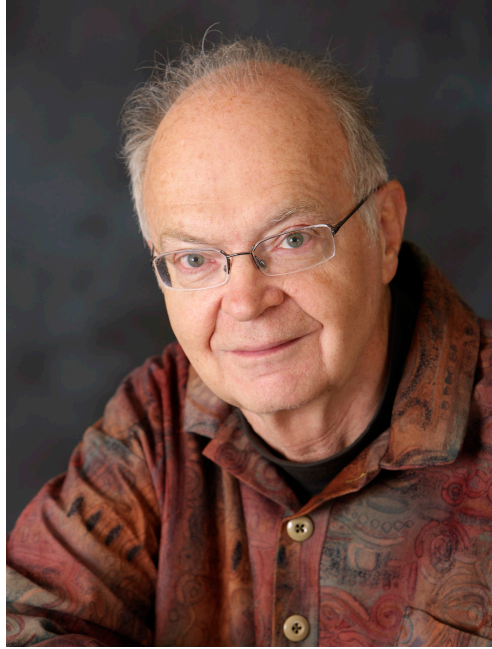


# **String Searching and KMP Algorithm**

**Shusen Wang**

# Knuth-Morris-Pratt Algorithm



Donald Knuth

1938 -



James H. Morris

1941 -



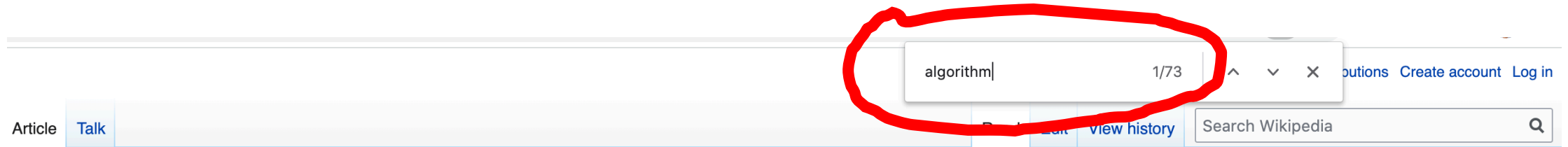
Vaughan Pratt

1944 -

## Reference:

- Knuth, Morris, & Pratt. [Fast pattern matching in strings](#). *SIAM Journal on Computing*, 1977.

# String Searching (aka String Matching)



## Knuth–Morris–Pratt algorithm

From Wikipedia, the free encyclopedia

In [computer science](#), the **Knuth–Morris–Pratt string-searching algorithm** (or **KMP algorithm**) searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

The **algorithm** was conceived by [James H. Morris](#) and independently discovered by [Donald Knuth](#) "a few weeks later" from automata theory.<sup>[1][2]</sup> Morris and [Vaughan Pratt](#) published a technical report in 1970.<sup>[3]</sup> The three also published the **algorithm** jointly in 1977.<sup>[1]</sup> Independently, in 1969, [Matiyasevich](#)<sup>[4][5]</sup> discovered a similar **algorithm**, coded by a two-dimensional Turing machine, while studying a string-pattern-matching recognition problem over a binary alphabet. This was the first linear-time **algorithm** for string matching.<sup>[6]</sup>

### Contents [\[hide\]](#)

- 1 [Background](#)
- 2 [KMP algorithm](#)
  - 2.1 [Example of the search algorithm](#)
  - 2.2 [Description of pseudocode for the search algorithm](#)
  - 2.3 [Efficiency of the search algorithm](#)
- 3 ["Partial match" table \(also known as "failure function"\)](#)
  - 3.1 [Working example of the table-building algorithm](#)
  - 3.2 [Description of pseudocode for the table-building algorithm](#)
  - 3.3 [Efficiency of the table-building algorithm](#)
- 4 [Efficiency of the KMP algorithm](#)
- 5 [Variants](#)
- 6 [References](#)
- 7 [External links](#)

# Examples

## Example Inputs



**txt =**

r	e	p	r	e	s	e	n	t
---	---	---	---	---	---	---	---	---



**pat =**

r	e
---	---

# Examples

## Example Inputs

`txt =`

<b>r</b> 0	<b>e</b>	<b>p</b>	<b>r</b> 3	<b>e</b>	<b>s</b>	<b>e</b>	<b>n</b>	<b>t</b>
---------------	----------	----------	---------------	----------	----------	----------	----------	----------

`pat =`

<b>r</b>	<b>e</b>
----------	----------

## Example Outputs

`pos = [0, 3]`

# Examples

## Example Inputs

`txt =`

r	e	p	r	e	s	e	n	t
---	---	---	---	---	---	---	---	---

`pat =`

a	b	c
---	---	---

## Example Outputs

`pos = []` (not found)

# Brute-Force Search

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

**txt** = 

a	a	a	a	a	a	a	a	b
---	---	---	---	---	---	---	---	---

**pat** = 

a	a	a	b
---	---	---	---



# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt = 

a	a	a	a	a	a	a	a	b
---	---	---	---	---	---	---	---	---

pat = 

a	a	a	b
---	---	---	---

Not matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt =

a	a <sub>1</sub>	a	a	a	a	a	a	b
---	----------------	---	---	---	---	---	---	---

pat =

a	a	a	b
---	---	---	---

Not matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt =

a	a	a <sub>2</sub>	a	a	a	a	a	b
---	---	----------------	---	---	---	---	---	---

pat =

a	a	a	b
---	---	---	---

Not matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt =

a	a	a	a	a	a	a	a	b
---	---	---	---	---	---	---	---	---

pat =

a	a	a	b
---	---	---	---

Not matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt =

a	a	a	a	a	a	a	a	b
---	---	---	---	---	---	---	---	---

pat =

a	a	a	b
---	---	---	---

Not matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.

txt =

a	a	a	a	a	a	a	a	b
---	---	---	---	---	---	---	---	---

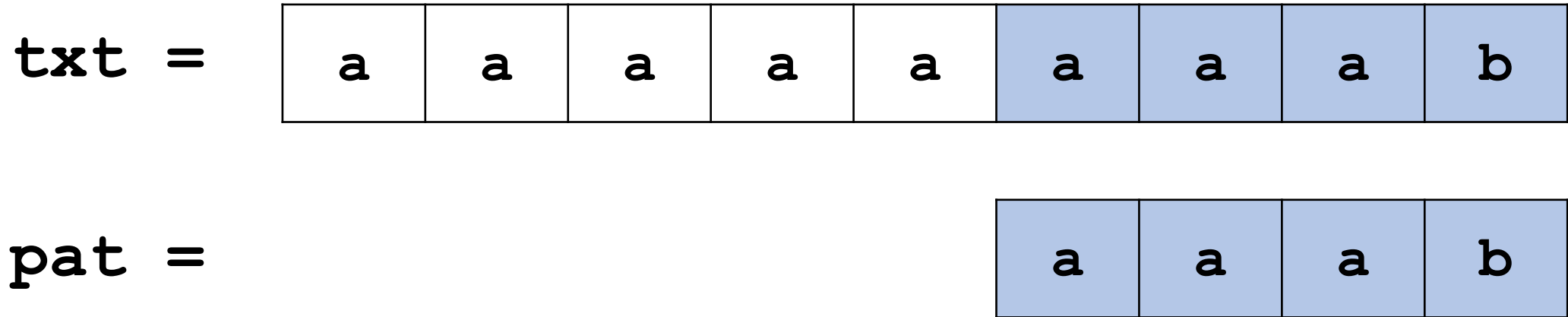
pat =

a	a	a	b
---	---	---	---

Matched!

