Longest Substring

No Capital Corp | Poly Team

The Problem

Return the **length** of the **longest** substring **without repeating characters** in s.

Some Javascript Recap

Creating Hashtable - O(1)

```
let hashtable = {};

let hashtable = new Object();
```

Some Javascript Recap

Inserting to Hashtable - O(1)

```
hashtable.key1 = “value”;

hashtable[“key2”] = “value”;
```

Some Javascript Recap

Accessing Values in a Hashtable - O(1)

```
console.log(hashtable.key1);

console.log(hashtable[“key2”]);
```

Some Javascript Recap

Deleting in Hashtable - O(1)

```
delete hashtable.key1;

delete hashtable[“key2”];
```

Some Javascript Recap

Finding Max/Min - O(1)

```
let max_val = Math.max(2, 3);

let min_val = Math.min(2, 3);
```

Sliding Window

A sliding window looks like the following:

`[i, j]`

Sliding Window

To solve this problem, we use the following sliding window:

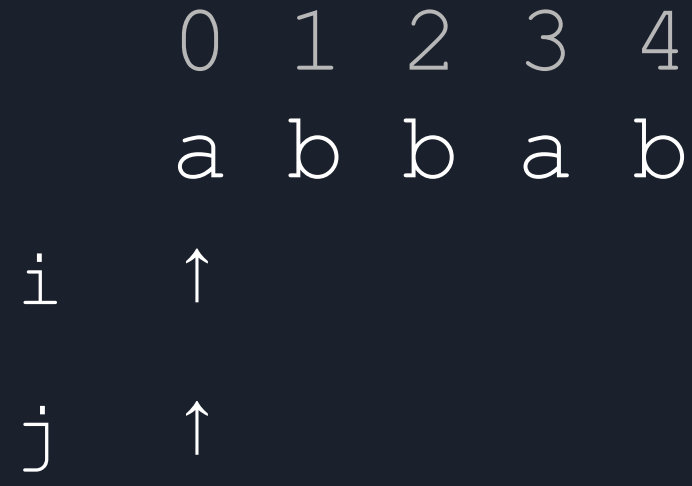
`[i, j)`

Example: Sliding Window

```
    0  1  2  3  4
    a  b  b  a  b
i   ↑
j   ↑
```

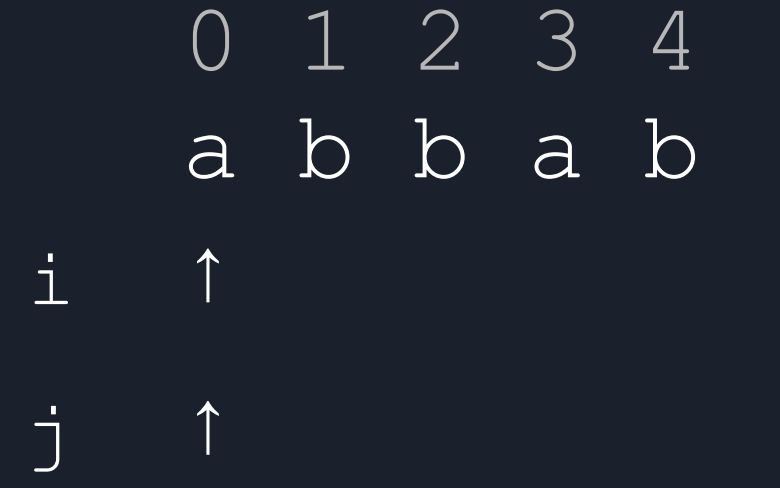
[i, j) = ???

Example: Sliding Window



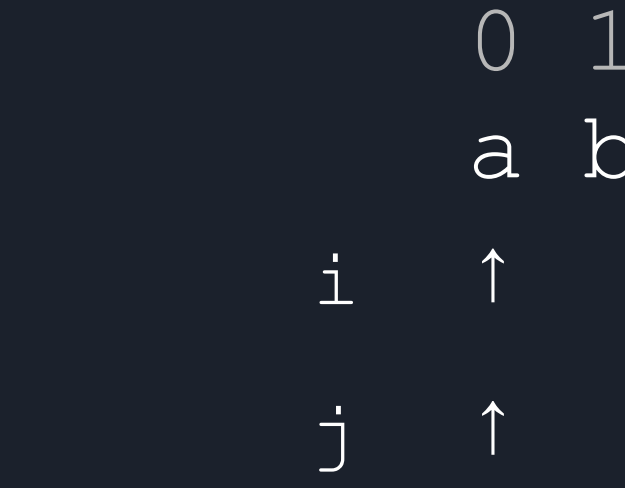
[i, j) = [0, 0)

Example: Sliding Window



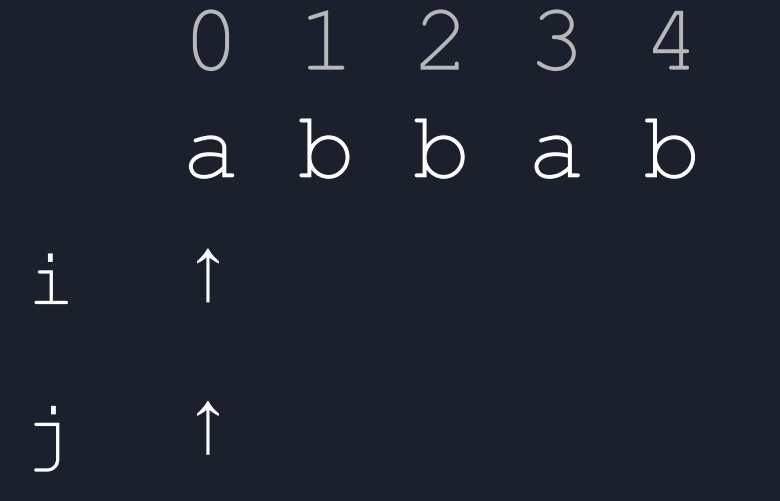
[i, j) = **[0, 0)** # of elements = **???**

Example: Sliding Window



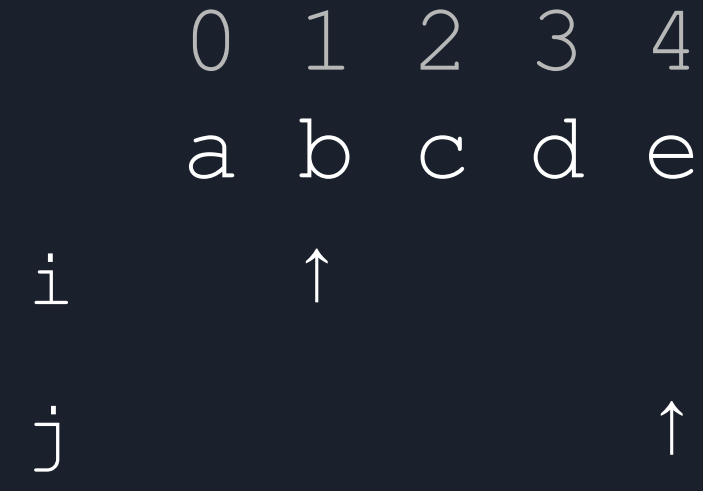
[i, j) = **[0, 0)** # of elements = **0**

Example: Sliding Window



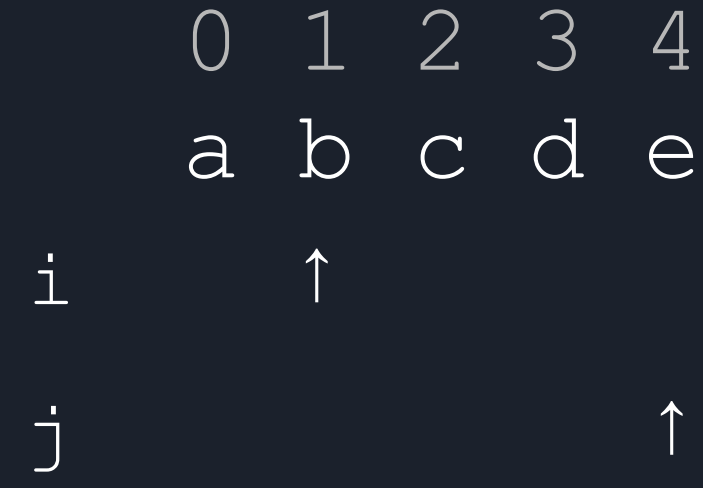
[i, j) = [0, 0) set = {}

Example: Sliding Window



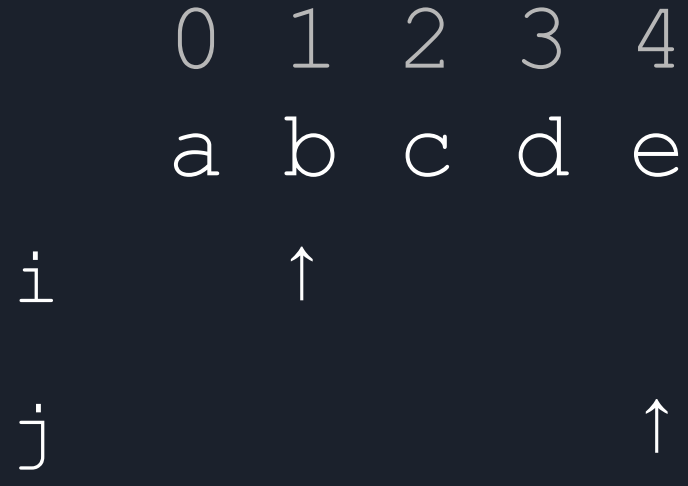
[i, j) = ??? set = ???

