KMP Algorithm

2015.04.07

Jeehyeong Kim

Exact Matching Problem

- **Exact matching problem**
  - Given a string P called the *pattern* and a longer string T called the *text*, the exact matching problem is to find all occurrences, if any, of pattern P in text T.
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	b	b	a	b	a	x	a	b	a	b	a	y			input
P	a	b	a												input

Exact Matching Problem

- **Exact matching problem**
  - Given a string P called the *pattern* and a longer string T called the *text*, the exact matching problem is to find all occurrences, if any, of pattern P in text T.
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	b	b	a	b	a	x	a	b	a	b	a	y			input
P	a	b	a												input
			a	b	a										

Exact Matching Problem

- **Exact matching problem**
  - Given a string *P* called the *pattern* and a longer string *T* called the *text*, the exact matching problem is to find all occurrences, if any, of pattern *P* in text *T*.
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	b	b	a	b	a	x	a	b	a	b	a	y			input
P	a	b	a												input
			a	b	a										
							a	b	a						

Exact Matching Problem

- **Exact matching problem**
  - Given a string P called the *pattern* and a longer string T called the *text*, the exact matching problem is to find all occurrences, if any, of pattern P in text T.
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	b	b	a	b	a	x	a	b	a	b	a	y			input
P	a	b	a												input
			a	b	a										
							a	b	a						
									a	b	a				

    - Note that two occurrences of P may overlap

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P	a	b	c												match at 1

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P	a	b	c												match at 2

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P	a	b	c												mismatch at 3

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P		a	b	c											Shift 1 place

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P		a	b	c											Mismatch at 2

Begin comparing again from the left end of *P*

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P			a	b	c										Shift 1 place

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P			a	b	c										Mismatch at 3

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P				a	b	c	d								Shift 1 place

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P				a	b	c									match at 4

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P				a	b	c									match at 5

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P				a	b	c									match at 6

P occurs at postion 4 of *T*

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P					a	b	c								Shift 1 place

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	d	a	b	c	b	d	a						
P					a	b	c								Mismatch at 5

Begin comparing again from the left end of *P*

The Knuth-Morris-Pratt Algorithm

- **Naïve algorithm**
  - Complexity
    - $O(mn)$
      - The length of $T = n$
      - The length of $P = m$
    - $O(m+n)$ → KMP algorithm

The KMP algorithm

- **The KMP shift idea**
  - Make **larger shifts** than the naïve algorithm
  - The number of **comparisons are smaller** than the naïve algorithm
    - After a shift, the left-most characters of *P* are guaranteed to match their counterparts in *T*

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve		a	b	c	x	a	b	c	d	e					Shift 1 place

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve			a	b	c	x	a	b	c	d	e				Shift 1 place

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve				a	b	c	x	a	b	c	d	e			Shift 1 place

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve					a	b	c	x	a	b	c	d	e		Shift 1 place

- Need to shift and compare 4 times
  - to find the start position of occurrences of P in T
- Have to compare the “already matched” part again

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve		a	b	c	x	a	b	c	d	e					
KMP					a	b	c	x	a	b	c	d	e		Shift 4 places

- KMP algorithm can shift *P* by four places without passing over any occurrences of *P* in *T*

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve		a	b	c	x	a	b	c	d	e					
KMP					a	b	c	x	a	b	c	d	e		Shift 4 places

  - KMP algorithm can shift *P* by four places without passing over any occurrences of *P* in *T*
  - Starts comparing at position 8
    - which was the "mismatched" position
    - Don't have to compare the "already matched" part again.

The KMP algorithm

- **The KMP shift idea**
  - For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T	a	b	c	x	a	b	c	a	b	c	x	a			
P	a	b	c	x	a	b	c	d	e						mismatch at 8
Naïve		a	b	c	x	a	b	c	d	e					
KMP					a	b	c	x	a	b	c	d	e		Shift 4 places

  - KMP algorithm can shift *P* by four places without passing over any occurrences of *P* in *T*
  - Starts comparing at position 8
    - which was the "mismatched" position
    - Don't have to compare the "already matched" part again.