# Brute-Force Search

**Algorithm:** Compare pat with all the substrings of txt.



**Time complexity:**  $O(|\text{txt}| \cdot |\text{pat}|)$ .

# **KMP Algorithm: Basic Idea**



# Basic Idea

**Question:** Anything special about the pattern string?

`txt =`

k	a	f	k	a	f	k	a	p	q	k
---	---	---	---	---	---	---	---	---	---	---

`pat =`

k	a	f	k	a	x	y
---	---	---	---	---	---	---

# Basic Idea

**Question:** Anything special about the pattern string?

`txt` = 

k	a	f	k	a	f	k	a	p	q	k
---	---	---	---	---	---	---	---	---	---	---

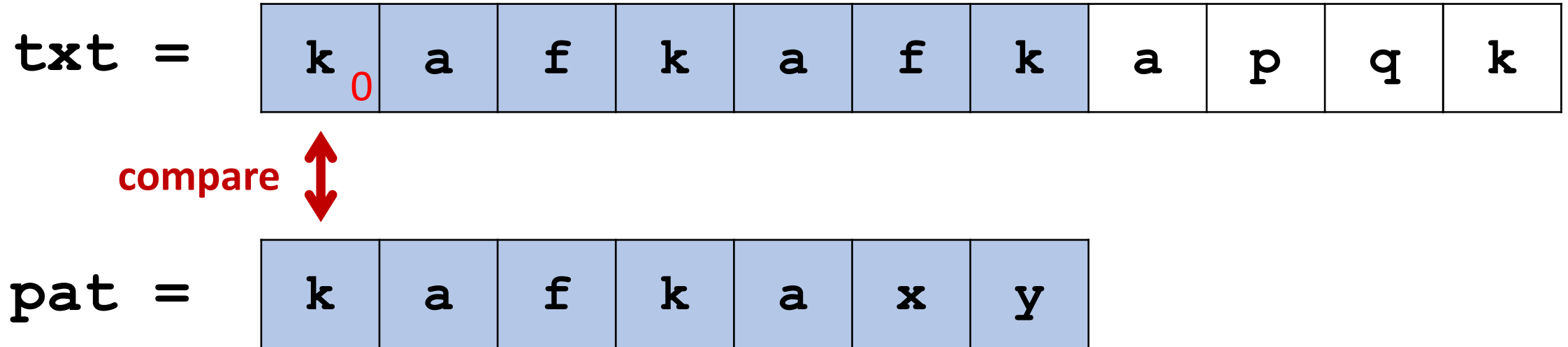
`pat` = 

k	a	f	k	a	x	y
---	---	---	---	---	---	---

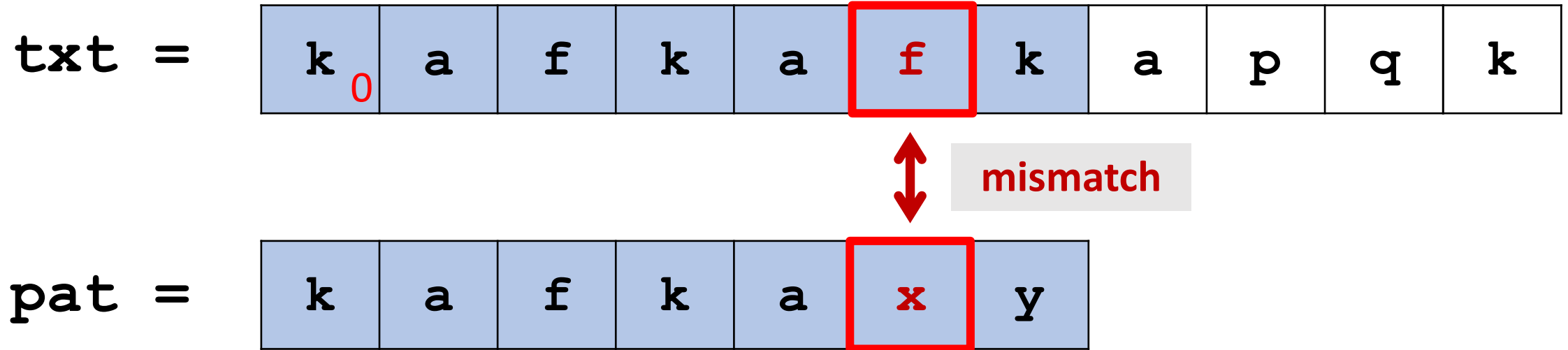
a prefix

a suffix of `pat[0:5]`

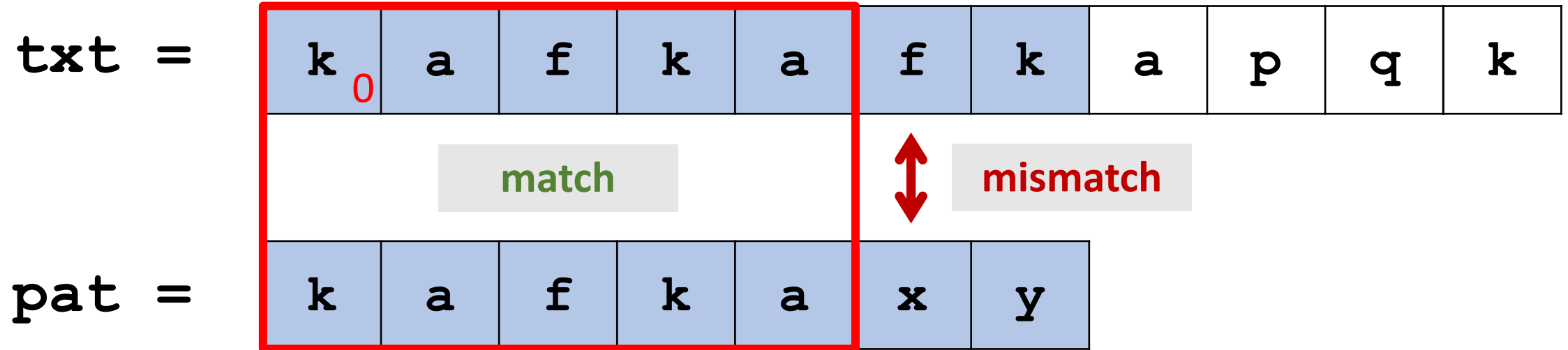
# Basic Idea



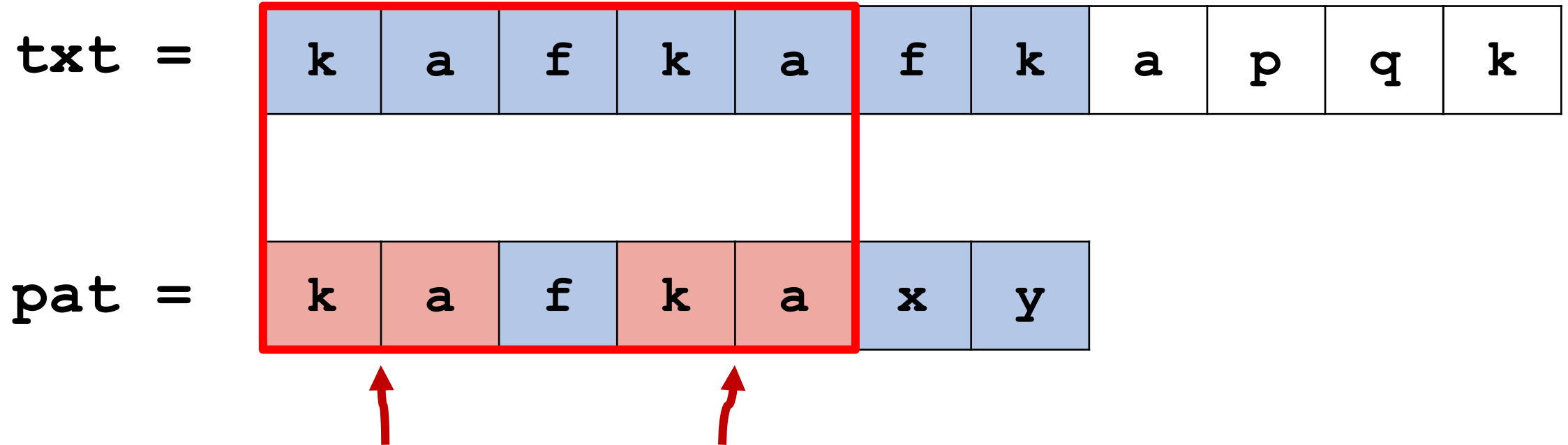
# Basic Idea



# Basic Idea

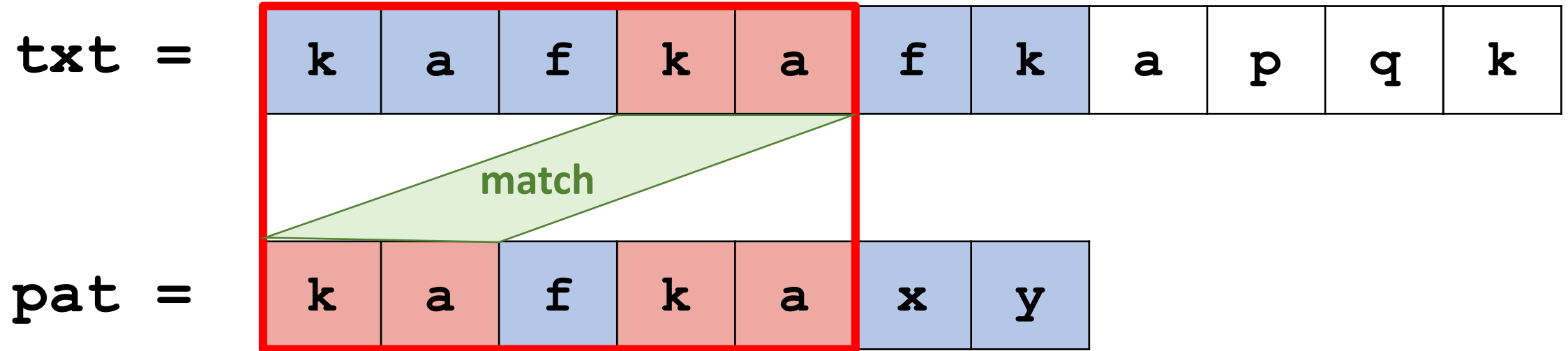


# Basic Idea

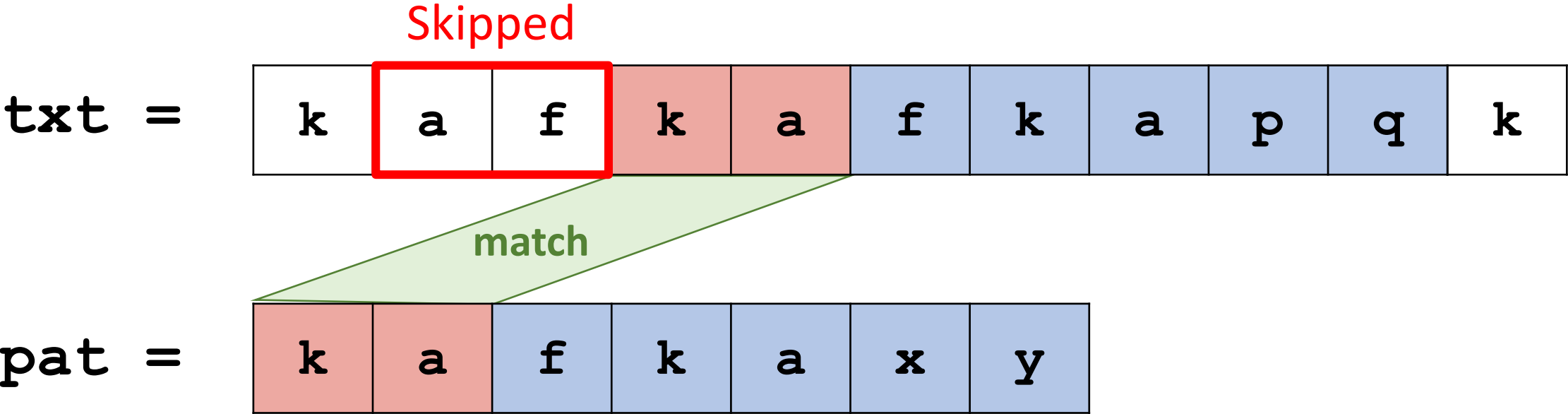


We know the **prefix** and the **suffix** are the same.