Example: Sliding Window



[i, j) = **[1, 4)** set = **???**

Example: Sliding Window



[i, j) = [1, 4) set = **{b, c, d}**

The Algorithm

- Maintain a set of characters in the current window

The Algorithm

- Maintain a set of characters in the current window
- Iterate if `j < s.length` and `i < s.length`
- Check if `s[j]` is in the set

The Algorithm

- Maintain a set of characters in the current window
- Iterate if `j < s.length` and `i < s.length`
- Check if `s[j]` is in the set
- If no, slide `j` further

The Algorithm

- …. …
- Iterate if `j < s.length` and `i < s.length`
- Check if `s[j]` is in the set
- If no, slide `j` further
- If yes, slide `i` to the right

The Algorithm

- …. …
- If yes, slide `i` to the right
- At some point, the set captures the required substring **(why?)**

Checking “abbab”

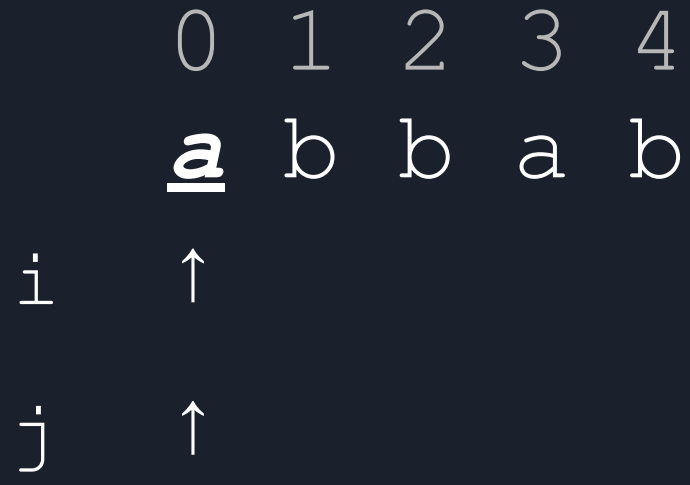
```
    0 1 2 3 4
    a b b a b
i   ↑
j   ↑
```

L = 0

[i, j) = [0, 0) set = {}

Checking “abbab”

Does the set contain ***a***?



L = 0

[i, j) = [0, 0) set = {}

Checking “abbab”

Does the set contain ***a***?

- No

```
  0 1 2 3 4
  a b b a b
i ↑
j ↑
```

L = 0

[i, j) = [0, 0) set = {}

Checking “abbab”

Does the set contain ***a***?

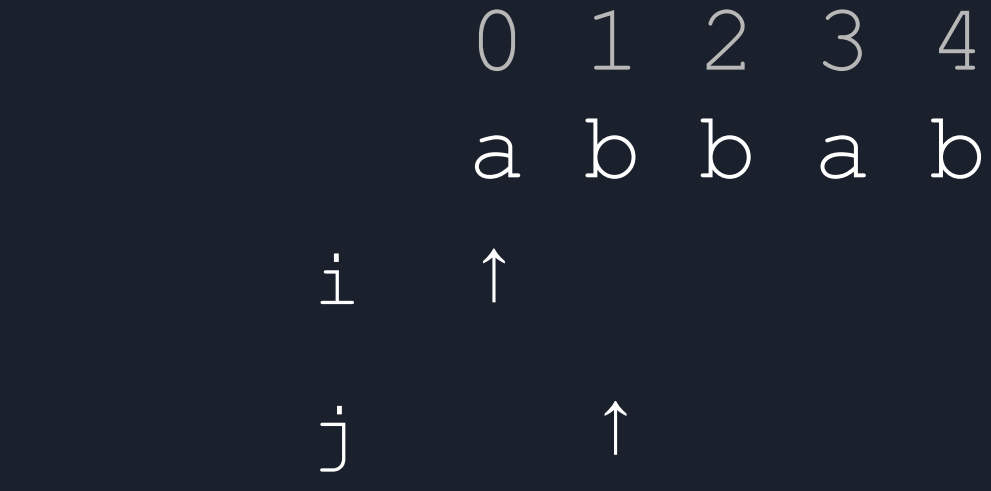
- No
- Slide `j`

```
  0 1 2 3 4
  a b b a b
i ↑
j ↑
```

L = 0

[i, j) = [0, 0) set = {}

Checking “abbab”

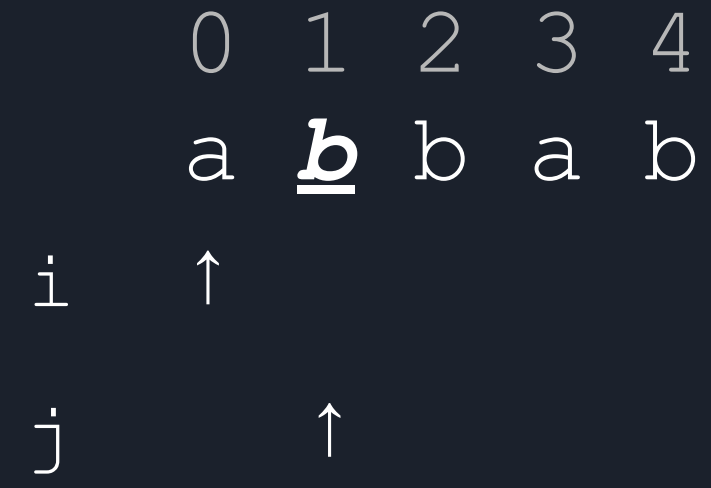


L = 1

[i, j) = **[0, 1)** set = **{a}**

Checking “abbab”

Does the set contain ***b***?



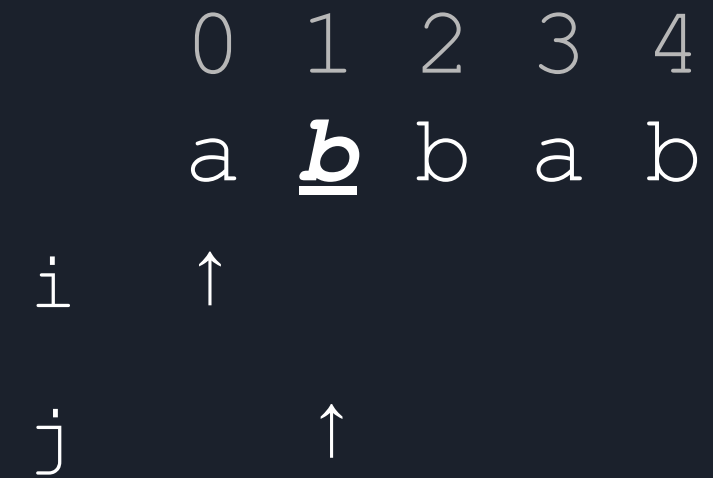
L = 1

[i, j) = **[0, 1)** set = **{a}**

Checking “abbab”

Does the set contain ***b***?

- No



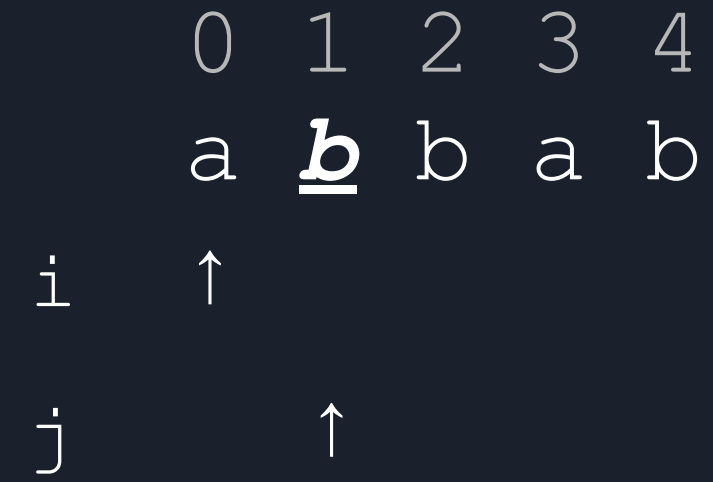
L = 1

[i, j) = **[0, 1)** set = **{a}**

Checking “abbab”

Does the set contain ***b***?

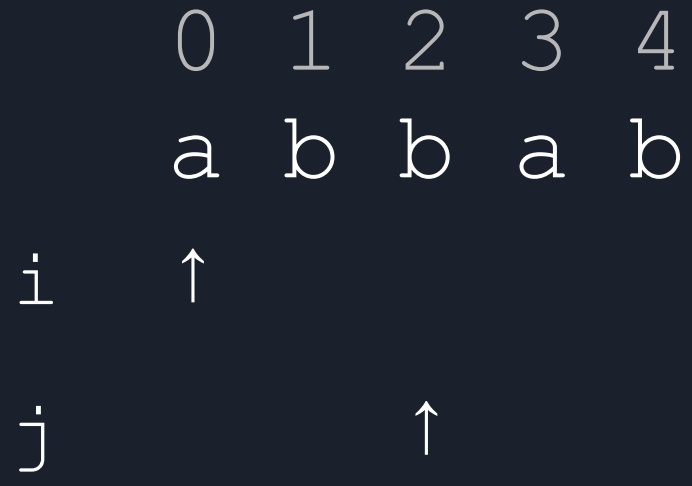
- No
- **Slide `j`**



L = 1

[i, j) = [0, 1) set = **{a}**

Checking “abbab”

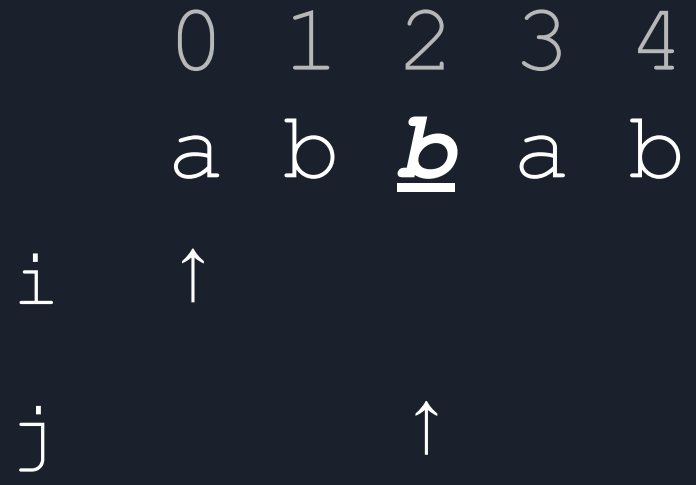


L = 2

[i, j) = **[0, 2)** set = **{a, b}**

Checking “abbab”

Does the set contain ***b***?