The KMP algorithm

mismatch

T	a	b	a	x	a	b	a	y	e	f	g	h
P	a	b	a	x	a	b	a	d	e			

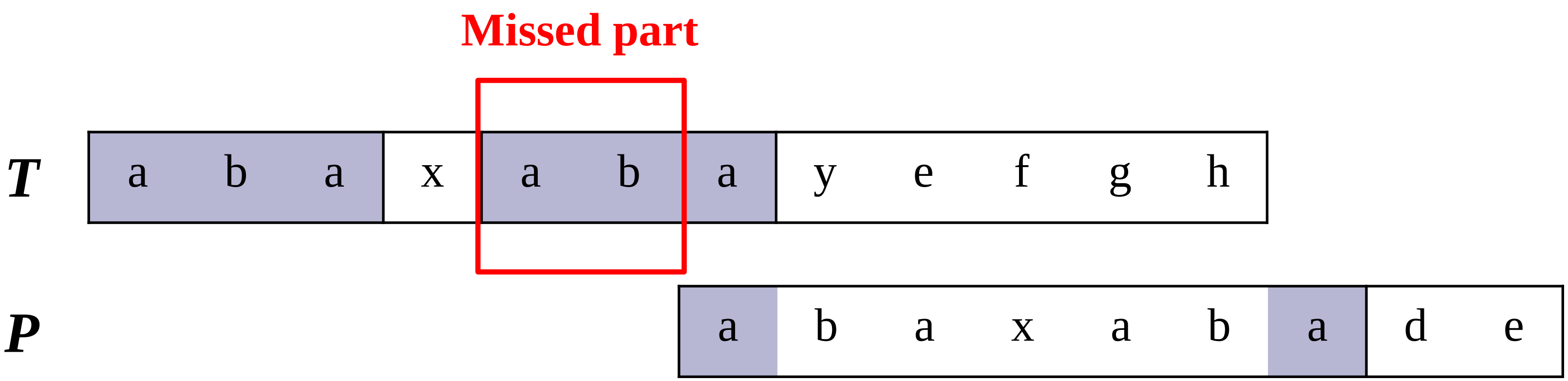
match

The KMP algorithm

mismatch

T	a	b	a	x	a	b	a	y	e	f	g	h
P	a	b	a	x	a	b	a	d	e			

The KMP algorithm



∴ Suffix of *P* should be the largest one

The KMP algorithm

- **The Definition of $sp_i(P)$**

For each position of i in pattern P, define $sp_i(P)$ to be the length of the longest proper suffix of $P[1…i]$ that matches a prefix of P.

- $sp_i(P)$ is the length of the longest proper substring of $P[1…i]$ that ends at i and matches a prefix of P.

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.

  - For example $P = aaabcaabe$, sp_6 = ??

i

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

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.

  - For example $P = aaabcaabe$, sp_6 = ??

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
				$\neq$								
		a	a	b	c	a						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, $sp_6 = ??$

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, sp_6 = ??

						i					
	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b $\neq$	c	a						Mismatch
			a	a	b	c						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.

  - For example $P = aaabcaabe$, sp_6 = ??

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, sp_6 = ??

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	$\neq$ c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, sp_6 = ??

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, $sp_6 = ??$

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a						Match

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.
  - For example $P = aaabcaabe$, sp_6 = ??

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a						Match

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.

  - For example $P = aaabcaabe$, $sp_6 = ??$

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a						Match
						a						Match

The KMP algorithm

- **The Definition of $sp_i(P)$**
  - $sp_i(P)$ is the length of the longest proper substring of $P[1\ldots i]$ that ends at i and matches a prefix of P.

  - For example $P = aaabcaabe$
  - So, $sp_6 = 2$

i

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a						Match

The KMP algorithm

- Weakness of sp_i
  - Shift using sp_i value
  - ex) P = a b c a b d a b c a b d $\qquad sp_{11} = 5$

sp_i

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
T	a	b	c	a	b	d	a	b	c	a	b	**c**	a	b	d	…
P	a	b	c	a	b	d	a	b	c	a	b	**d**				

T	a	b	c	a	b	d	a	b	c	a	b	**c**	a	b	d	…
P	→						a	b	c	a	b	**d**				

There's an mismatch at position 12. We can shift 6 character. But, the same kind of mismatch(comparing c with d) occurs at the same position 12.

The KMP algorithm

- **The Definition of $sp'_i(P)$**

For each position of i in pattern P, define $sp'_i(P)$ to be the length of the longest proper suffix of $P[1\ldots i]$ that matches a prefix of P,

with the character $P(i+1) \neq P(sp'_i+1)$

- means right character of matched prefix and suffix are not equal.

The KMP algorithm

- **The Definition of $sp'_i(P)$**
  - with the character $P(i+1) \neq P(sp'_i+1)$
  - For example $P = aabcaabe$, $sp_6 = 2$, $sp'_6 = ??$

	sp'_i		sp'_i+1			i	$i+1$					
	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a	b					Match
						a	a					Match

The KMP algorithm

- **The Definition of $sp'_i(P)$**
  - with the character $P(i+1) \neq P(sp'_i+1)$
  - For example $P = aabcaabe$, $sp_6 = 2$, $sp'_6 = ??$

sp'_i sp'_i+1 i $i+1$

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a	b					Match
						a	a					Match

The KMP algorithm

- **The Definition of $sp'_i(P)$**
  - with the character $P(i+1) \neq P(sp'_i+1)$
  - For example $P = aabcaabe$, $sp_6 = 2$, $sp'_6 = ??$

sp'_i sp'_i+1 i $i+1$

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a	b					Match
						a	a					Match

The KMP algorithm

- **The Definition of $sp'_i(P)$**
  - with the character $P(i+1) \neq P(sp'_i+1)$
  - For example $P = aabcaabe$, $sp_6 = 2$, $sp'_6 = ??$

sp'_i sp'_i+1 i $i+1$

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a	b					Match
						a	a					Match

The KMP algorithm

- **The Definition of $sp'_i(P)$**
  - with the character $P(i+1) \neq P(sp'_i+1)$
  - For example P = *aabcaabe*, $sp_6 = 2$, $sp'_6 = 1$

sp'_i sp'_i+1 i $i+1$

	1	2	3	4	5	6	7	8	9	10	11	
P	a	a	b	c	a	a	b	e				
		a	a	b	c	a						Mismatch
			a	a	b	c						Mismatch
				a	a	b						Mismatch
					a	a	b					Match
						a	a					Match

The KMP algorithm

- Weakness of sp_i
  - Shift using sp_i value
  - ex) P = a b c a b d a b c a b d