# Basic Idea

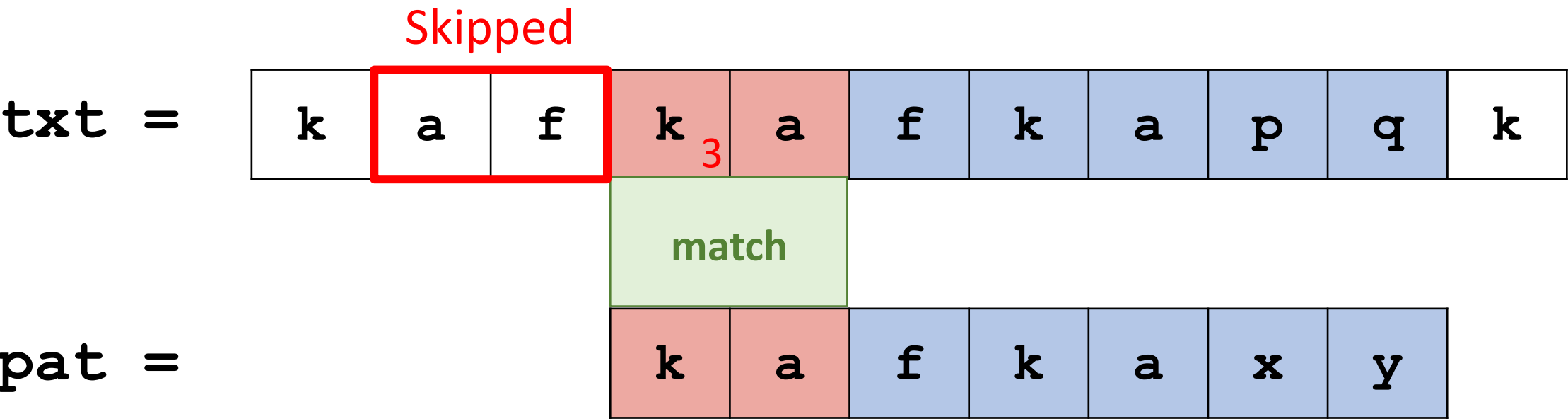


# Basic Idea

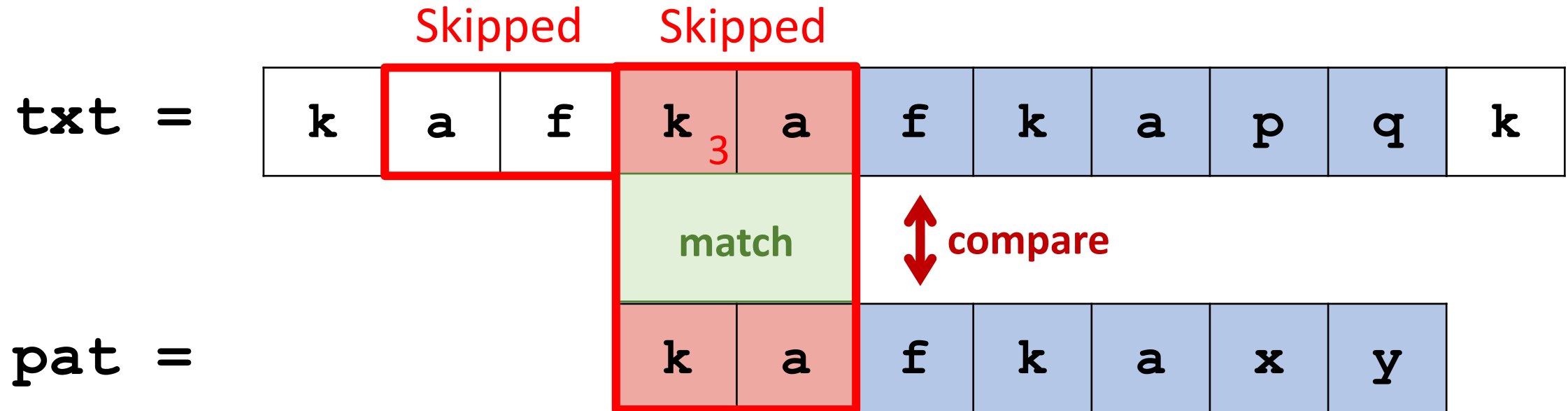




# Basic Idea



# Basic Idea



In this way, useless comparisons are skipped.

# Longest Prefix Suffix Array

# Longest Prefix Suffix Array

pat =	a	b	a	b	a	b	c	a
L =	0	0	1	2	3	4	0	1

# Longest Prefix Suffix Array

<b>pat</b> =	a	b	a	b	a	b	c	a
<b>L</b> =	0	0	1	2	3	4	0	1

What does the number mean?

# Longest Prefix Suffix Array

The first 3 chars

pat	=	a	b	a	b	a	b	c	a
L	=	0	0	1	2	3	4	0	1

What does the number mean?

# Longest Prefix Suffix Array

The first 3 chars      The last 3 chars

pat =	a	b	a	b	a	b	c	a
L =	0	0	1	2	3	4	0	1

What does the number mean?


# Longest Prefix Suffix Array

pat =

L =

**next** =

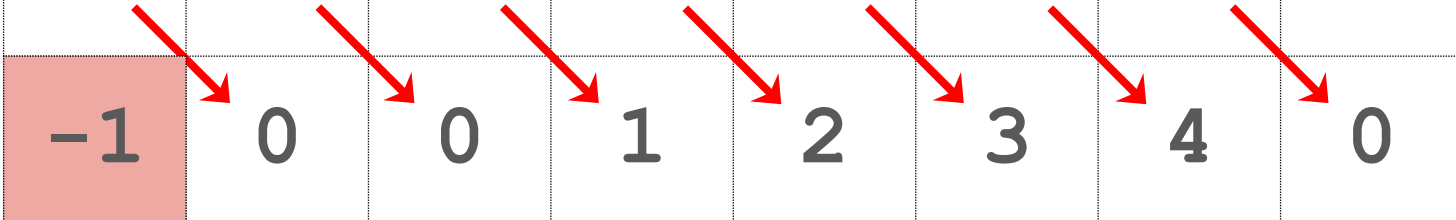
a	b	a	b	a	b	c	a
0	0	1	2	3	4	0	1
	0	0	1	2	3	4	0





# Longest Prefix Suffix Array

<b>pat</b> =	a	b	a	b	a	b	c	a
<b>L</b> =	0	0	1	2	3	4	0	1
<b>next</b> =	-1	0	0	1	2	3	4	0



# Property of the Array

<b>pat</b> =	a	b	a	b	a	b	c	a
	0	1	2	3	4	5	6	7
<b>next</b> =	-1	0	0	1	2	3	4	0

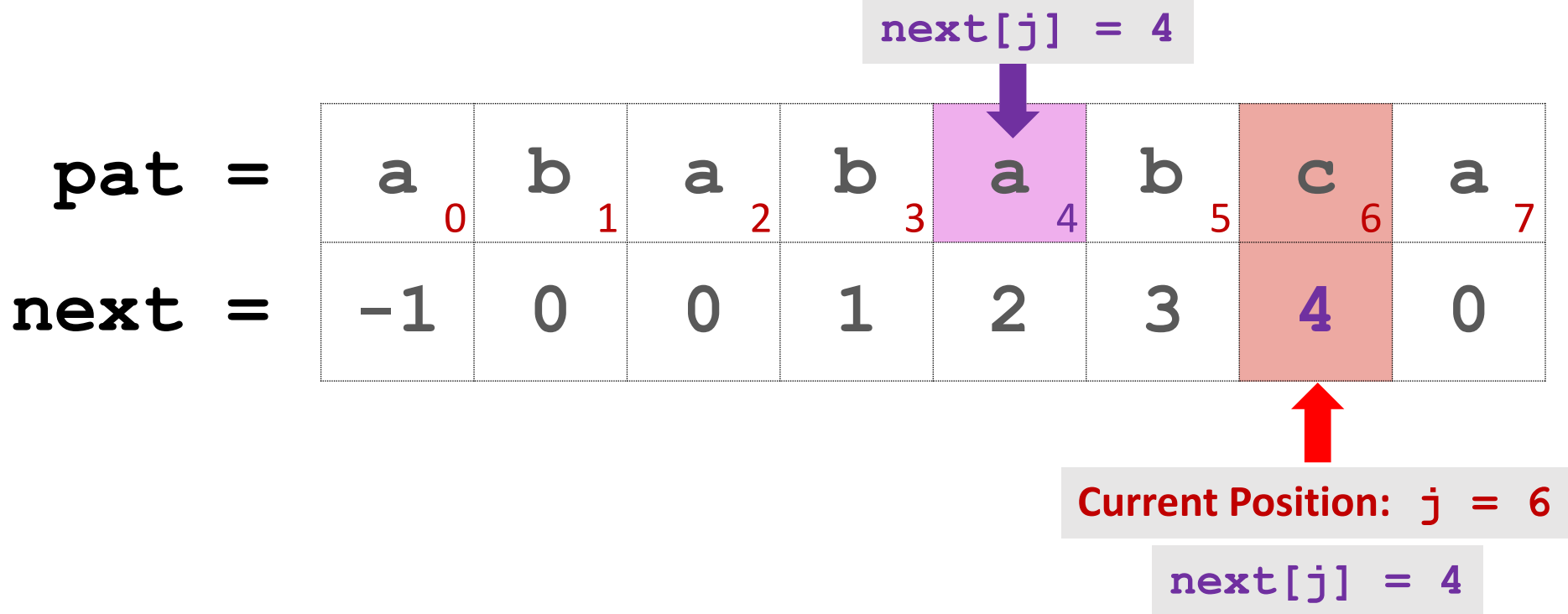
# Property of the Array

<b>pat</b> =	a	b	a	b	a	b	c	a
	0	1	2	3	4	5	6	7
<b>next</b> =	-1	0	0	1	2	3	4	0

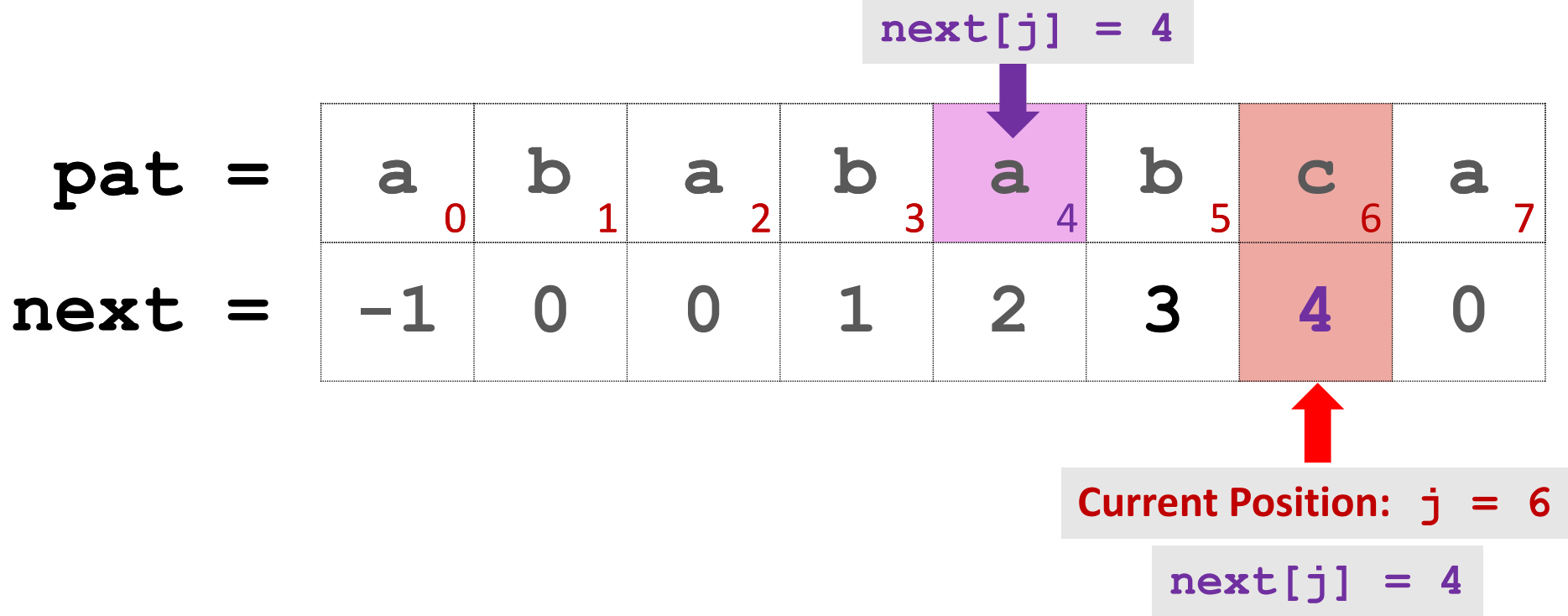
Current Position:  $j = 6$

$next[j] = 4$

# Property of the Array



# Property of the Array



# Property of the Array

`next[j] = 4`

**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7
-1	0	0	1	2	3	4	0

**next** =

Current Position:  $j = 6$

`next[j] = 4`

**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Property of the Array

pat =

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
-1	0	0	1	2	3	4	0

next =

next[j] = 4



Current Position: j = 6

next[j] = 4



pat =

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

# Property of the Array

<b>pat</b> =	a	b	a	b	a	b	c	a
	0	1	2	3	4	5	6	7
<b>next</b> =	-1	0	0	1	2	3	4	0

<b>pat</b> =	a	b	a	b	a	b	c	a
	0	1	2	3	4	5	6	7



# Property of the Array

**pat =**

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
-1	0	0	1	2	3	4	0

**next =**

**pat =**

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Match!

