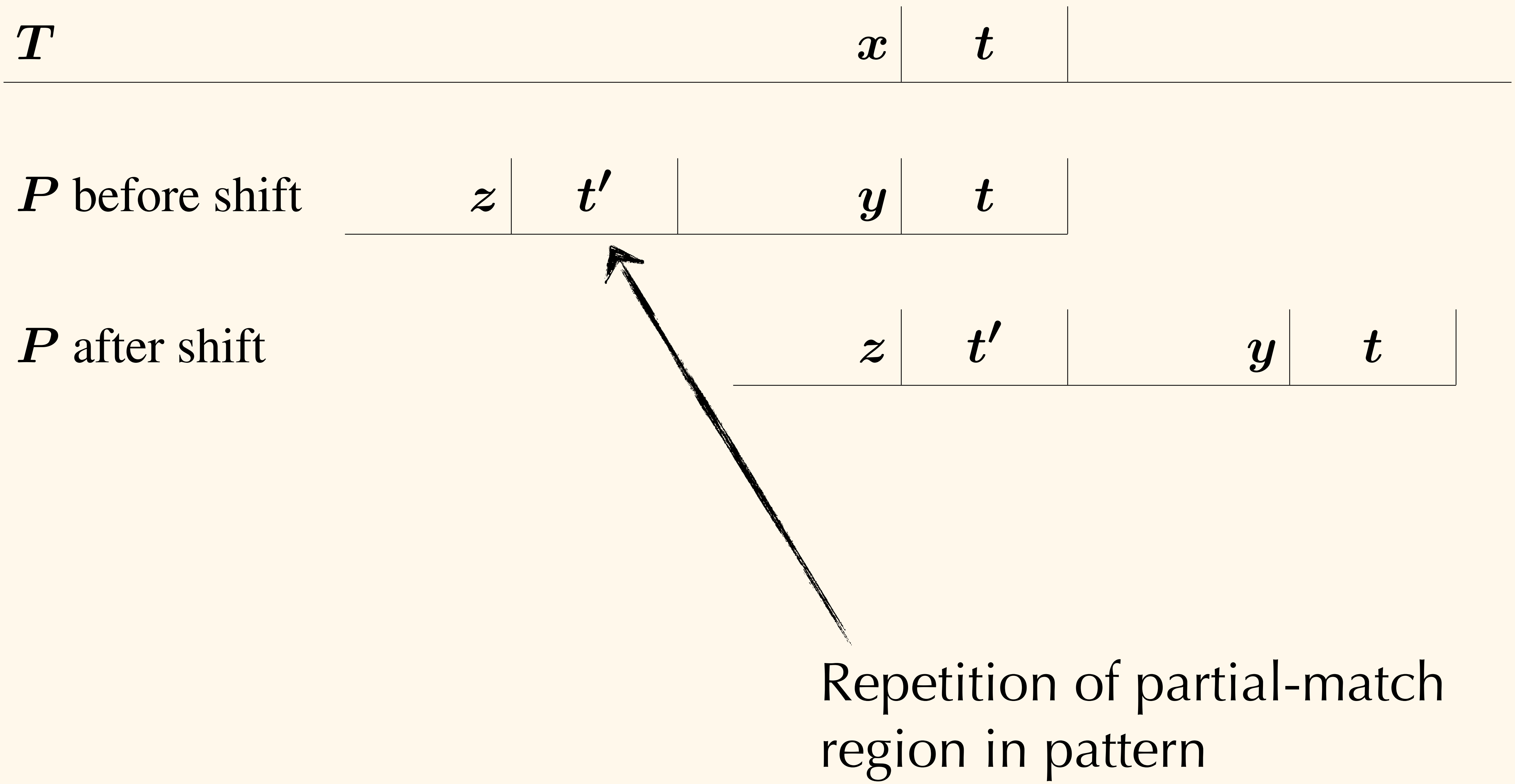
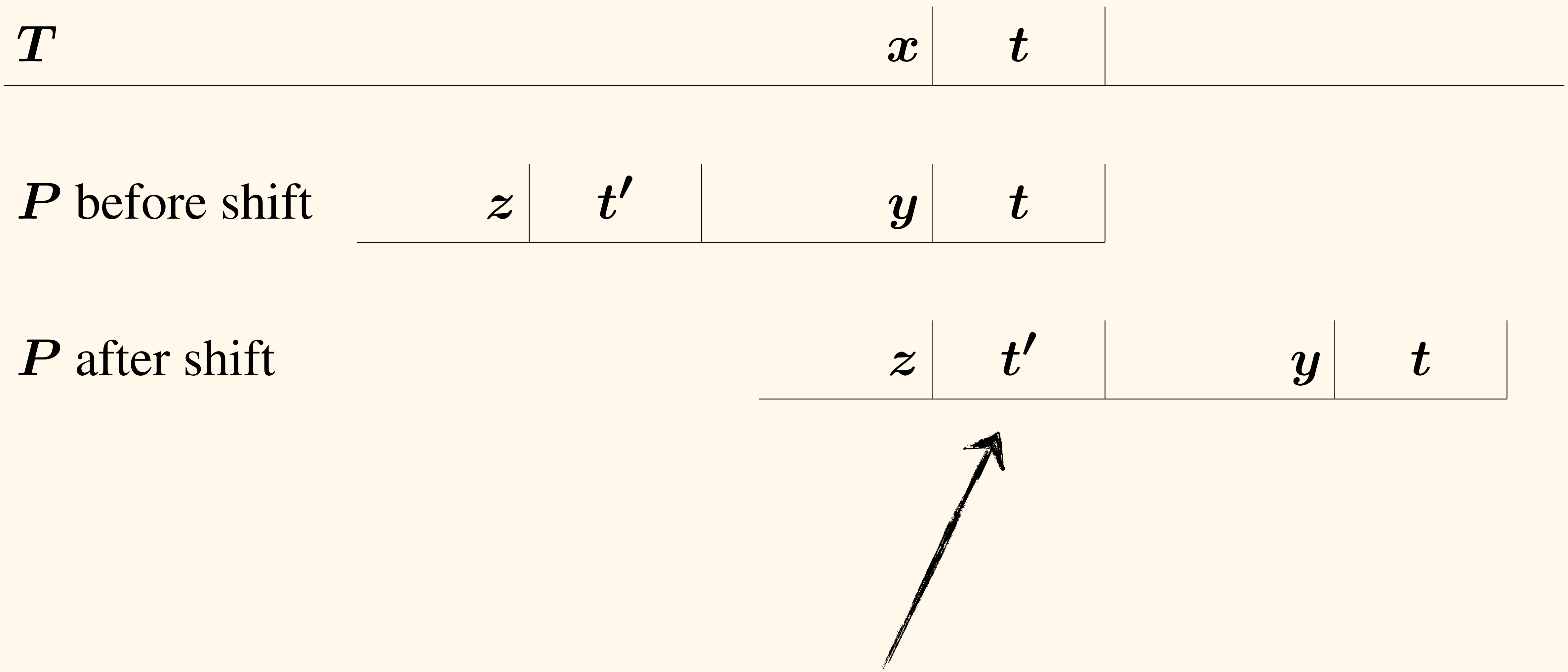