$sp_{11} = 5$

$sp'_{11} = ??$

sp_i

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
T	a	b	c	a	b	d	a	b	c	a	b	**c**	a	b	d	…
P	a	b	c	a	b	d	a	b	c	a	b	**d**				

T	a	b	c	a	b	d	a	b	c	a	b	**c**	a	b	d	…
P							a	b	c	a	b	**d**				

The KMP algorithm

- Weakness of sp_i
  - Shift using sp'_i value
  - ex) P = a b c a b d a b c a b d

$sp_{11} = 5$
$sp'_{11} = 2$

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
sp'_i	T	a	b	c	a	b	d	a	b	c	a	b	c	a	b	d	...
	P	a	b	c	a	b	d	a	b	c	a	b	d				

T	a	b	c	a	b	d	a	b	c	a	b	c	a	b	d	...
P										a	b	c	a	b	d	...

$sp'_{11} = 2$, We can **shift more characters** than when we use sp_{11} value. And the next character is not the same character that was mismatched. We don't have to do the same comparison again.

The KMP algorithm

- **Theorem 2.3.3 (Time Complexity)**

  In the KMP method, the number of character comparisons is at most $2n$.

Time complexity of KMP

- the total number of character comparisons =

  number of matches + number of mismatches
- For example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	x	y	a	b	c	x	a	b	c	x	a	d	c	x	a	d	c	d	q	f	e	g
P			a	b	c	x	a	b	c	d	e											
compare			1	1	1	1	1	1	1	1												
Shift 4							a	b	c	x	a	b	c	d	e							
compare										1	1	1	1	1								
Shift 4											a	b	c	x	a	b	c	d	e			
compare														1	1							
Total compare			1	1	1	1	1	1	1	2	1	1	1	2	1							

Shift Rule

- **Proof**
  - n : the length of T
  - s : number of shifts

the total number of character comparisons =

number of matches + number of mismatches

1. The number of matches
   ➔ if character matches, never compare again (the number of matches = n)

2. The number of mismatches
   ➔ if character mismatches, shift occurs (the number of mismatches = s)

Shift Rule

- **Proof**
  - n : the length of T
  - s : number of shifts

the total number of character comparisons

$$= n + s$$

The number of shifts cannot be larger than the length of T

$$\leq n + n$$

$$\leq 2n$$

The Original Preprocessing For KMP

- **The Preprocessing for KMP**

  - Compute $sp_i(P)$ for each position i
    - From $i = 2$ to $i = n$, ($sp_1 = 0$)

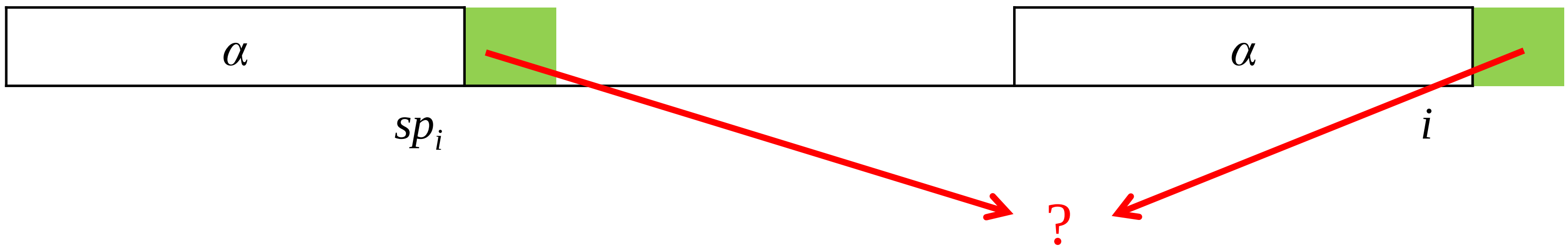
  - Inductively

    $$sp_1 \rightarrow sp_2 \rightarrow sp_3 \rightarrow sp_4 \ldots$$

    - To compute sp_i , assume that we know sp_1 , $sp_2 \ldots sp_{i-1}$

The Original Preprocessing For KMP

- **How to compute sp_{i+1}**



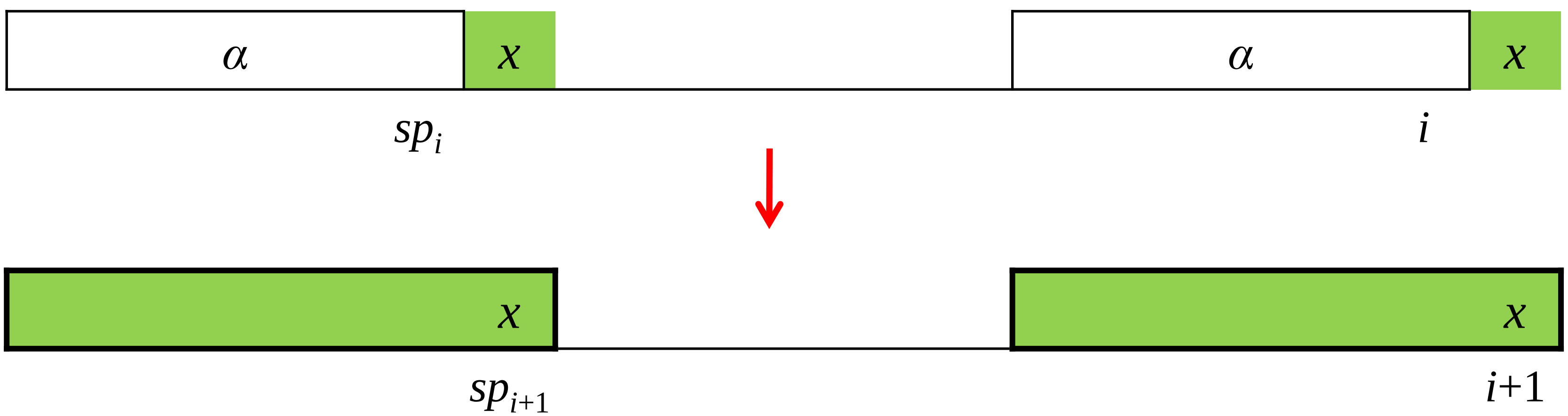
There are two cases :