# Property of the Array

**pat** =

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
-1	0	0	1	2	3	4	0

**next** =

Match!

**pat** =

a <sub>0</sub>	b <sub>1</sub>	a <sub>2</sub>	b <sub>3</sub>	a <sub>4</sub>	b <sub>5</sub>	c <sub>6</sub>	a <sub>7</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

# KMP Algorithm for String Searching

# KMP Algorithm

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

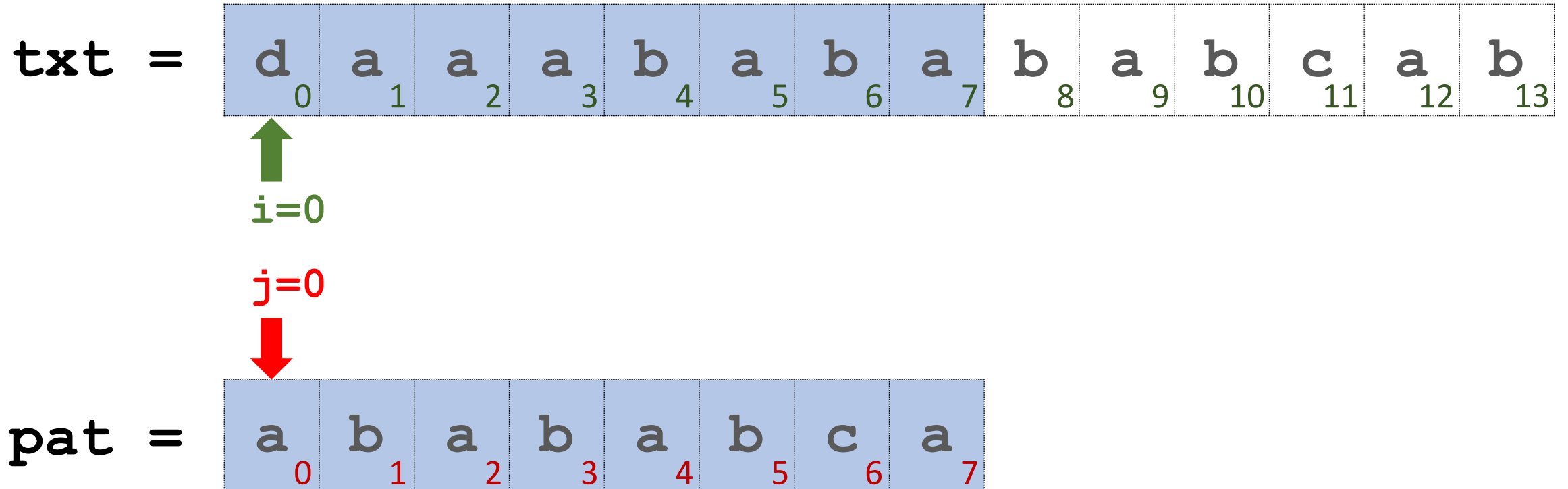
**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

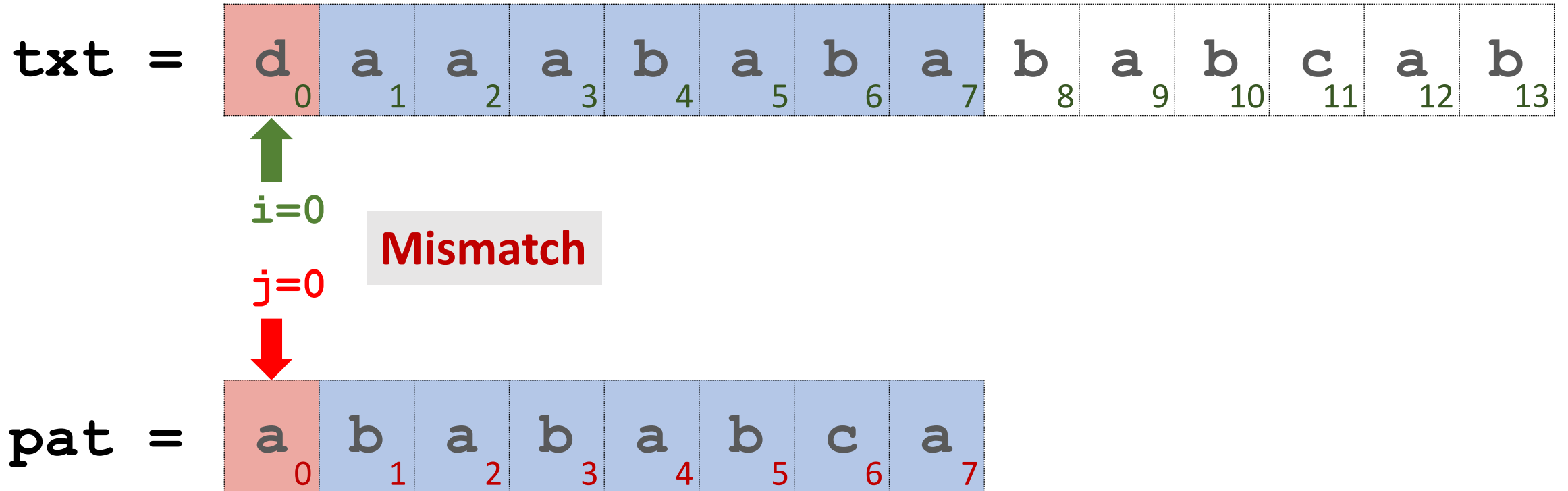
**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 1



# Iteration 1



# Iteration 1

**txt** =

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



i=0

j=0



**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next** =

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 1

**txt** =

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



$i=0$

$j=0$



**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next** =

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---



$\text{next}[j] = -1$



# Iteration 1

**txt** =

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=0**

↓  
**j=0**

Set  $j \leftarrow \text{next}[j]$

**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next** =

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 1

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



$i=0$

$j=-1$



Set  $j \leftarrow \text{next}[j]$

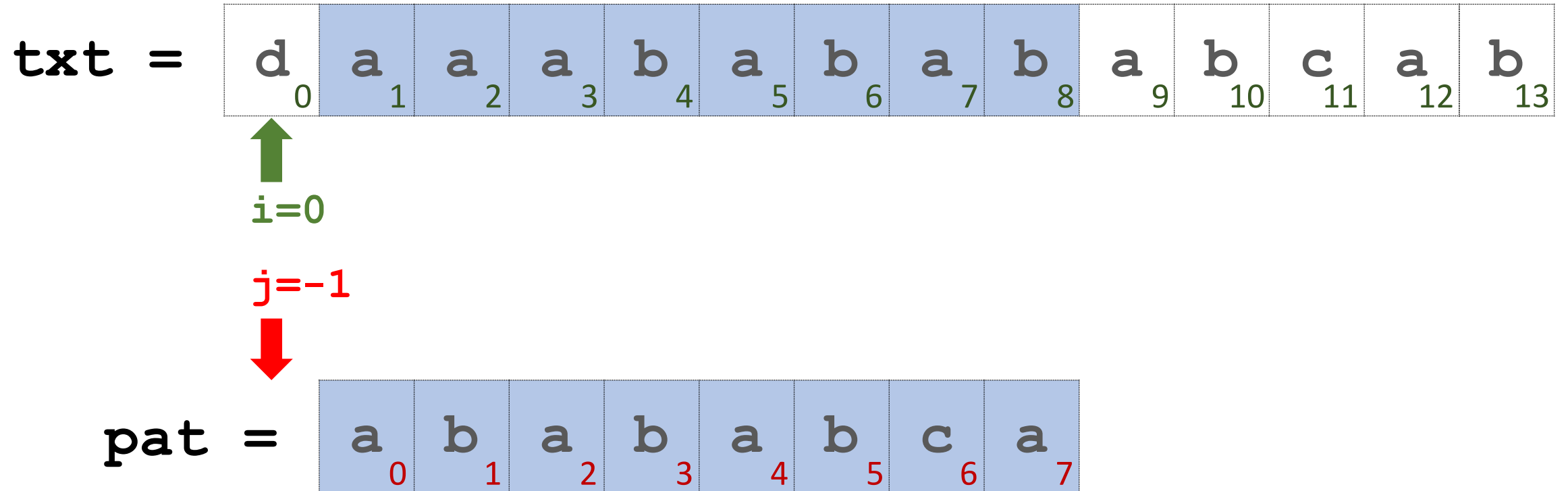
**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

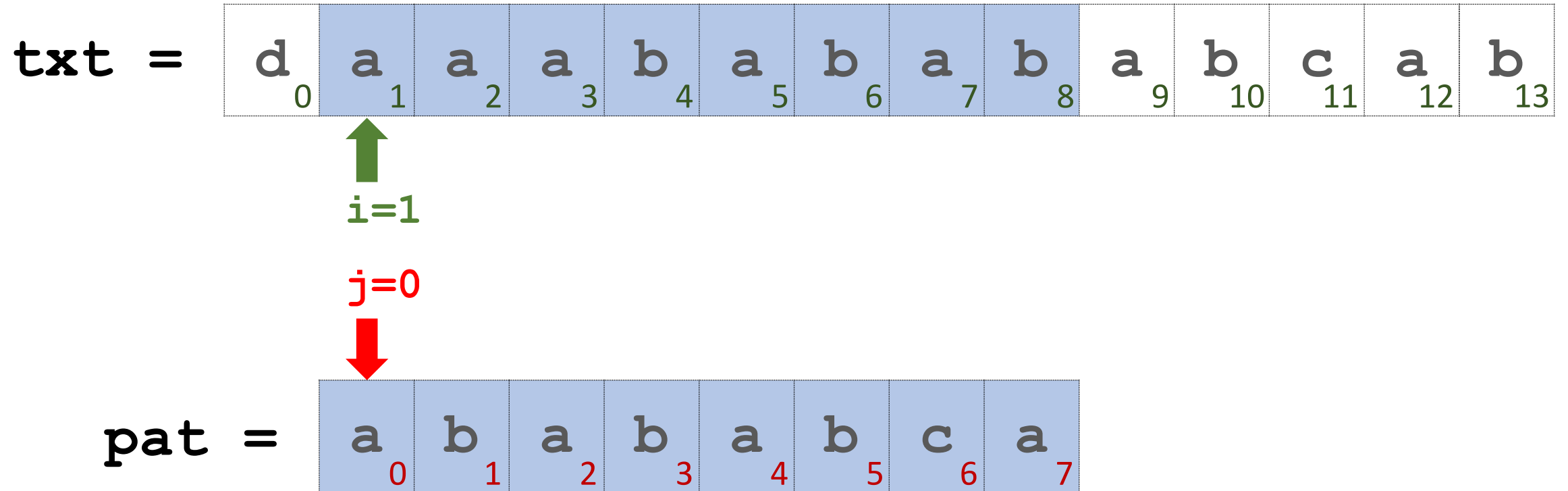
**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