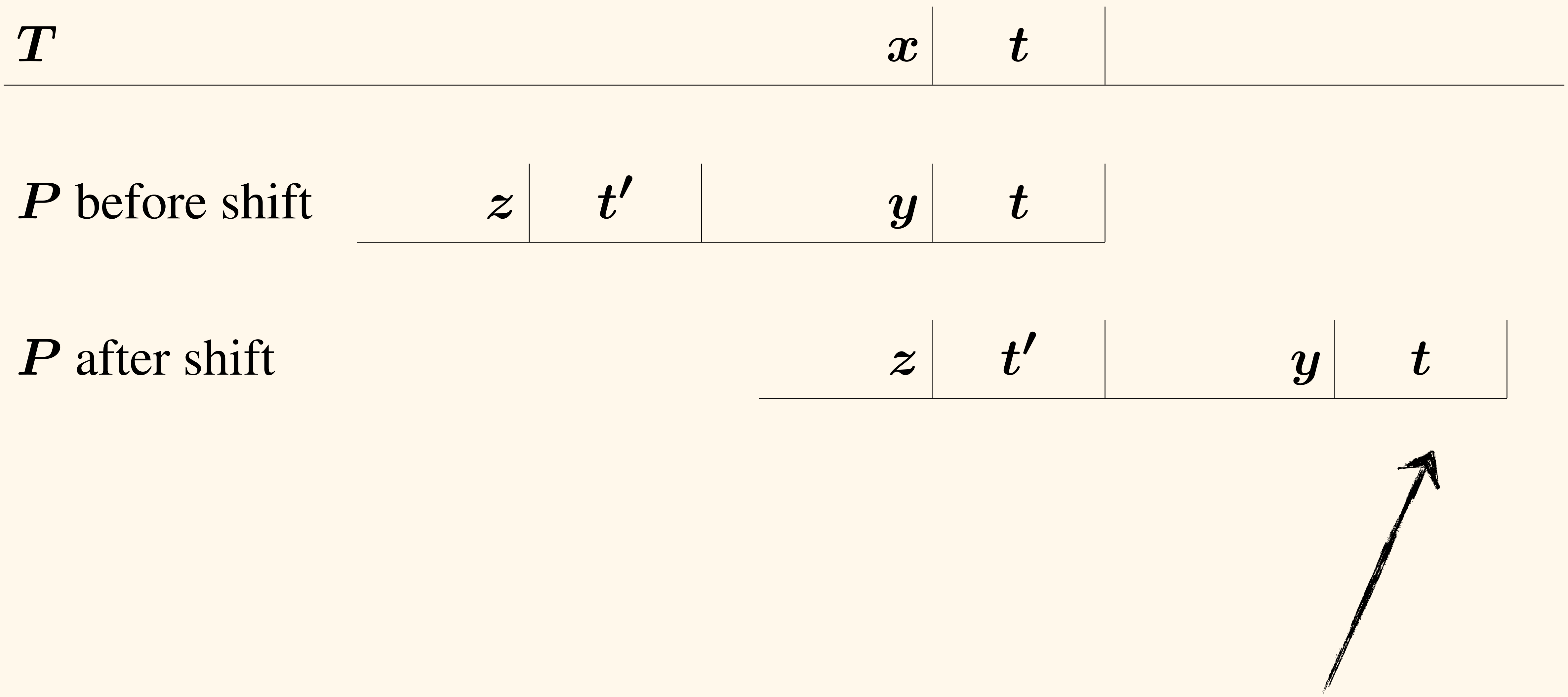