① $P(i+1) = P(sp_i)+1$

② $P(i+1) \neq P(sp_i)+1$

The Original Preprocessing For KMP

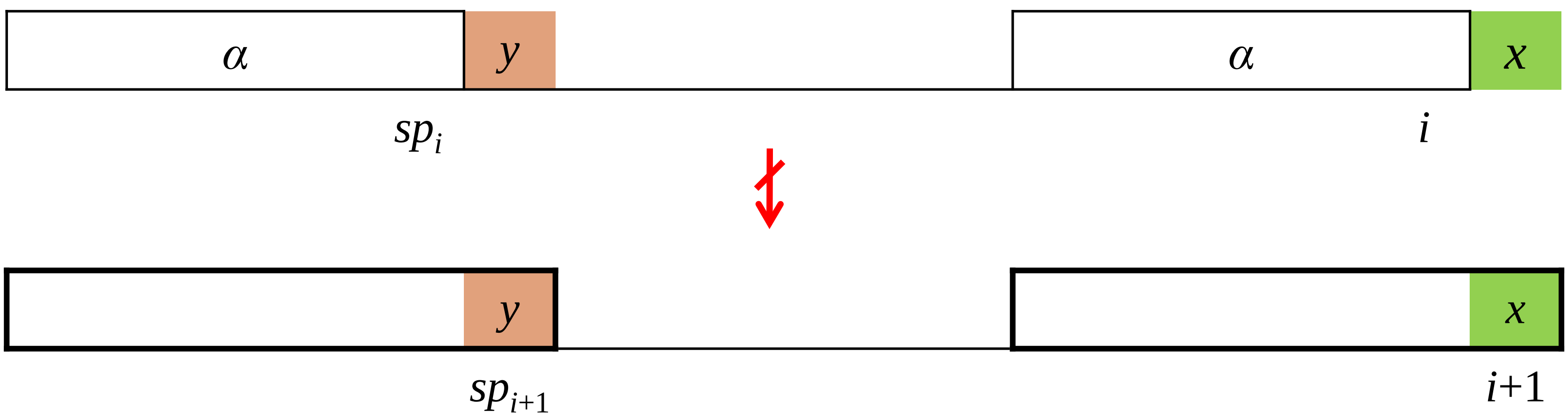
① $P(sp_i)+1 = P(i+1)$



$sp_{i+1} = sp_i + 1$

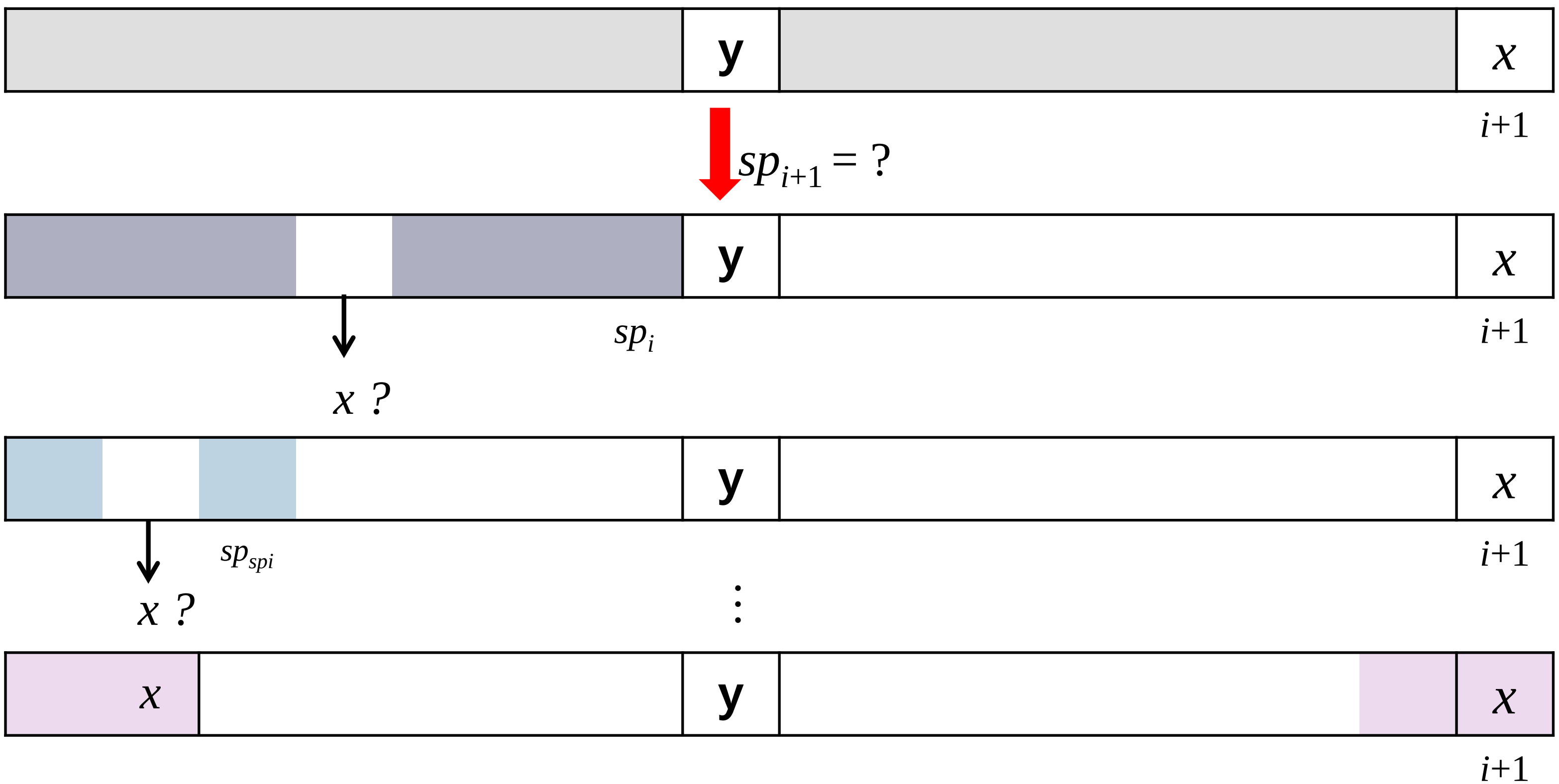
The Original Preprocessing For KMP

② $P(sp_i)+1 \neq P(i+1)$ (the general case of computing sp_{i+1})



The Original Preprocessing For KMP

② $P(sp_i)+1 \neq P(i+1)$



The Original Preprocessing For KMP

a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
										↑												↑	↑
										sp_i												i	$i+1$