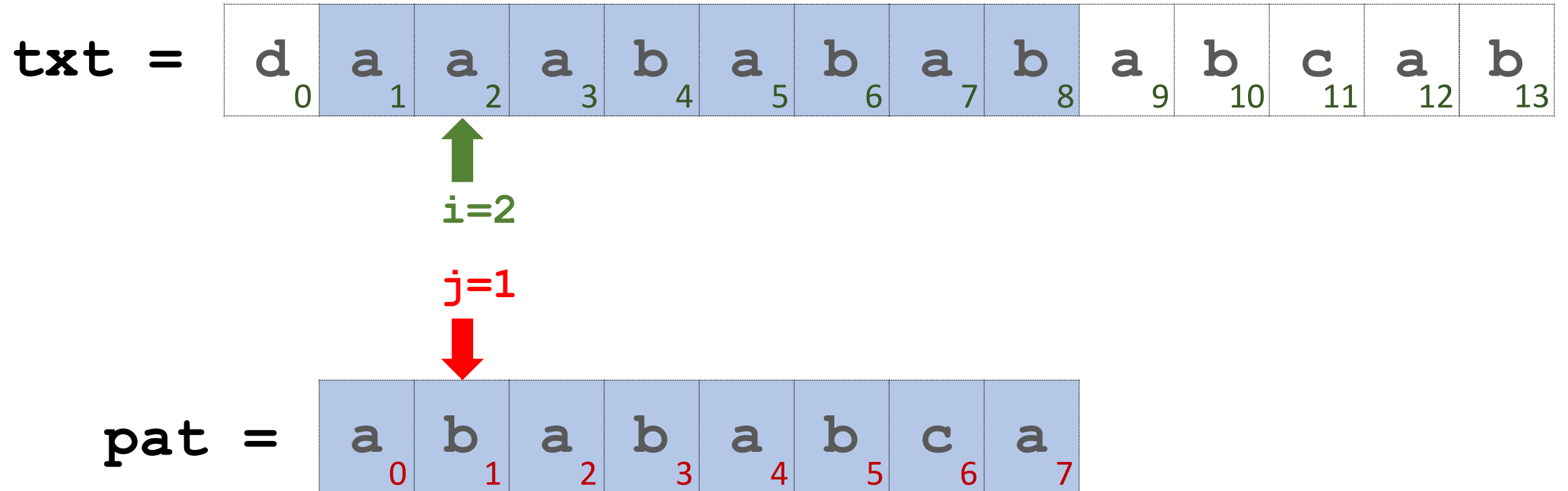
# Iteration 2



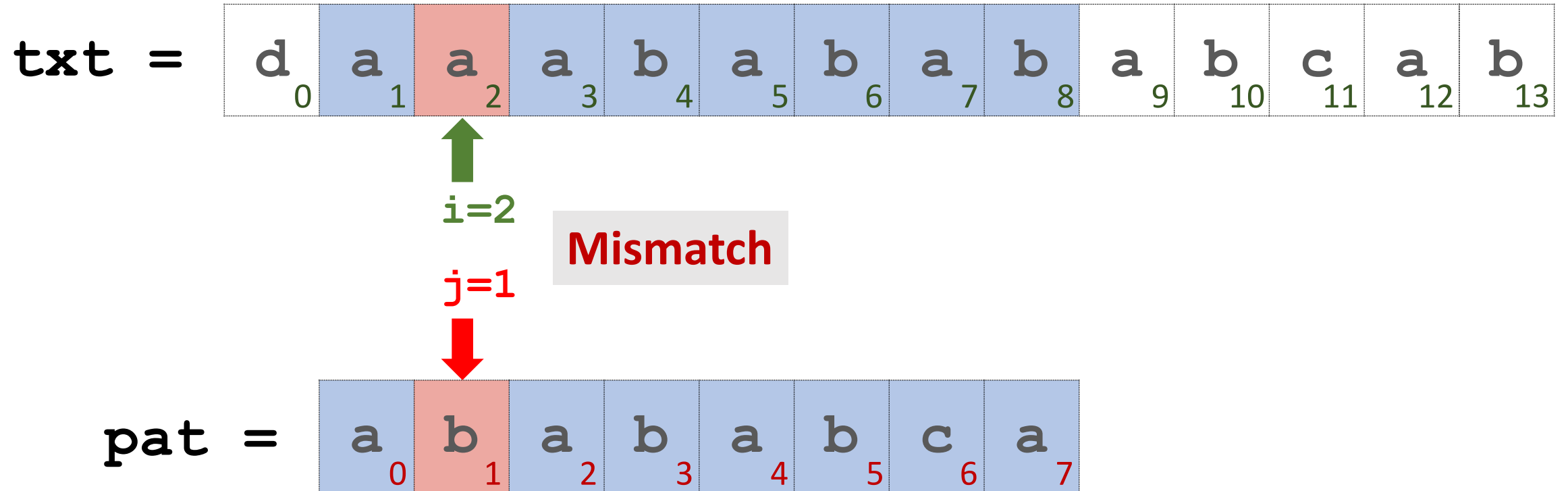
# Iteration 2



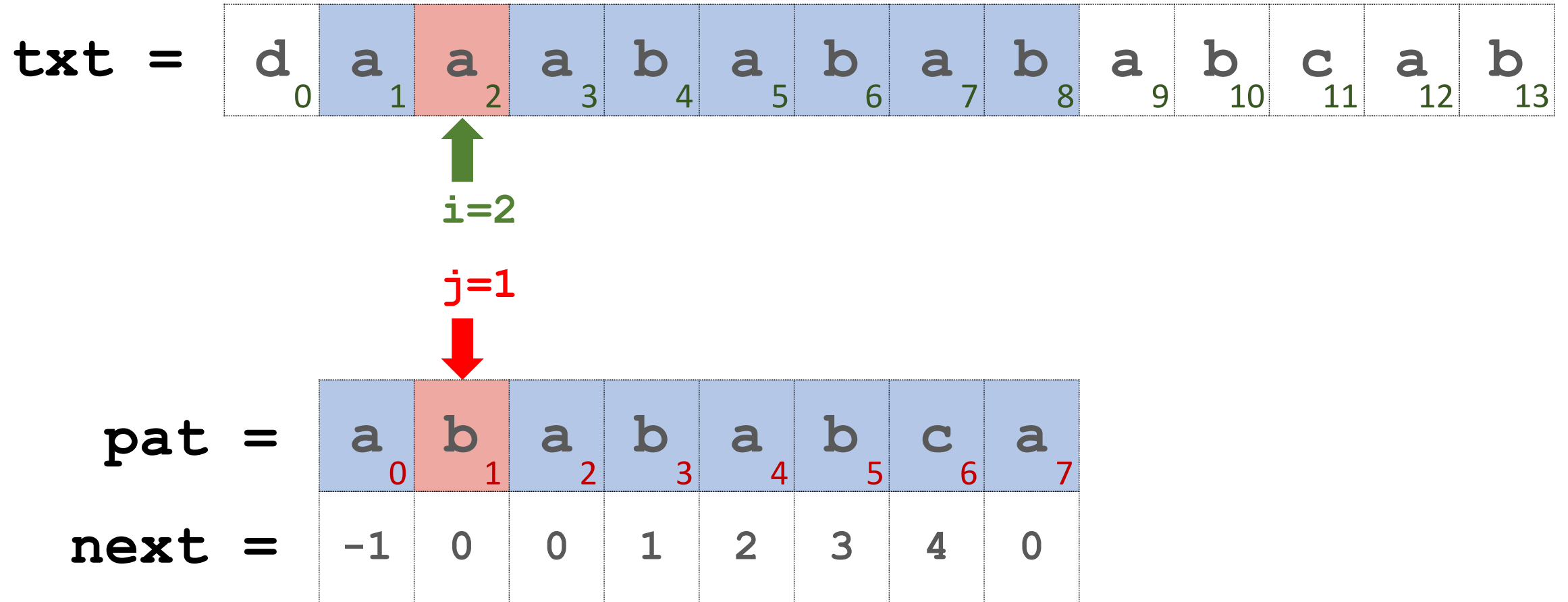
# Iteration 2



# Iteration 2



# Iteration 2



# Iteration 2

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



**i=2**

**j=1**



**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

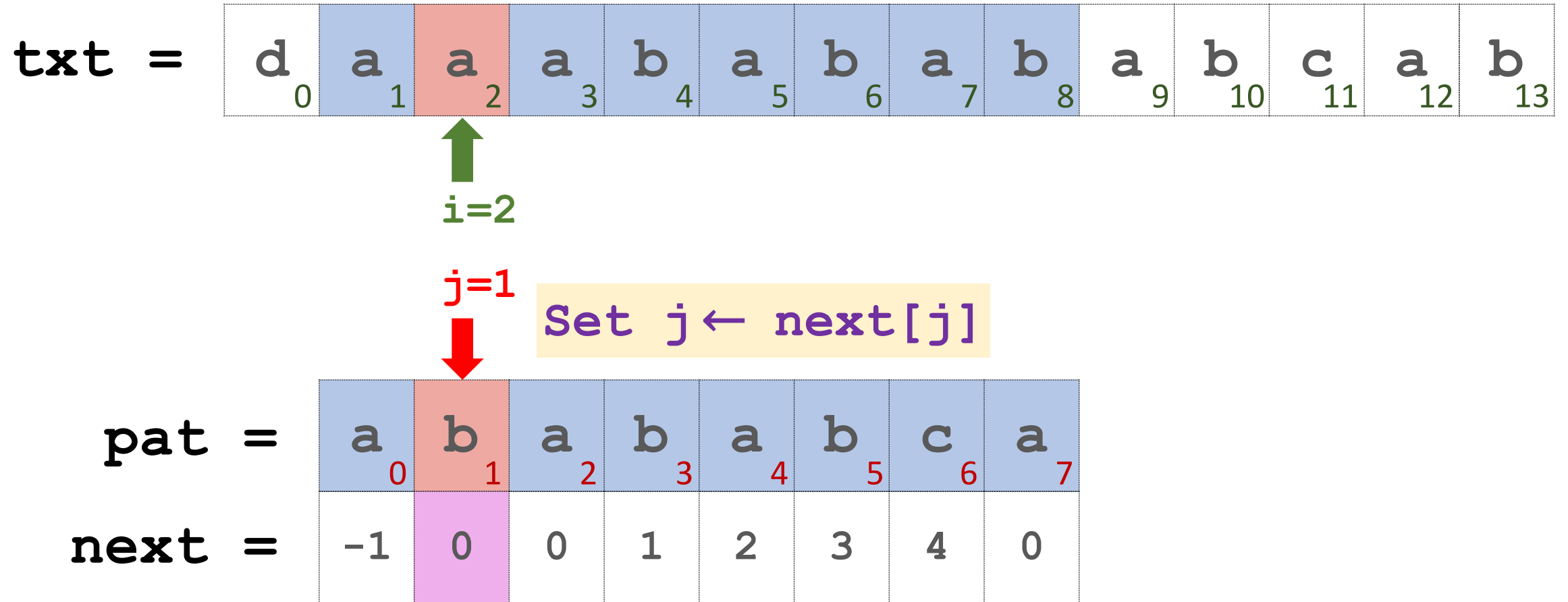
-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---