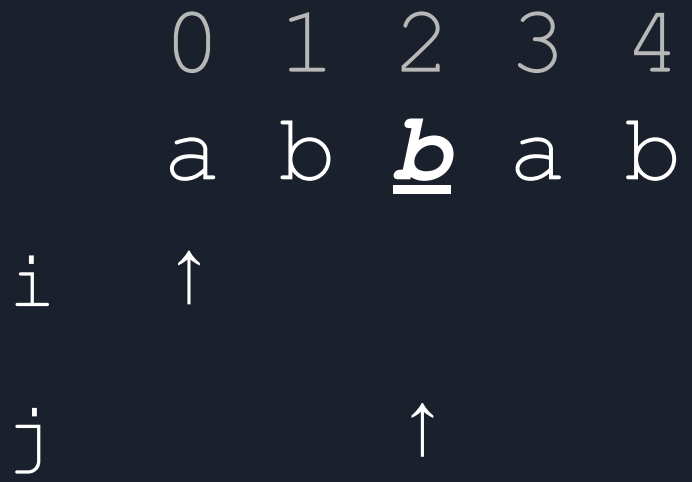
L = 2

[i, j) = [0, 2) set = **{a, b}**

Checking “abbab”

Does the set contain ***b***?

- Yes



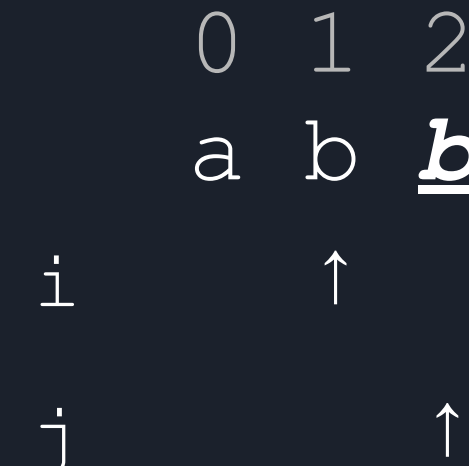
L = 2

[i, j) = [0, 2) **set = {a, b}**

Checking “abbab”

Does the set contain ***b***?

- Yes
- **Slide `i`**



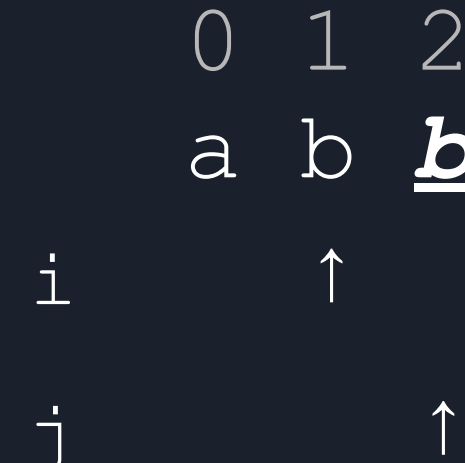
L = 2

[i, j) = [1, 2) set = {a, b}

Checking “abbab”

Does the set contain ***b***?

- Yes
- **Slide `i`**



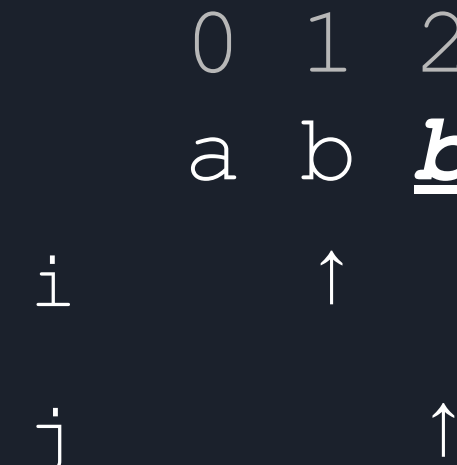
L = 1

[i, j) = [1, 2) set = {~~a, b~~}

Checking “abbab”

Does the set contain ***b***?

- Yes
- **Slide `i`**



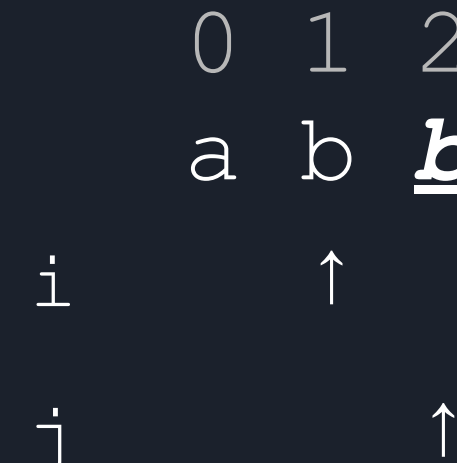
L = 1

[i, j) = [1, 2) set = {b}

Checking “abbab”

Does the set contain ***b***?

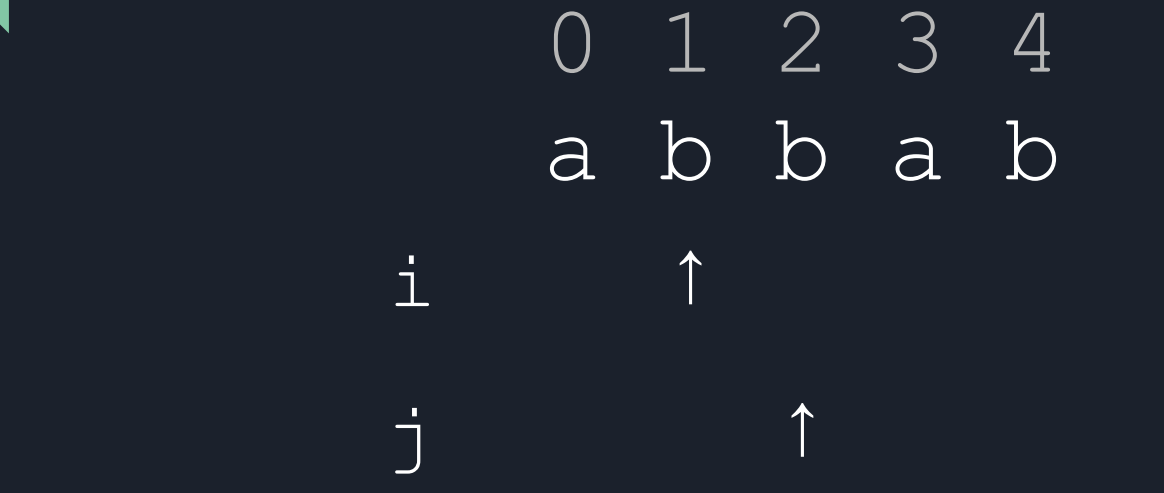
- Yes
- Slide `i`



L = 1

[i, j) = [1, 2) set = {b}

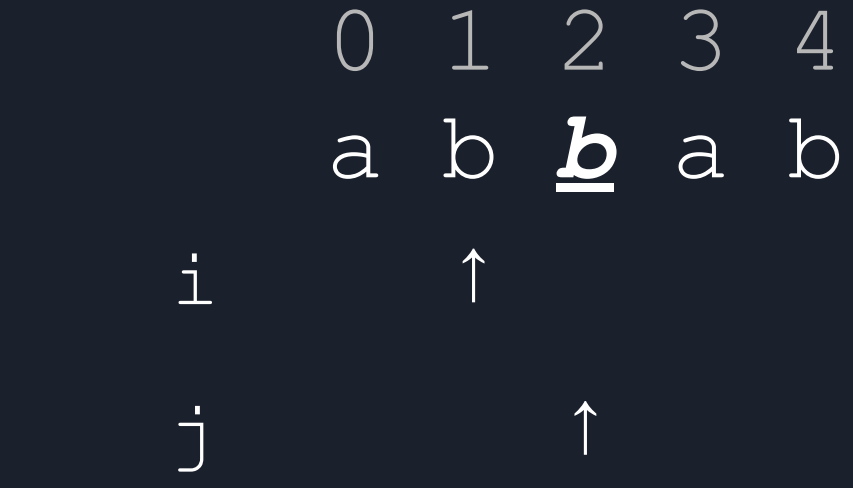
Checking “abbab”