The Original Preprocessing For KMP

$\neq$

a b x a b q a b x a b **r** a b x a b q a b x a b **x**

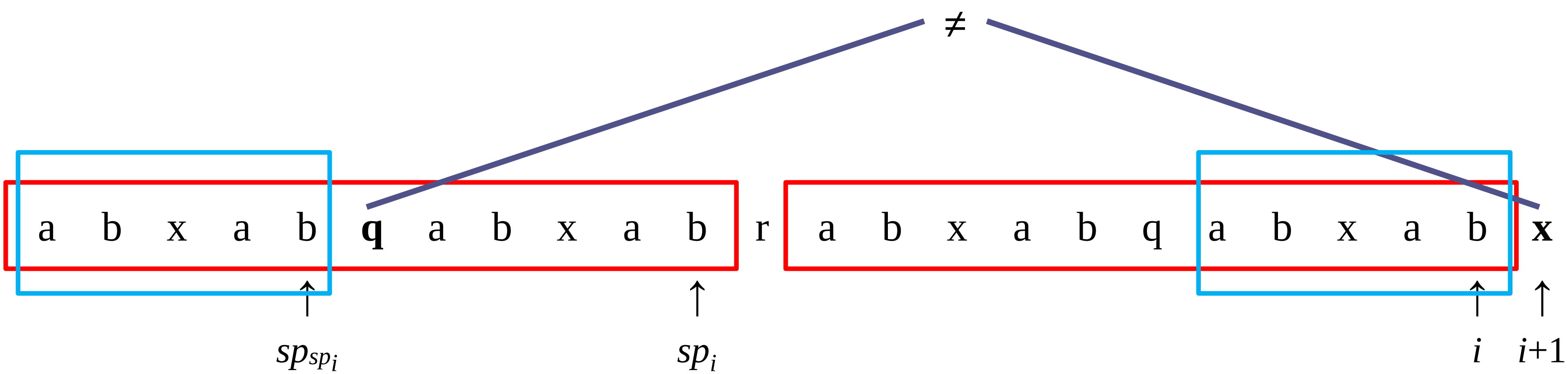
↑ sp_i ↑ i ↑ $i+1$

The Original Preprocessing For KMP

a b x a b q a b x a b **r** a b x a b q a b x a b **x**

sp_{sp_i} sp_i i $i+1$

The Original Preprocessing For KMP

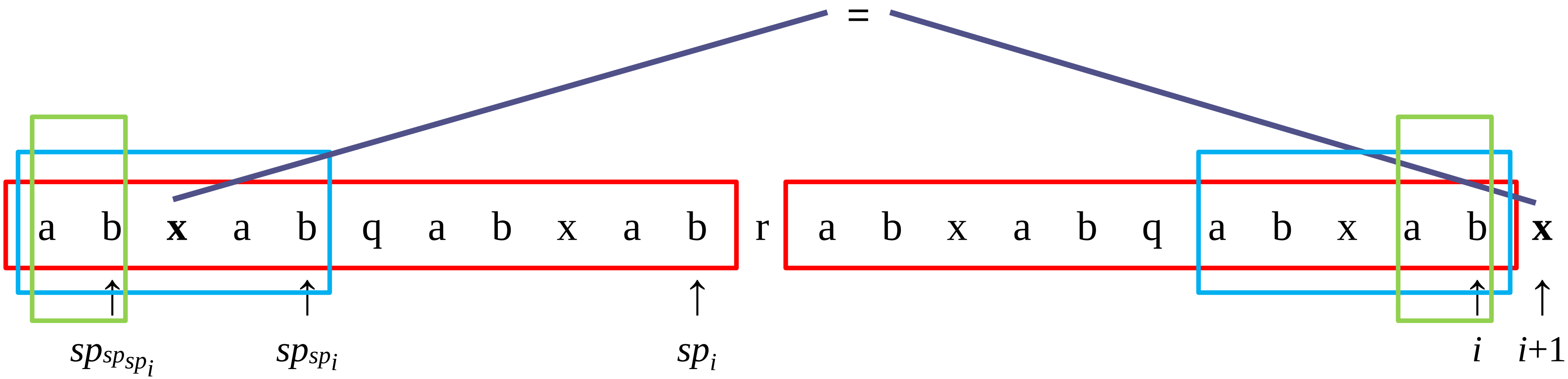


The Original Preprocessing For KMP

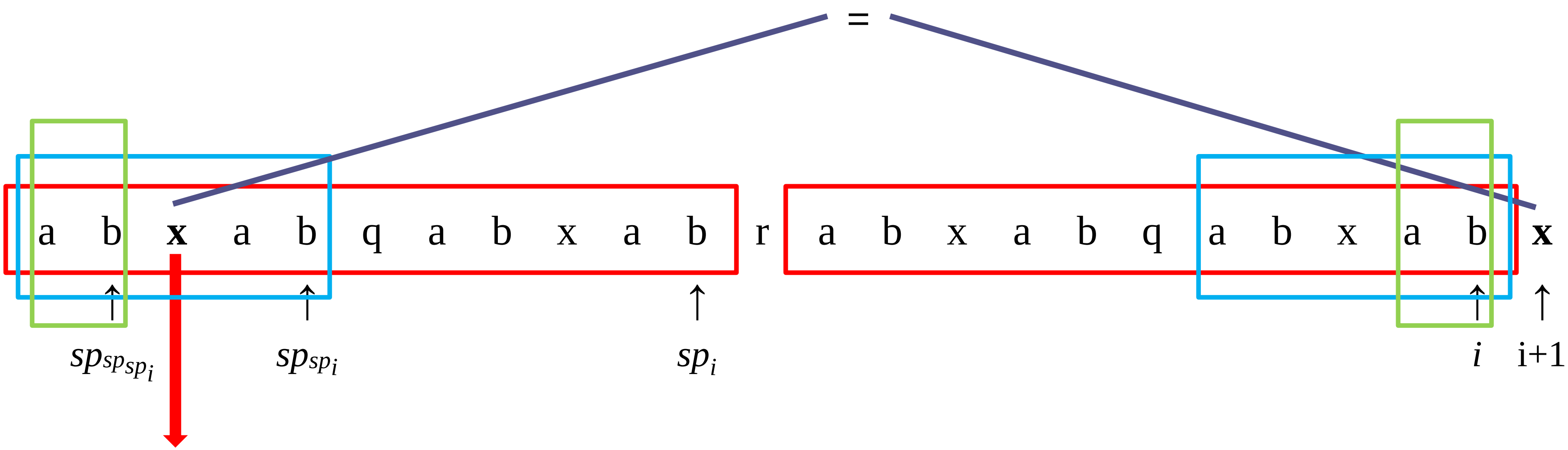
a b **x** a B q a b x a b r a b x a b q a b x a b **x**

$sp_{sp_{sp_i}}$ sp_{sp_i} sp_i i $i+1$

The Original Preprocessing For KMP



The Original Preprocessing For KMP



$$sp_{i+1} = sp_{sp_{sp_i}} + 1$$

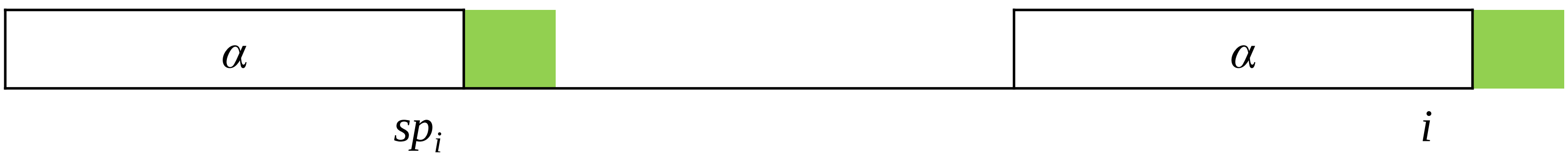
The Original Preprocessing For KMP

- **Theorem 3.3.1 (Complexity)**

  Algorithm SP finds all the $sp_i(P)$ values in $O(m)$ time, where m is the length of P

The Original Preprocessing For KMP

- time complexity



- If

① $P(sp_i)+1 = P(i+1)$ ➔ $\boldsymbol{O(1)}$

② $P(sp_i)+1 \neq P(i+1)$ ➔ depends on

the number of times $\boldsymbol{sp_i}$ jumps

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0																					

The Original Preprocessing For KMP

- time complexity

	$\boxed{a}$	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1																				

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2																			

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0																		

The Original Preprocessing For KMP

- time complexity

	$\boxed{a}$	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1																	

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2																

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2	3															

The Original Preprocessing For KMP

- time complexity

	$\boxed{a}$	$\boxed{b}$	$\boxed{x}$	$\boxed{a}$	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2	3	4														

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2	3	4	5													

When the value of sp_i is increasing, it can only increase by 1

➔ The value of sp_i can increase at most m-1.