**next[j] = 0**



# Iteration 2



# Iteration 2

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



i=2

j=0



Set  $j \leftarrow \text{next}[j]$

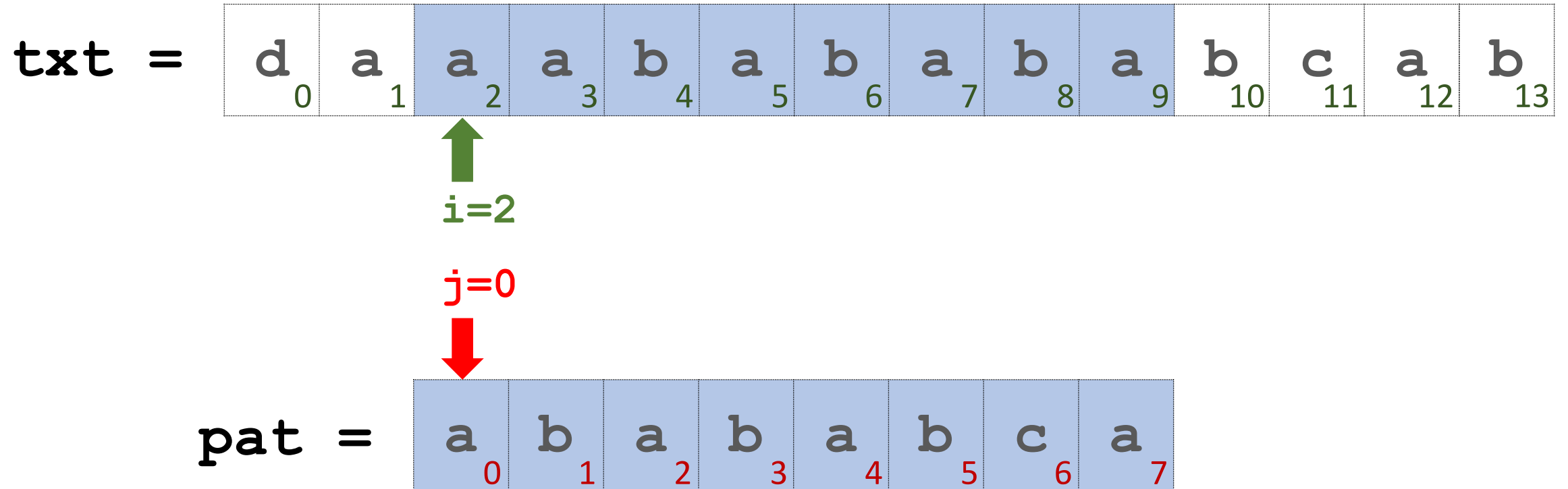
**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

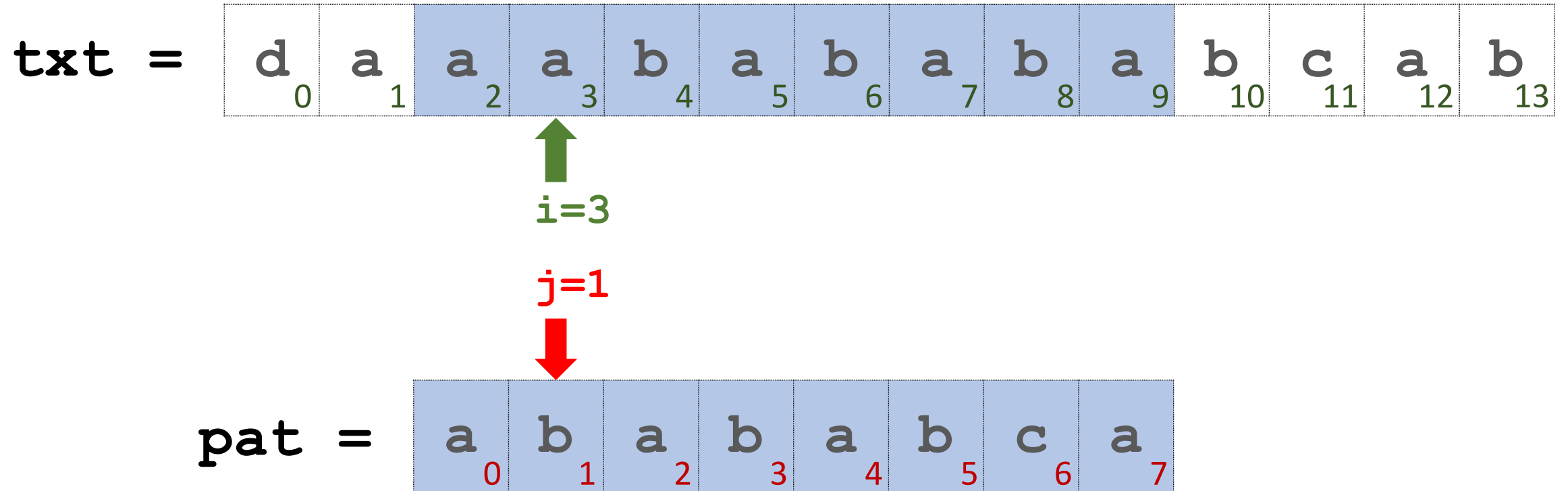
**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

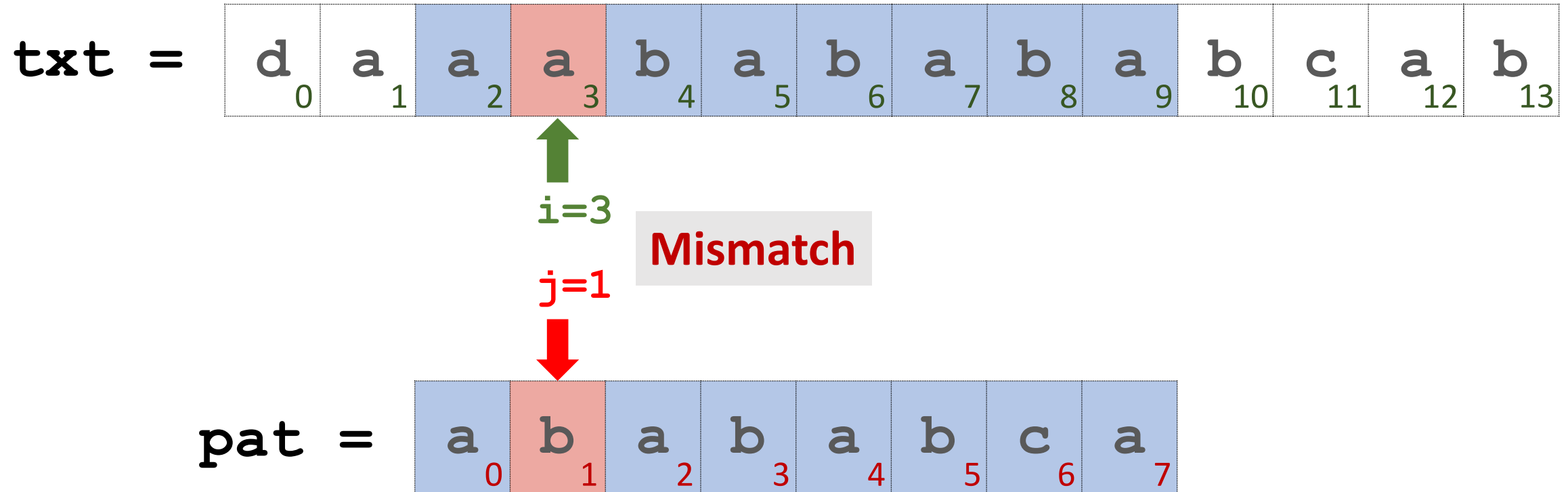
# Iteration 3



# Iteration 3



# Iteration 3



# Iteration 3

**txt** =

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=3**

**j=1**  
↓

**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next** =

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 3

**txt** =

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=3**

↓  
**j=1**

**pat** =

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next** =

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

↑  
**next[j] = 0**

# Iteration 3

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=3**

↓  
**j=1**

Set  $j \leftarrow \text{next}[j]$

**pat =**

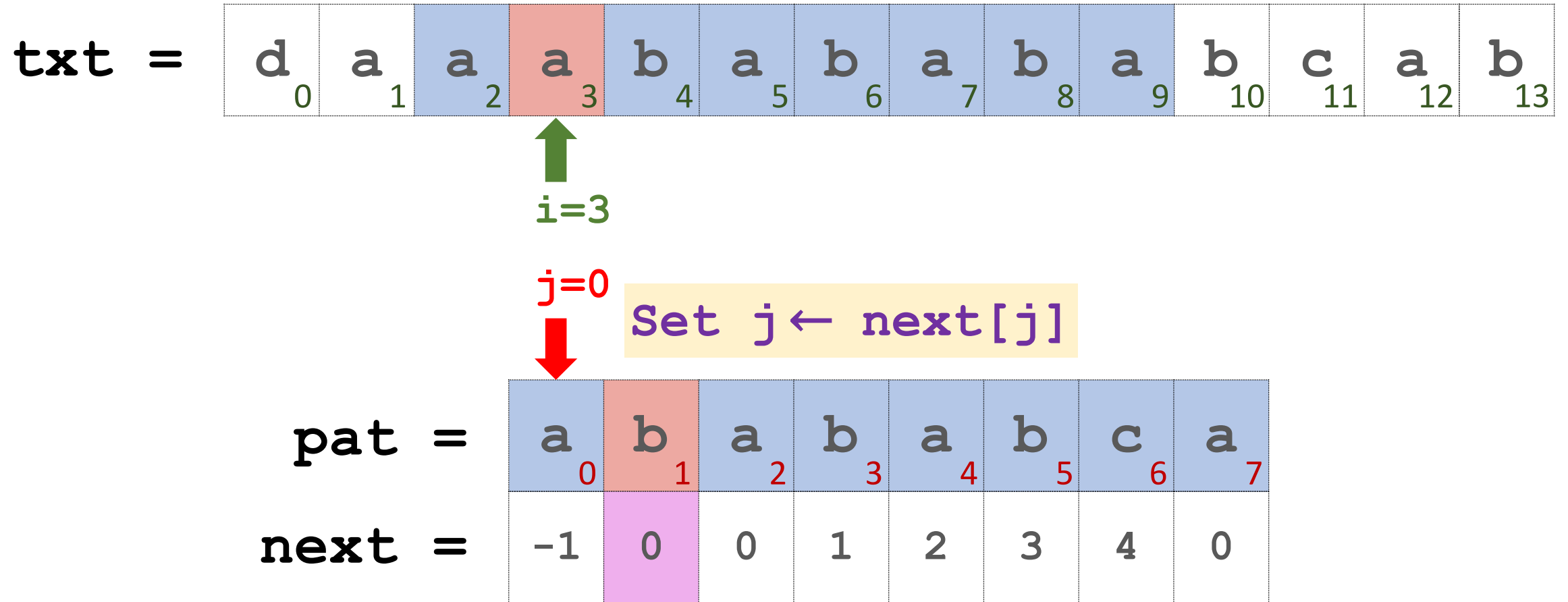
a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