Illustration from Gusfield



Closest *possible* place for
match to occur.

Illustration from Gusfield



Point from which we start our next iteration of the algorithm.

More formally:

t: A substring of T and P matching at a particular alignment.

More formally:

t: A substring of T and P matching at a particular alignment.

t': The right-most copy of t in P s.t.:

More formally:

t: A substring of T and P matching at a particular alignment.

t': The right-most copy of t in P s.t.:

t' is *not* a suffix of P **and:**

More formally:

t: A substring of T and P matching at a particular alignment.

t': The right-most copy of t in P s.t.:

t' is *not* a suffix of P **and:**

the character to the left of t' in P differs from the character to the left of t in P.

More formally:

t: A substring of T and P matching at a particular alignment.

t': The right-most copy of t in P s.t.:

t' is *not* a suffix of P **and:**

the character to the left of t' in P differs from the character to the left of t in P.

If t' exists, shift P so that t' in P is below substring t in T.

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
	p	r	s	t	a	b	s	t	u	b	a	b	v	q	x	r	s	t
			q	c	a	b	d	a	b	d	a	b						
		i	1	2	3	4	5	6	7	8	9	0						
										↑								

t: ab

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
	p	r	s	t	a	b	s	t	u	b	a	b	v	q	x	r	s	t
			q	c	a	b	d	a	b	d	a	b						
		i	1	2	3	4	5	6	7	8	9	0						

t: ab

t′: occurs at position 3 in *P*

1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8
p	r	s	t	a	b	s	t	u	b	a	b	v	q	x	r	s	t
								q	c	a	b	d	a	b	d	a	b
							i	1	2	3	4	5	6	7	8	9	0
																	↑