L = 1

[i, j) = **[1, 2)** set = **{b}**

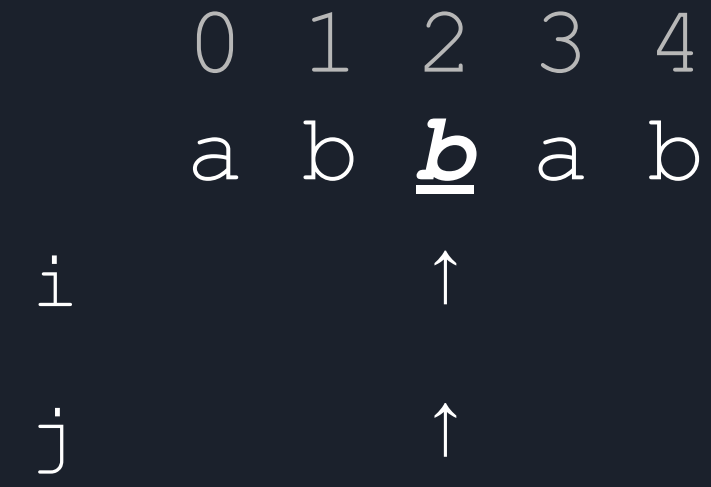
Checking “abbab”



L = 1

[i, j) = **[1, 2)** set = **{b}**

Checking “abbab”



L = 0

[i, j) = [2, 2) set = {}

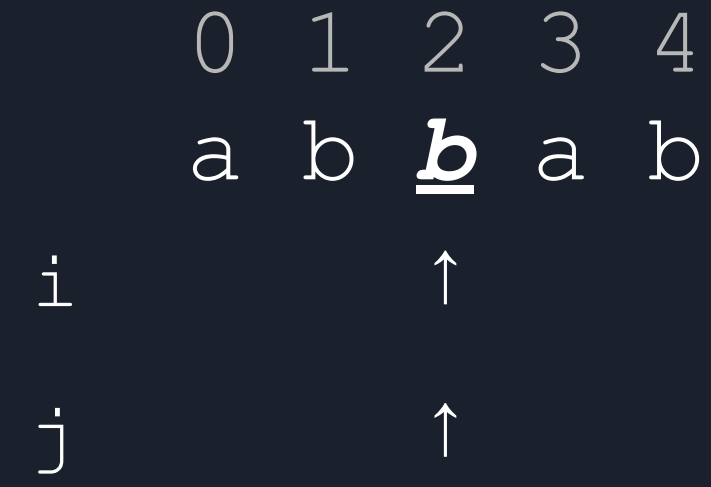
Checking “abbab”

```
    0 1 2 3 4
    a b b a b
i       ↑
j       ↑
```

L = 0

[i, j) = [2, 2) set = {}

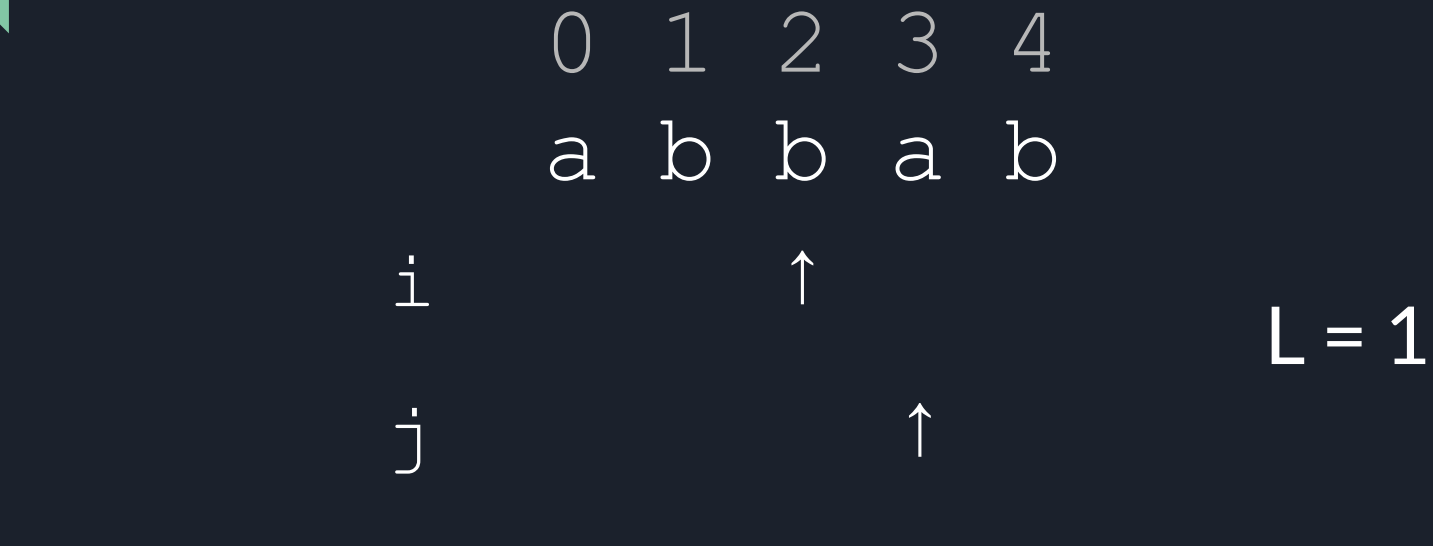
Checking “abbab”



L = 0

[i, j) = [2, 2) set = {}

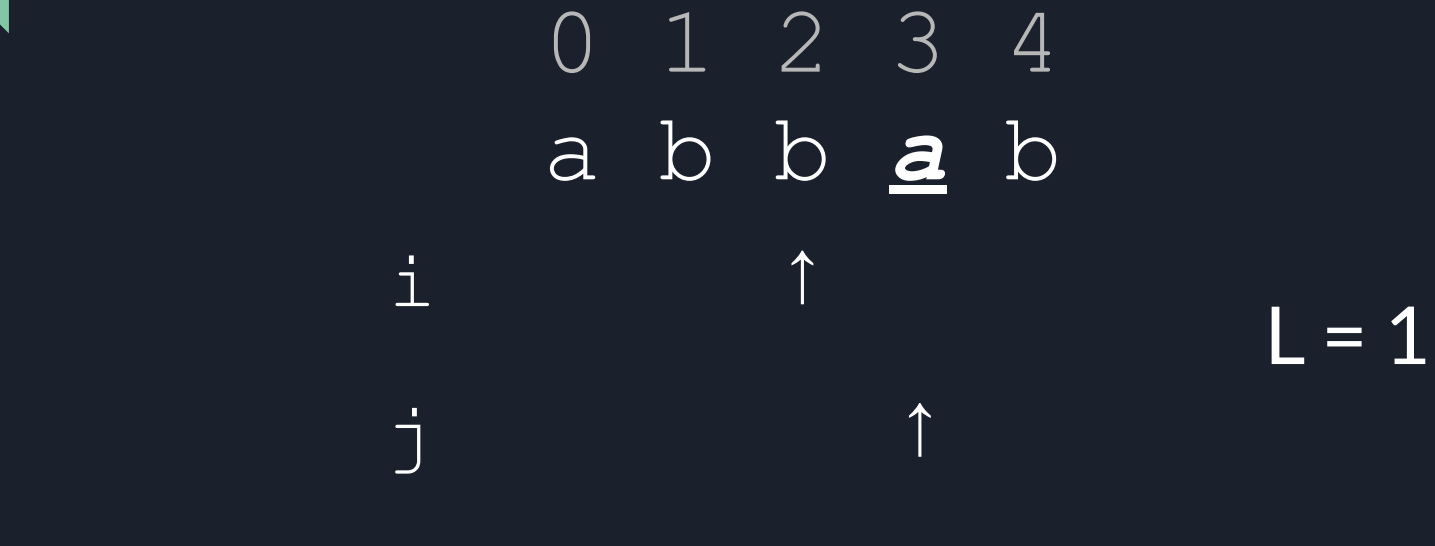
Checking “abbab”



L = 1

[i, j) = [2, 3) set = {b}

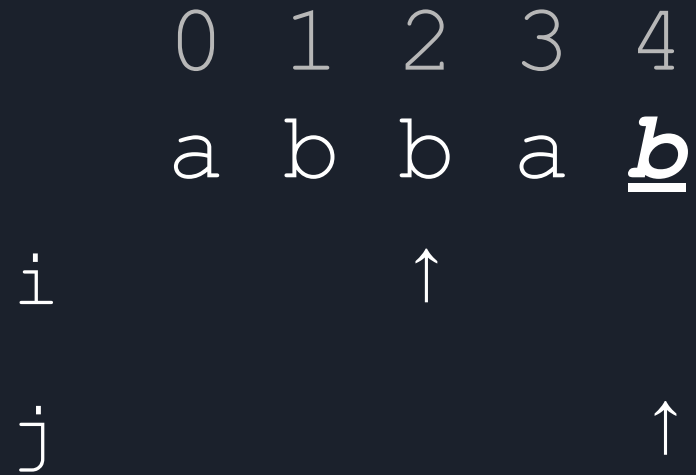
Checking “abbab”