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2	3	4	5	0	…											

The value of sp_i can decrease as large as the amount it has increased.

The Original Preprocessing For KMP

- time complexity

	a	b	x	a	b	q	a	b	x	a	b	r	a	b	x	a	b	q	a	b	x	a	b	x
sp_i	0	0	0	1	2	0	1	2	3	4	5	0	1	…										

The total value of sp_i can increase at most m-1 over the entire algorithm → No matter how many times does sp_i decrease, the total amount that the value of sp_i can decrease is bounded by m-1 → $O(m)$

The Original Preprocessing For KMP

- time complexity

If

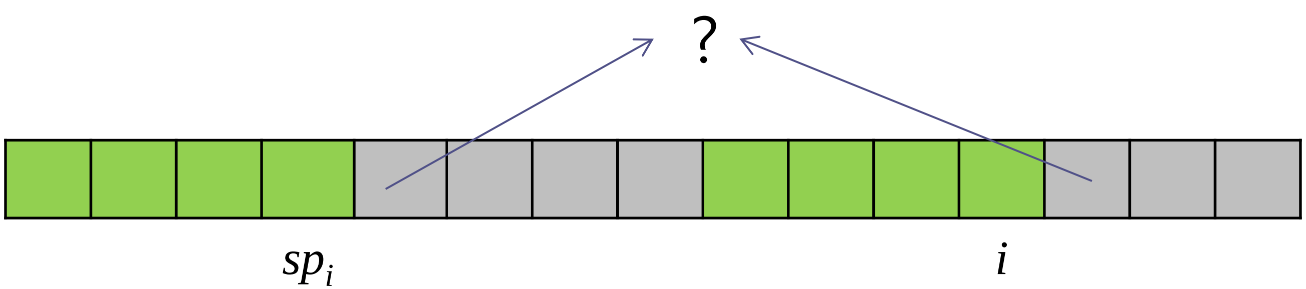
① $P(sp_i)+1 = P(i+1)$ ➔ $\boldsymbol{O}(\mathbf{1}) \; \mathbf{*} \; \mathbf{m}$

② $P(sp_i)+1 \neq P(i+1)$ ➔ $\boldsymbol{O}(\mathbf{m})$

$O(m)$

The Original Preprocessing For KMP

- **How to compute sp'_i**
  - assume that we know the value sp_i



① $P(sp_i)+1 \neq P(i+1)$ ➔ $sp'_i = sp_i$

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$

The Original Preprocessing For KMP

① $P(sp_i)+1 \neq P(i+1)$ ➔ $sp'_i = sp_i$

$\neq$

$\boxed{a\ b\ x\ a\ b\ q\ a\ b\ x\ a\ b}$ **y** $\boxed{a\ b\ x\ a\ b\ q\ a\ b\ x\ a\ b}$ **x**