t: ab

t′: occurs at position 3 in *P*

1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

p r s t a b s t u b a b v q x r s t

q	c	a	b	d	a	b	d	a	b

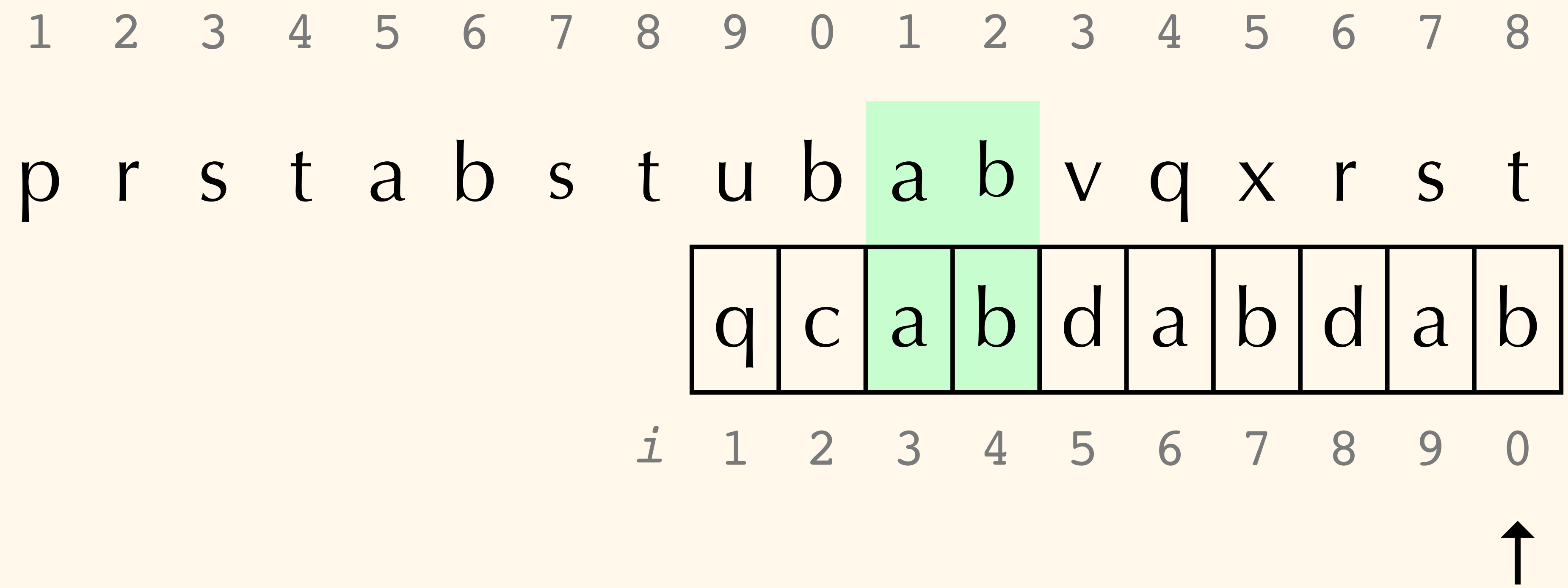
i 1 2 3 4 5 6 7 8 9 0

↑

t: ab

t′: occurs at position 3 in *P*

Note that had we relied on the “bad character” rule, we only would have been able to shift down 1 position!



t: ab

t': occurs at position 3 in P

Great! How do we find t'?

Definitions:

For each position i, let k = |P| - i + 1

(i.e., $k = |P[i,|P|]|$)
(i.e., k = the length of the suffix of P starting at i.)

Definitions:

For each position i, let $k = |P| - i + 1$

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

Definitions:

For each position i, let $k = |P| - i + 1$

Let L(i) = the largest position from which P[i,] matches a suffix of P[1,L(i)].

Let L′(i) = largest position from which P[i,] matches a suffix of P[1,L(i)] *and* s.t. the character preceding that suffix is not equal to P(i - 1).

	c	a	b	d	a	b	d	a	b
i	1	2	3	4	5	6	7	8	9
								↑	

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

	c	a	b	d	a	b	d	a	b
i	1	2	3	4	5	6	7	8	9

$L(8) = 6$

↑

Definitions:

For each position *i*, let *k* = |P| - *i* + 1

Let L(*i*) = the largest position from which P[*i*,] matches a suffix of P[1,L(*i*)].

Let L′(*i*) = largest position from which P[*i*,] matches a suffix of P[1,L(*i*)] *and* s.t. the character preceding that suffix is not equal to P(*i* - 1).

	c	a	b	d	a	b	d	a	b
i	1	2	3	4	5	6	7	8	9

L(8) = 6

↑

Definitions:

For each position *i*, let *k* = |P| - *i* + 1

Let L(*i*) = the largest position from which P[*i*,] matches a suffix of P[1,L(*i*)].

Let L′(*i*) = largest position from which P[*i*,] matches a suffix of P[1,L(*i*)] *and* s.t. the character preceding that suffix is not equal to P(*i* - 1).

	c	a	b	d	a	b	d	a	b
i	1	2	3	4	5	6	7	8	9

↑

L(8) = 6

L′(8) = 3

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

We'll use L(i) and L'(i) for detailed steps in the algorithm...

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

We'll use L(i) and L'(i) for detailed steps in the algorithm...

... but note that any position i in P where $L_i(P) > 0$ must have a corresponding repeating segment! (t' in previous slides)

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

We'll use $L(i)$ and $L'(i)$ for detailed steps in the algorithm...

Gusfield chap. 2 gives a lovely algorithm for computing $L(i)$ and $L'(i)$ in $O(|P|)$ time by using Z-boxes!

Definitions:

For each position i, let $k = |P| - i + 1$

Let $L(i)$ = the largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$.

Let $L'(i)$ = largest position from which $P[i,]$ matches a suffix of $P[1,L(i)]$ *and* s.t. the character preceding that suffix is not equal to $P(i - 1)$.

One more definition: $l(i)$.

$l(i)$ is the length of the largest suffix of $P(i,)$ that is also a prefix of P, and can also be calculated in linear time (see Gusfield).

Putting it all together:

1. Pre-calculate bad character table;

2. Pre-calculate good-suffix table;

3. Start at position |P| in T, move from right to left.

   a. Look for matching characters;

   b. If no match, skip max(bad character, good suffix) positions further down in the string.

4. Wash, Rinse, Repeat!

Putting it all together: roark_boyer_moore.pdf