L = 1

[i, j) = [2, 3) set = {b}

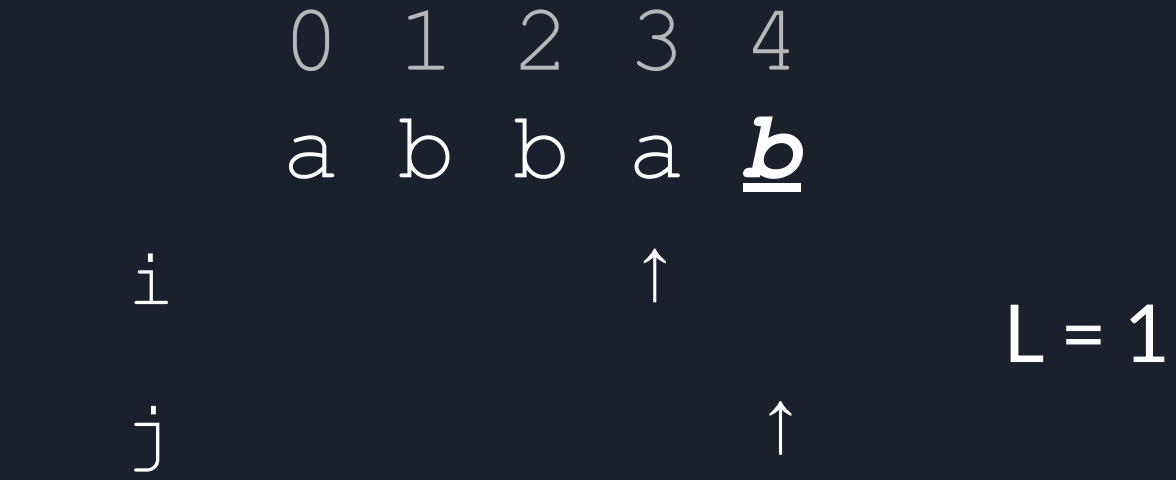
Checking “abbab”



L = 2

[i, j) = **[2, 4)** set = **{b, a}**

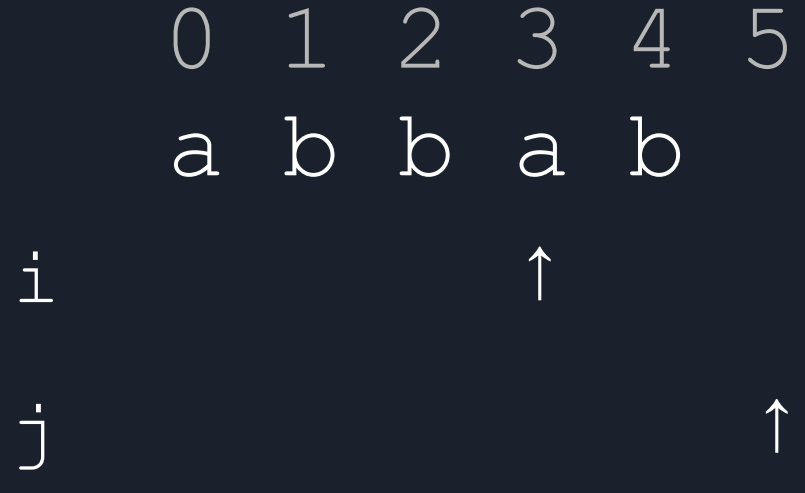
Checking “abbab”



L = 1

[i, j) = [3, 4) set = {a}

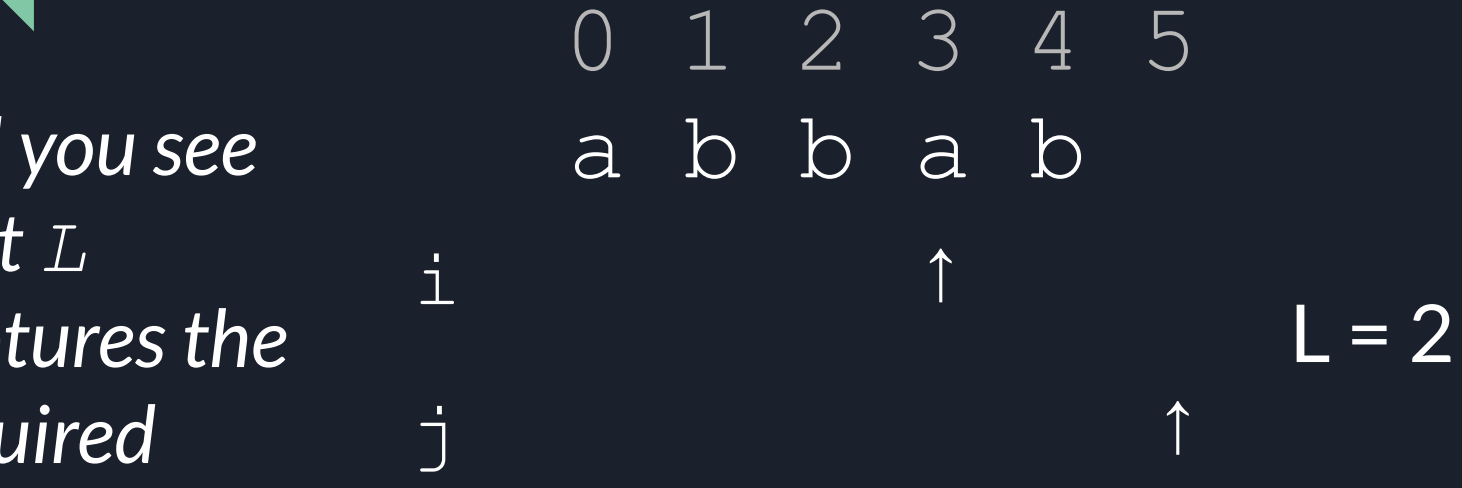
Checking “abbab”



L = 2

[i, j) = [3, 5) set = {a, b}

Checking “abbab”



Did you see that `L` captures the required length at some point?

[i, j) = **[3, 5)** set = **{a, b}**

Checking “pwkeww”

```
     0 1 2 3 4 5
     p w k e w w
i    ↑
j    ↑
```

L = 0
Max = 0

[i, j) = [0, 0) set = {}

Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	w	w
i	↑					
j	↑					

L = 0
Max = 0

[i, j) = [0, 0) set = {}

Checking “pwkeww”

```
    0 1 2 3 4 5
    p w k e w w
i   ↑
j     ↑
```

L = 1
Max = 1

[i, j) = **[0, 1)** set = **{p}**

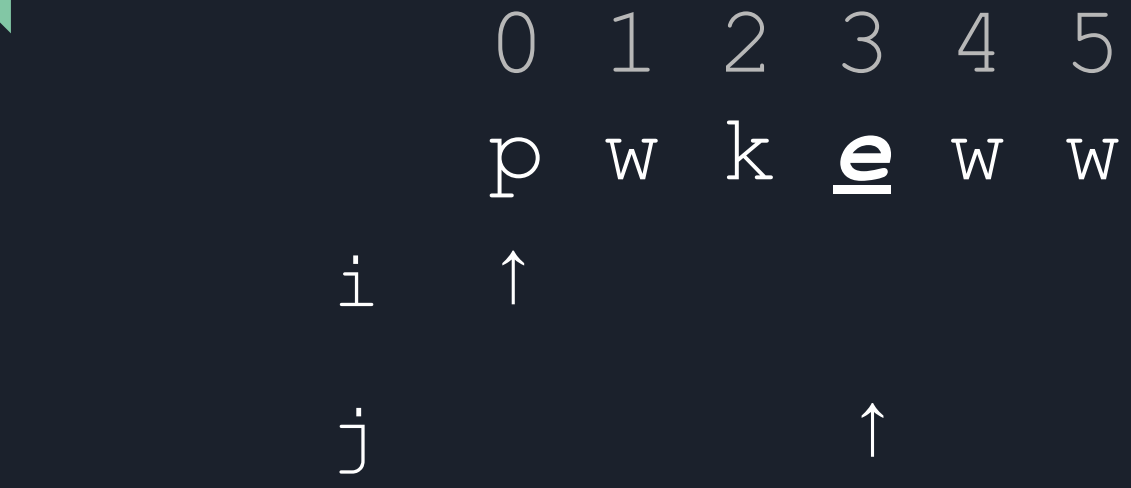
Checking “pwkeww”

	0	1	2	3	4	5
	p	w	***k***	e	w	w
i	↑					
j			↑			

L = 2
Max = 2

[i, j) = [0, 2) set = **{p, w}**

Checking “pwkeww”



L = 3
Max = 3

[i, j) = **[0, 3)** set = **{p, w, k}**

Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	***w***	w
i	↑					
j					↑	

L = 4
Max = 4

[i, j) = **[0, 4)** set = **{p, w, k, e}**

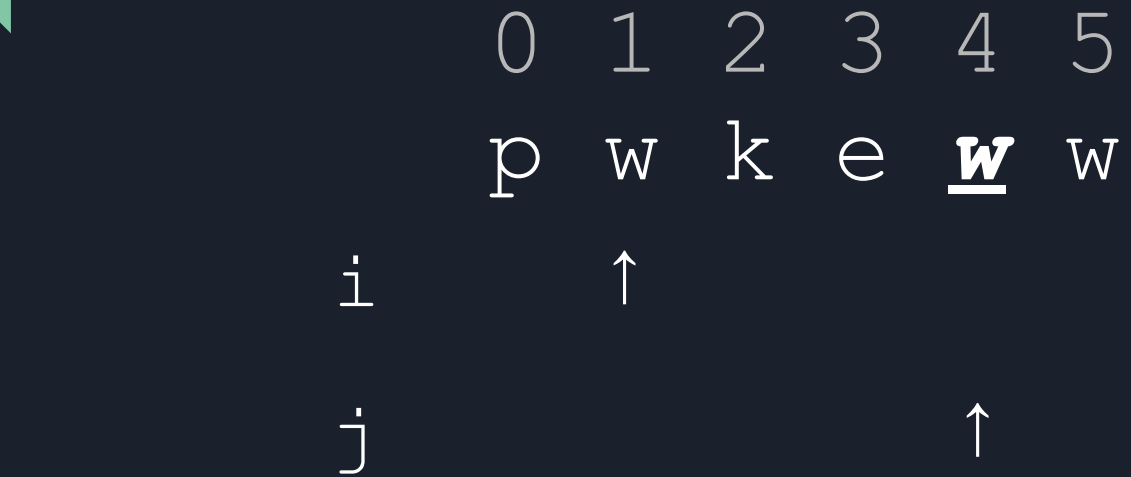
Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	***w***	w
i		↑				
j					↑	