↑ sp_i ↑ i

The Original Preprocessing For KMP

① $P(sp_i)+1 \neq P(i+1)$ ➔ $sp'_i = sp_i$

$\neq$

a b x a b q a b x a b **y** a b x a b q a b x a b **x**

↑ sp_i **= sp'_i** ↑ i

according to the definition of sp'_i

$sp'_i = sp_i$

The Original Preprocessing For KMP

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$

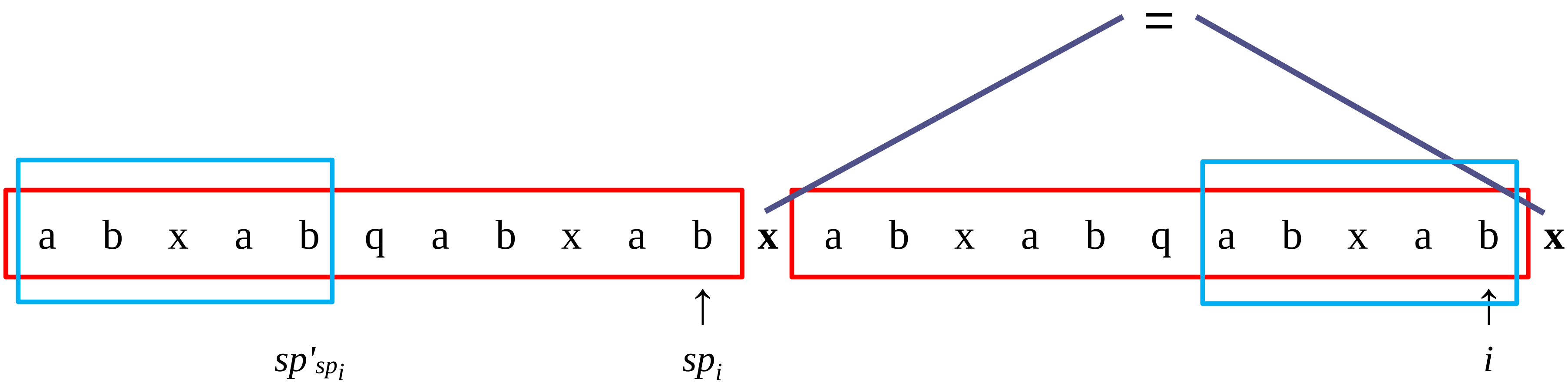
=

a b x a b q a b x a b **x** a b x a b q a b x a b **x**

↑ sp_i ↑ i

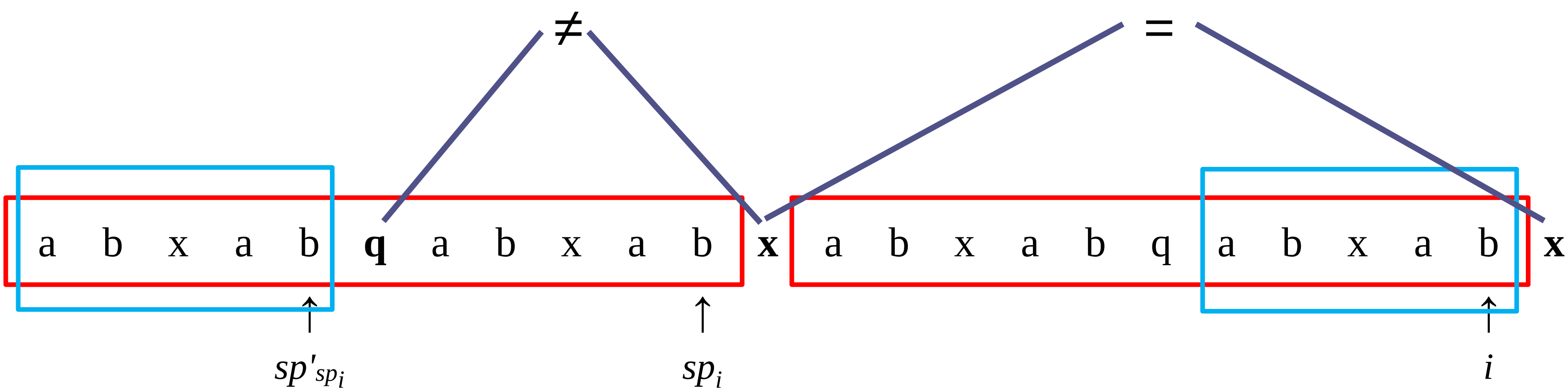
The Original Preprocessing For KMP

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$



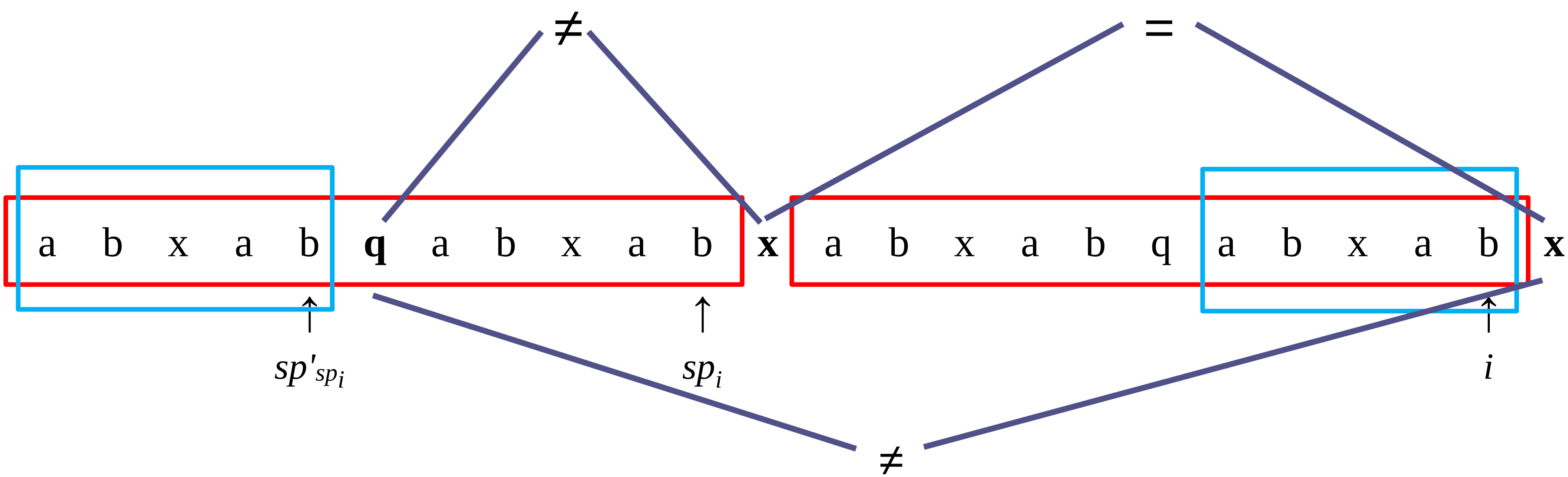
The Original Preprocessing For KMP

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$



The Original Preprocessing For KMP

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$



Satisfies the definition of sp'_i

$sp'_i = sp'_{sp_i}$

The Original Preprocessing For KMP

- time complexity

The time used to compute $\boldsymbol{sp_i}$ $\quad O(\mathrm{m})$

① $P(sp_i)+1 \neq P(i+1)$ ➔ $sp'_i = sp_i$ $\quad O(1) * m$

② $P(sp_i)+1 = P(i+1)$ ➔ $sp'_i = sp'_{sp_i}$ $\quad O(1) * m$

$$\left.\begin{array}{l} O(\mathrm{m}) \\ O(1) * m \\ O(1) * m \end{array}\right\} O(m)$$