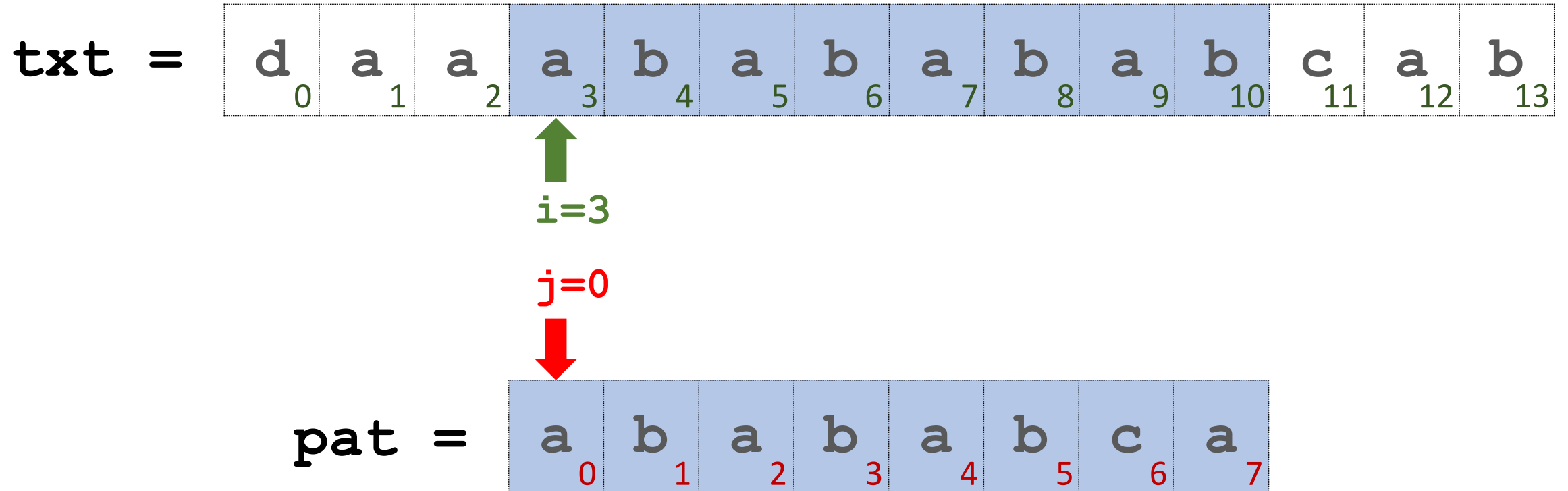
-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---



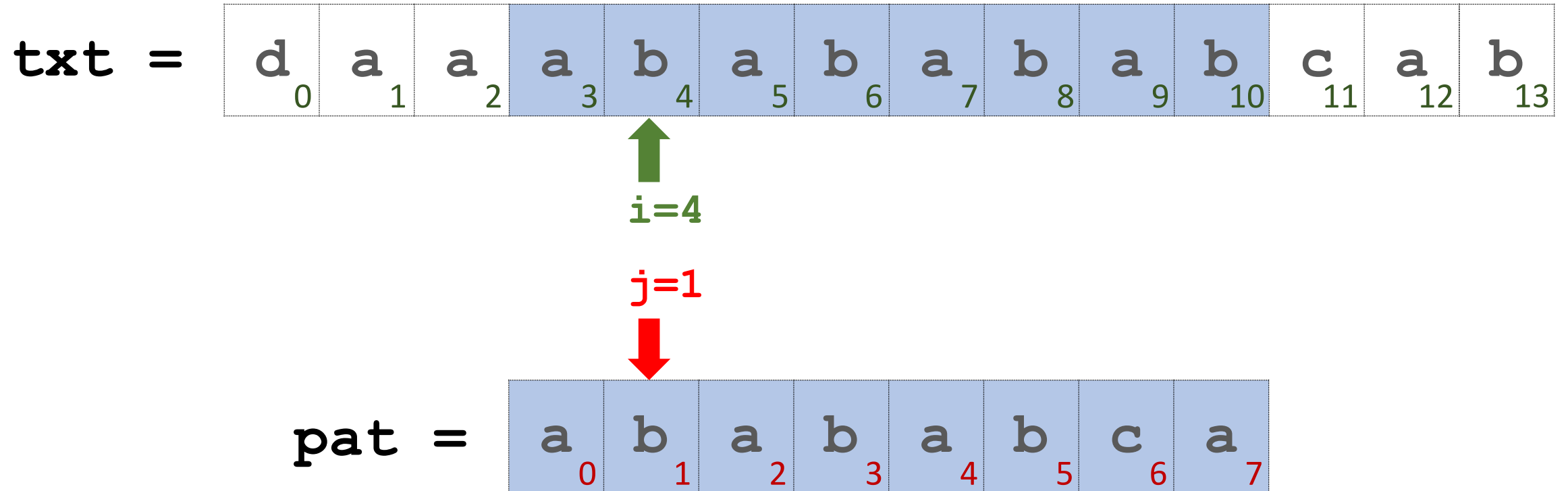
# Iteration 3



# Iteration 4



# Iteration 4



# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=5**

**j=2**  
↓

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



**i=6**

**j=3**



**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=7**

↓  
**j=4**

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



**i=8**

**j=5**



**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=9**

**j=6**  
↓

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7



# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13



i=9

j=6



**Mismatch**

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
i=9

↓  
j=6

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=9**

**j=6**  
↓

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

↑  
**next[j] = 4**

# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
i=9

Set  $j \leftarrow \text{next}[j]$

↓  
j=6

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---


# Iteration 4

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

  
**i=9**

**Set  $j \leftarrow \text{next}[j]$**

**j=4**  


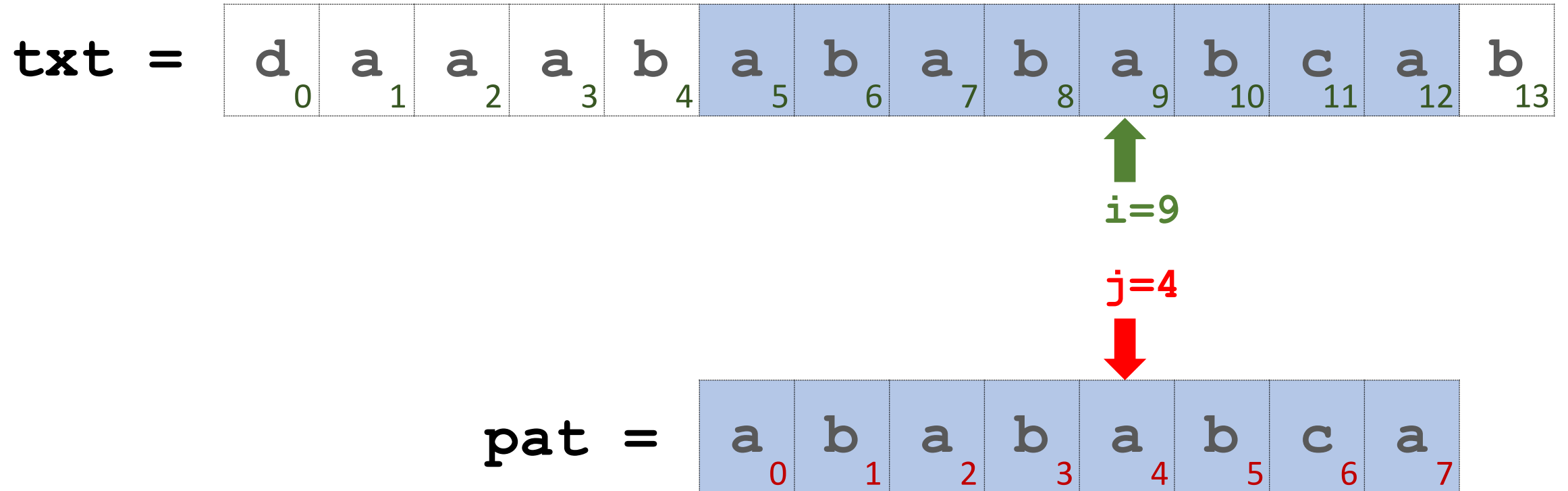
**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