L = 3
Max = 4

[i, j) = **[1, 4)** set = **{~~p,~~ w, k, e}**

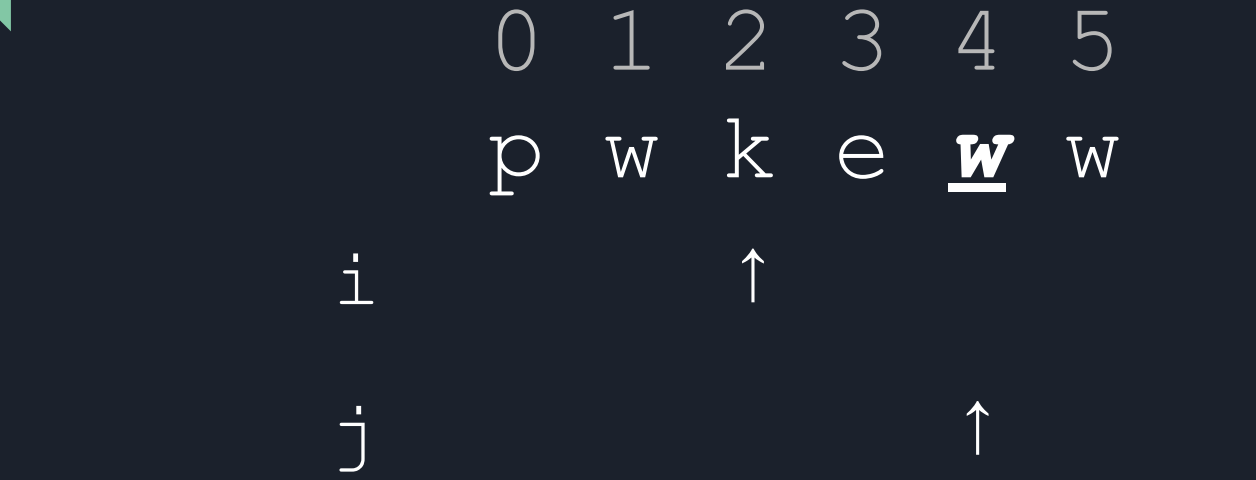
Checking “pwkeww”



L = 3
Max = 4

[i, j) = **[1, 4)** set = **{w, k, e}**

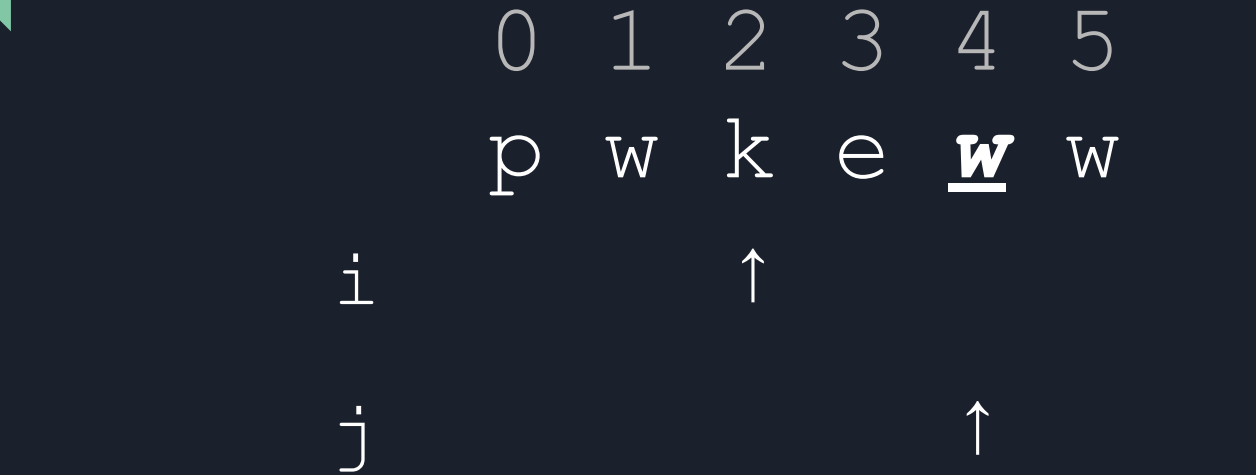
Checking “pwkeww”



L = 2
Max = 4

[i, j) = [2, 4) set = {~~w,~~ k, e}

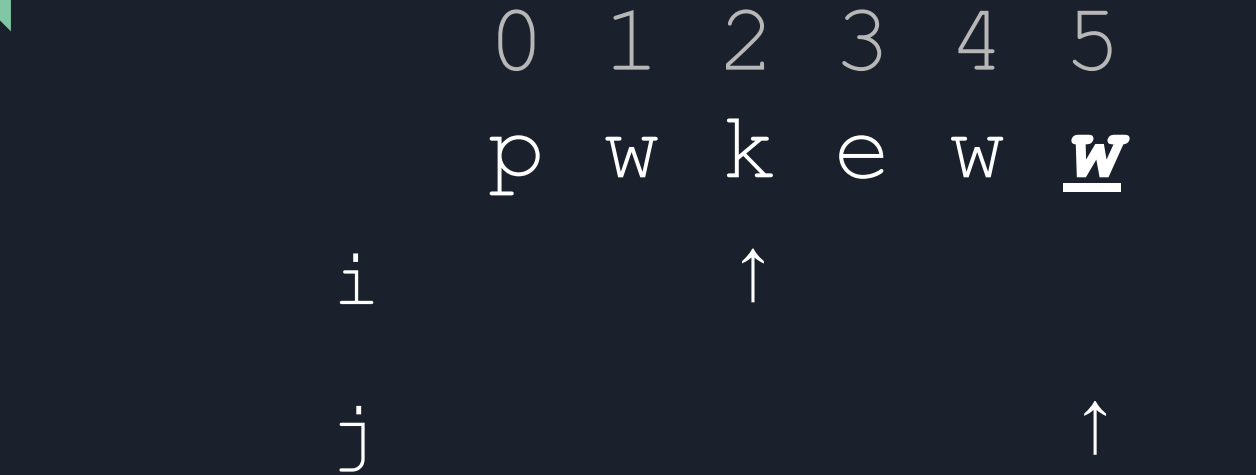
Checking “pwkeww”



L = 2
Max = 4

[i, j) = [2, 4) set = **{k, e}**

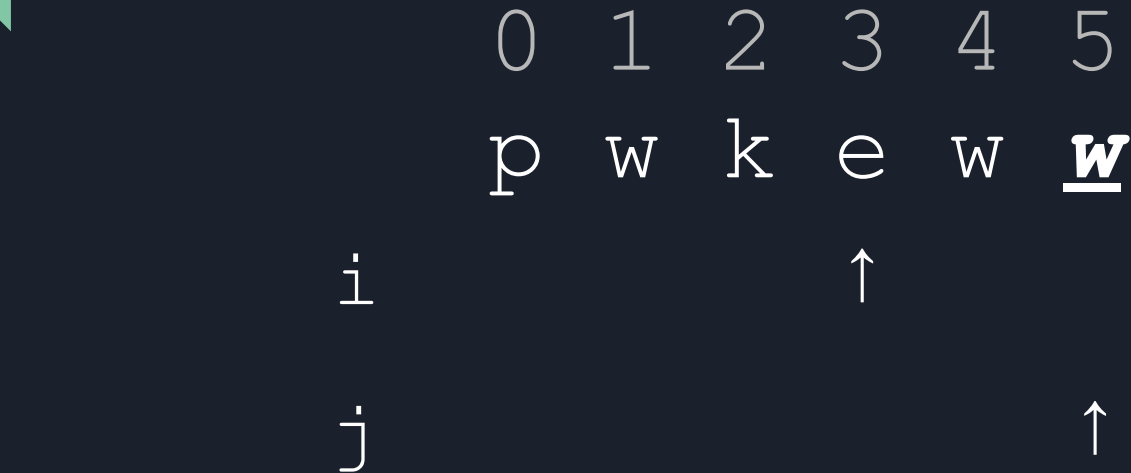
Checking “pwkeww”



L = 3
Max = 4

[i, j) = [2, 5) set = {k, e, w}

Checking “pwkeww”



L = 2
Max = 4

[i, j) = [3, 5) set = {~~k,~~ e, w}

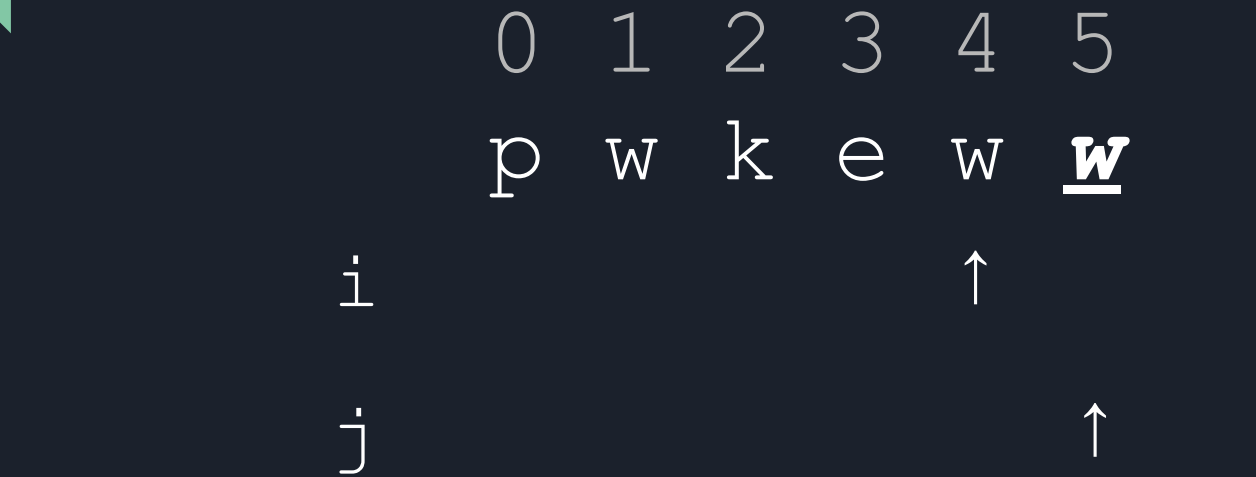
Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	w	***w***
i				↑		
j						↑

L = 2
Max = 4

[i, j) = **[3, 5)** set = **{e, w}**

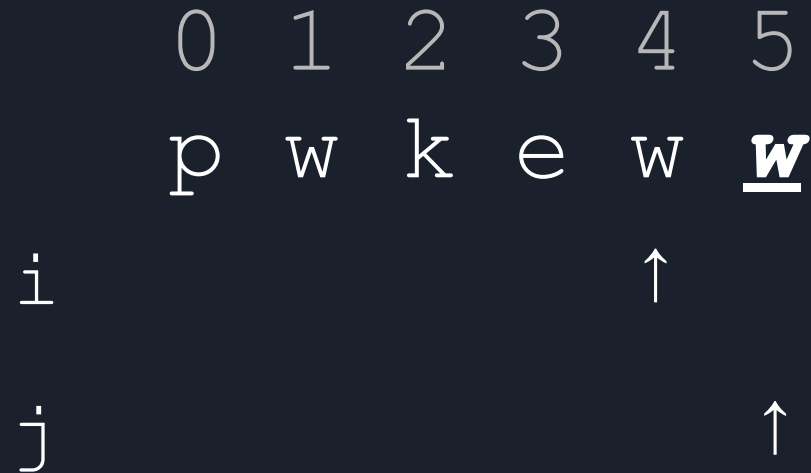
Checking “pwkeww”



L = 1
Max = 4

[i, j) = [4, 5) set = {~~e, w~~}

Checking “pwkeww”



L = 1
Max = 4

[i, j) = **[4, 5)** set = **{w}**

Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	w	***w***
i						↑
j						↑

L = 0
Max = 4

[i, j) = **[5, 5)** set = {~~**w**~~}

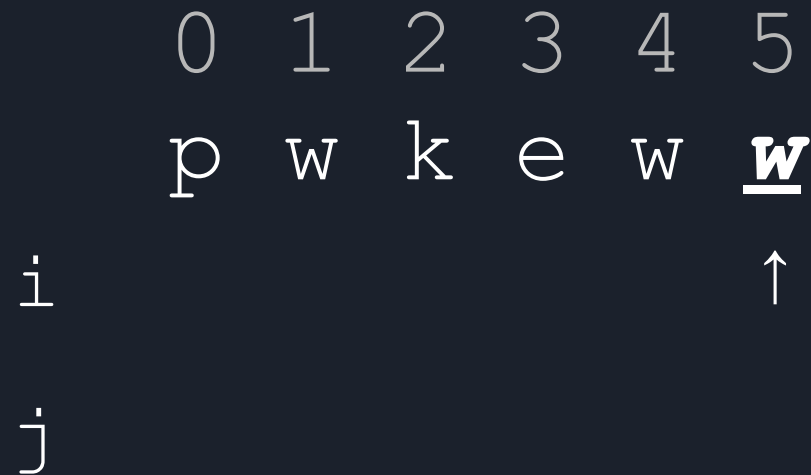
Checking “pwkeww”

	0	1	2	3	4	5
	p	w	k	e	w	***w***
i						↑
j						↑

L = 0
Max = 4

[i, j) = [5, 5) set = {}

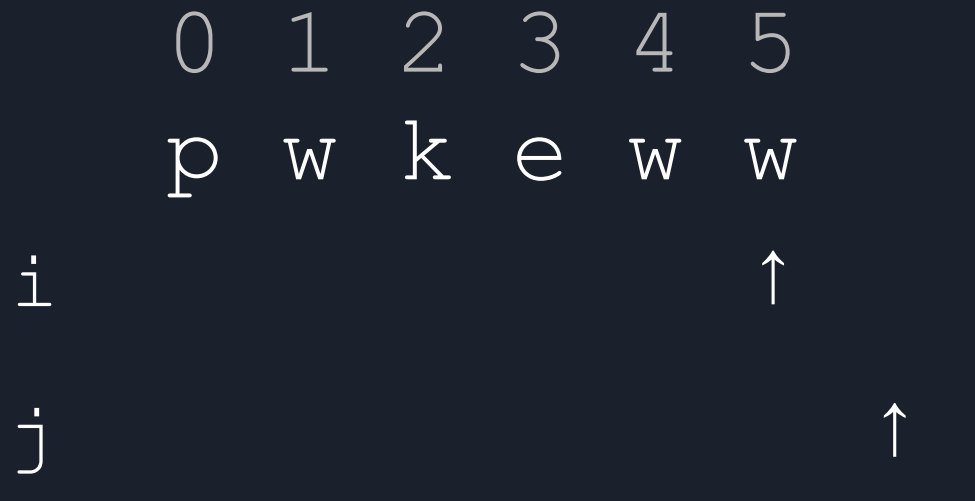
Checking “pwkeww”



L = 1
Max = 4

[i, j) = **[5, 6)** set = **{w}**

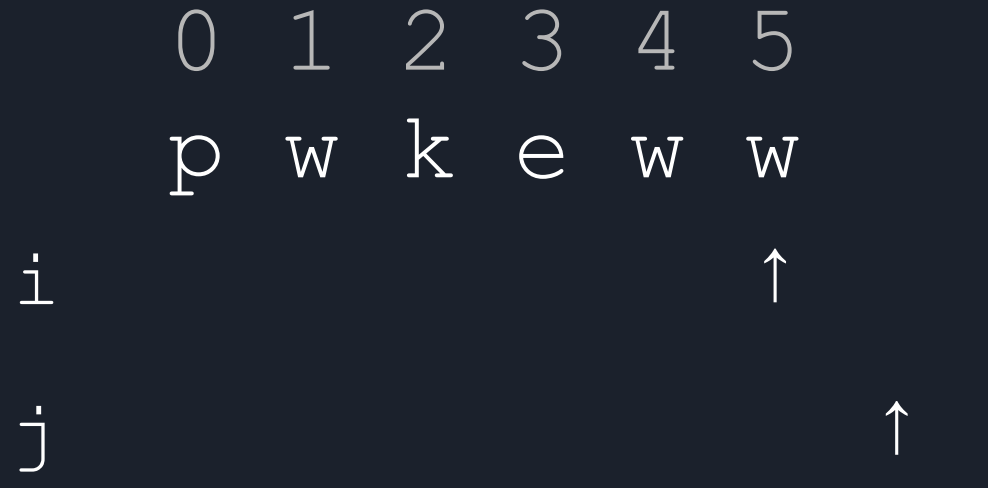
Checking “pwkeww”



L = 1
Max = 4

[i, j) = **[5, 6)** set = **{w}**

Checking “pwkeww”

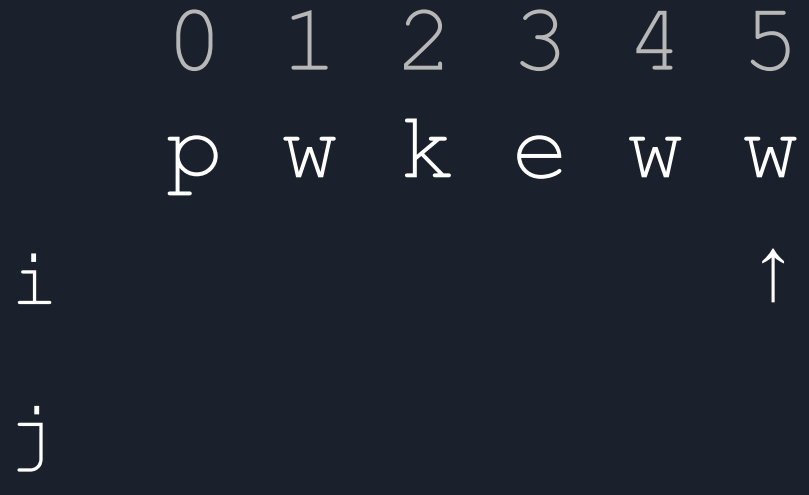


L = 1

Max = 4

[i, j) = [5, 6) set = {w}

Checking “pwkeww”



L = 1

Max = 4

[i, j) = [5, 6) set = {w}

Runtime?

O(n)

Runtime?