**next =**

-1	0	0	1	2	3	4	0
----	---	---	---	---	---	---	---

# Iteration 5



# Iteration 5

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=10**

**j=5**  
↓

**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

# Iteration 5

**txt =**

d	a	a	a	b	a	b	a	b	a	b	c	a	b
0	1	2	3	4	5	6	7	8	9	10	11	12	13

↑  
**i=11**

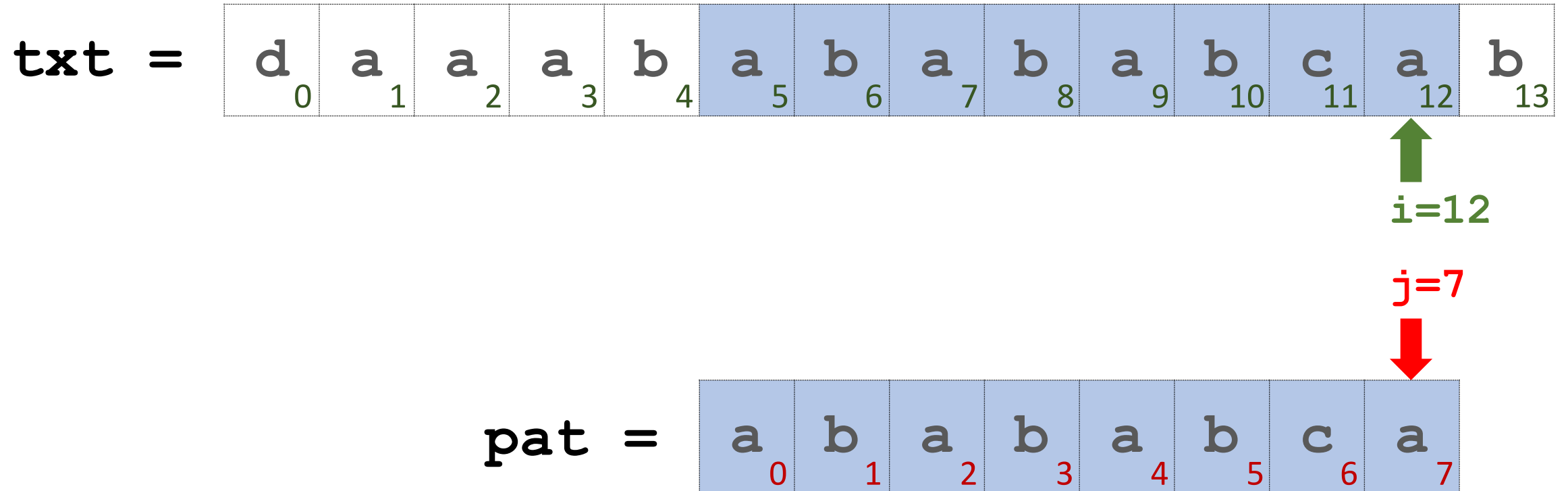
**pat =**

a	b	a	b	a	b	c	a
0	1	2	3	4	5	6	7

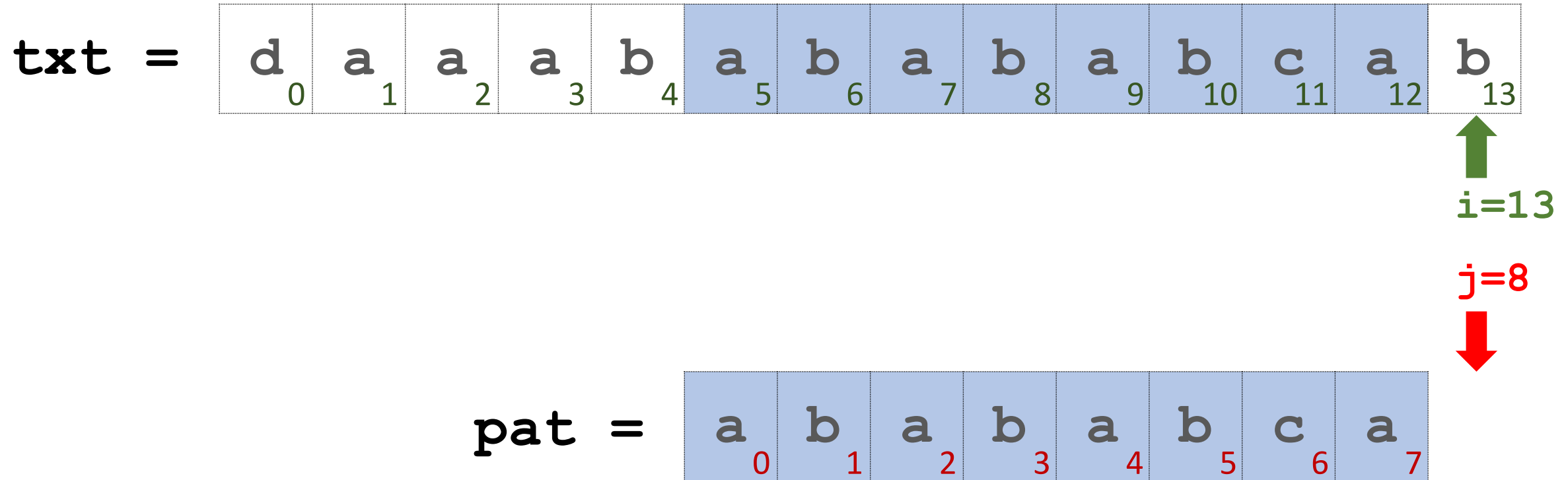
↓  
**j=6**



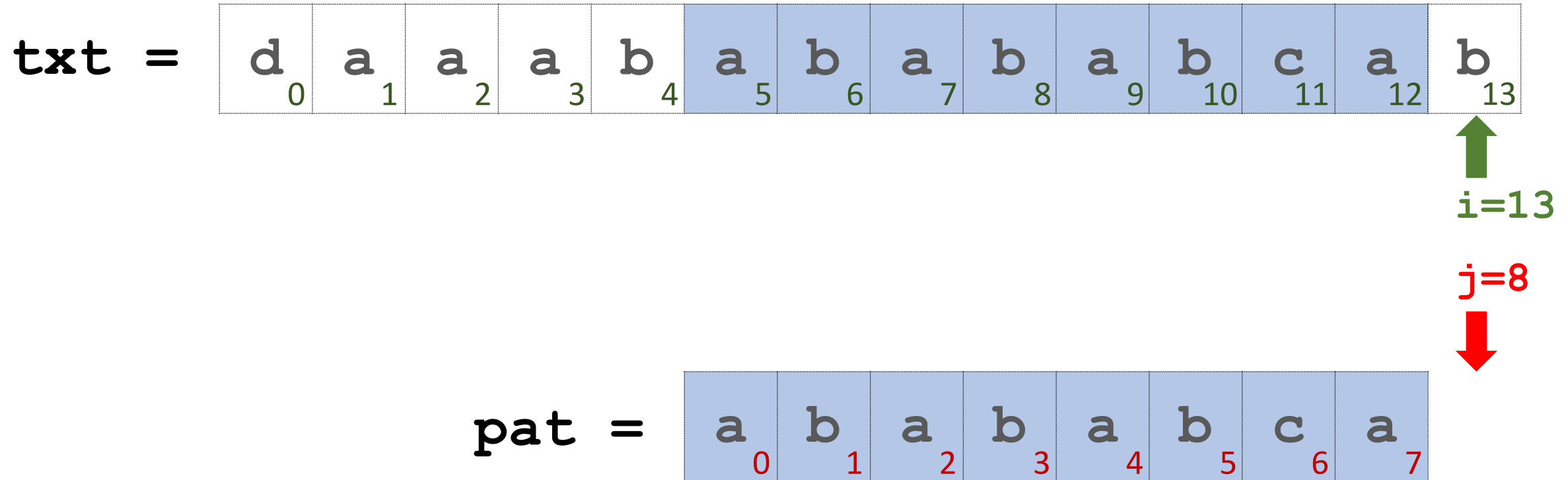
# Iteration 5



# Iteration 5

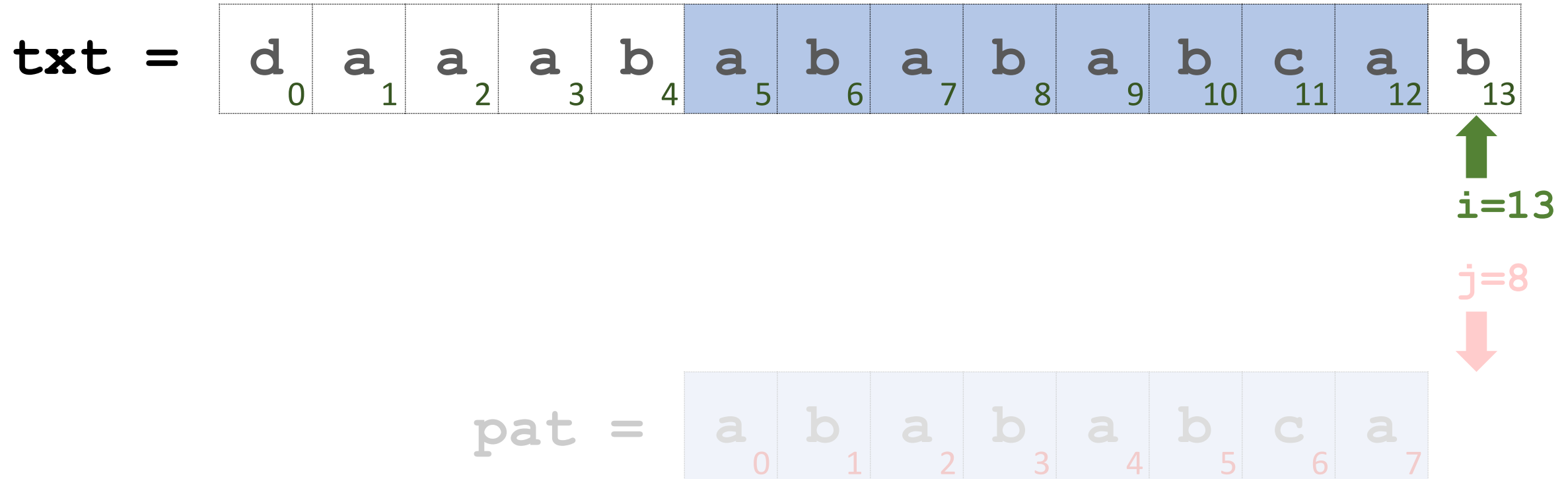


# End of Procedure



$j \geq |\text{pat}| \implies \text{Match is found!}$

# End of Procedure



Throughout, the pointer *i* has never moved leftward.

# Time Complexity

- $O(|pat|)$  for building the longest prefix suffix array.

# Time Complexity

- $O(|\text{pat}|)$  for building the longest prefix suffix array.
- $O(|\text{txt}|)$  for comparing string **pat** and string **txt**.
  - Throughout, the pointer *i* has never moved leftward.
  - Throughout, the pointer *i* has moved  $|\text{txt}|$  steps.
  - Totally  $O(|\text{txt}|)$  comparisons.

# Time Complexity


- $O(|\text{pat}|)$  for building the longest prefix suffix array.
- $O(|\text{txt}|)$  for comparing string **pat** and string **txt**.
  - Throughout, the pointer  $i$  has never moved leftward.
  - Throughout, the pointer  $i$  has moved  $|\text{txt}|$  steps.
  - Totally  $O(|\text{txt}|)$  comparisons.
- **Overall time complexity:**  $O(|\text{pat}| + |\text{txt}|)$ .

```
int KMP(char* txt, char* pat, int* next) {  
    int i = 0;  
    int j = 0;  
    while (i < strlen(txt) && j < strlen(pat)) {  
        if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
    else // no match is found  
        return -1;  
}
```





```
int KMP(char* txt, char* pat, int* next) {  
    int i = 0;  
    int j = 0;  
    while (i < strlen(txt) && j < strlen(pat)) {  
        if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
    else // no match is found  
        return -1;  
}
```

```
int KMP(char* txt, char* pat, int* next) {  
    ➡ int i = 0;  
    ➡ int j = 0;  
    while (i < strlen(txt) && j < strlen(pat)) {  
        if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
    else // no match is found  
        return -1;  
}
```

```
int KMP(char* txt, char* pat, int* next) {  
    int i = 0;  
    int j = 0;  
     while (i < strlen(txt) && j < strlen(pat)) {  
        if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
    else // no match is found  
        return -1;  
}
```

```
int KMP(char* txt, char* pat, int* next) {  
    int i = 0;  
    int j = 0;  
    while (i < strlen(txt) && j < strlen(pat)) {  
        ➡ if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        ➡ else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
    else // no match is found  
        return -1;  
}
```

```
int KMP(char* txt, char* pat, int* next) {
    int i = 0;
    int j = 0;
    while (i < strlen(txt) && j < strlen(pat)) {
        if (j == -1 || txt[i] == pat[j]) {
            i++; j++;
        }
        else // when mismatched
            j = next[j];
    }
     if (j == strlen(pat)) // found a match
        return i - j; // position of match
    else // no match is found
        return -1;
}
```

```
int KMP(char* txt, char* pat, int* next) {  
    int i = 0;  
    int j = 0;  
    while (i < strlen(txt) && j < strlen(pat)) {  
        if (j == -1 || txt[i] == pat[j]) {  
            i++; j++;  
        }  
        else // when mismatched  
            j = next[j];  
    }  
    if (j == strlen(pat)) // found a match  
        return i - j; // position of match  
     else // no match is found  
        return -1;  
}
```

**Thank You!**