O(n)

Why?

Runtime?

O(n)

Why?

Hint: how many times did we access each element?

Runtime?

O(n)

Why?

Hint: how many times did we access each element? **2**

Runtime?

O(n)

Why does it work?

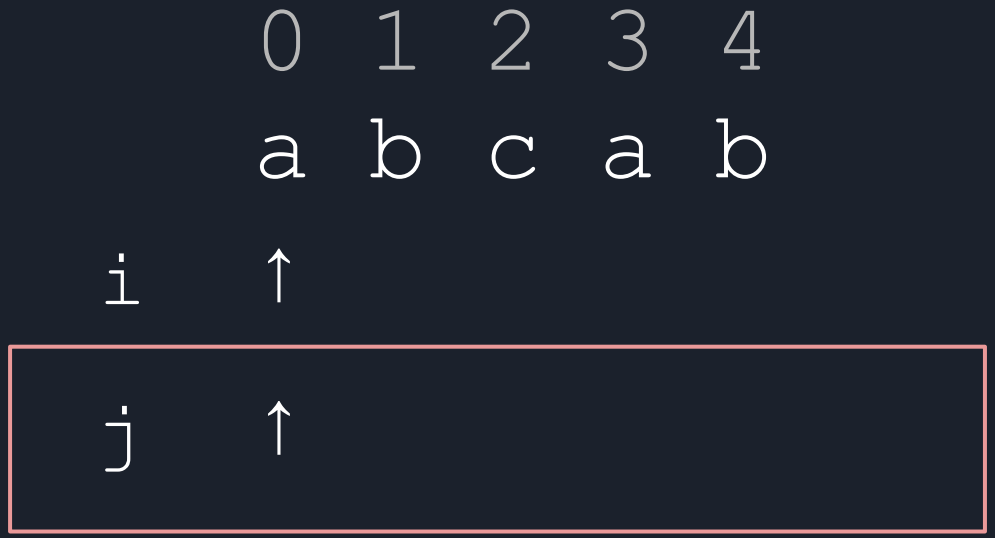
What does pointer `i` do?

Why does it work?

What does pointer `i` do?

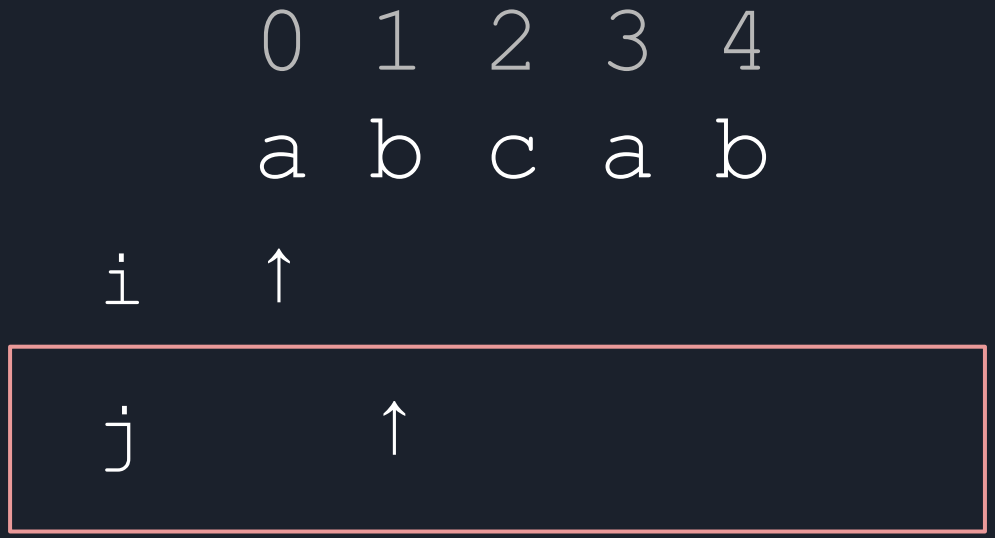
- It indicates the start of a substring that we are checking now

Example: Sliding Window



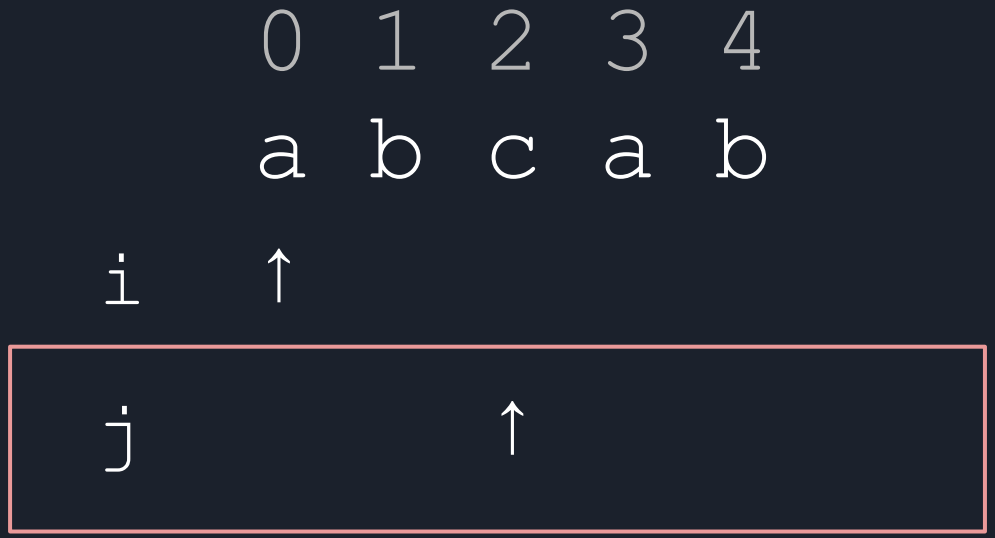
Checks: “a”

Example: Sliding Window



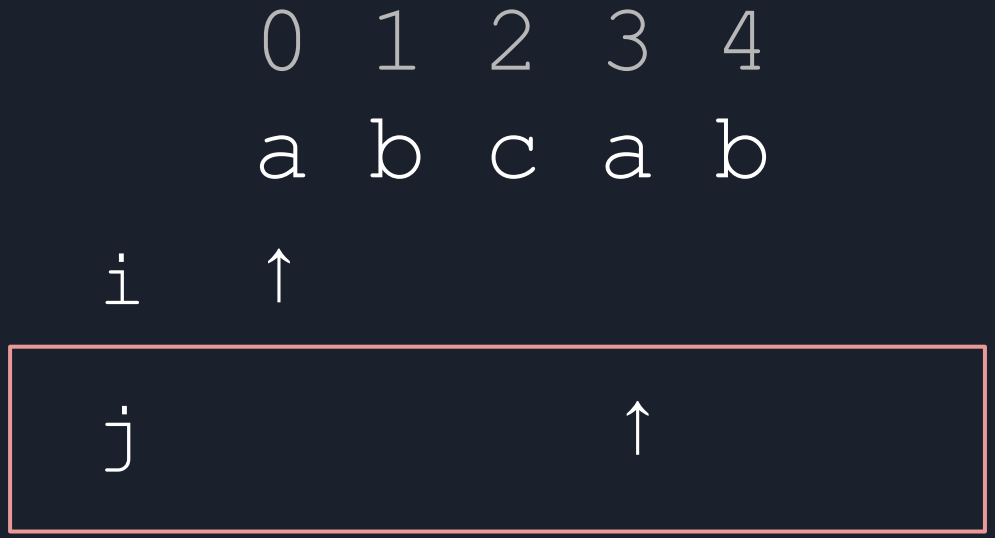
Checks: “ab”

Example: Sliding Window



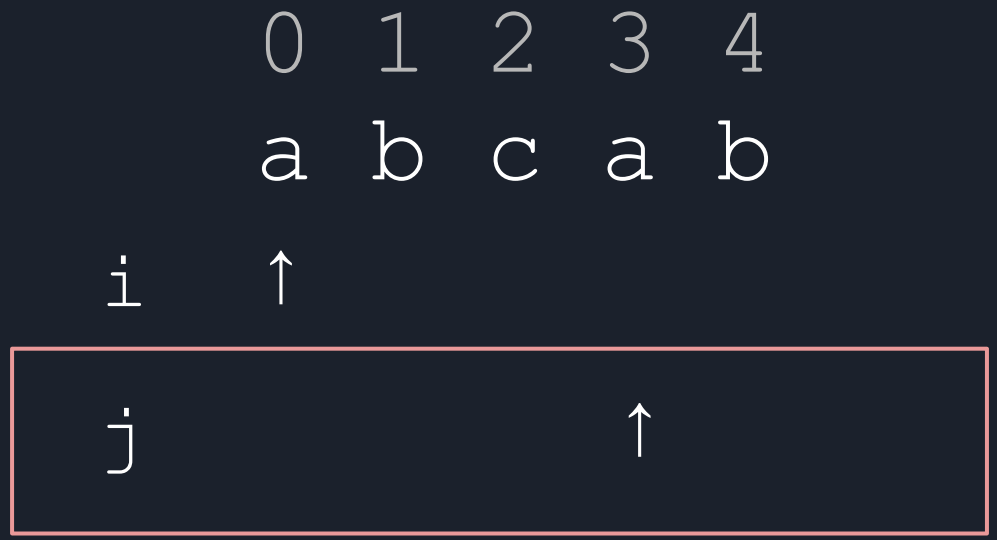
Checks: “abc”

Example: Sliding Window



Checks: “abca”

Example: Sliding Window

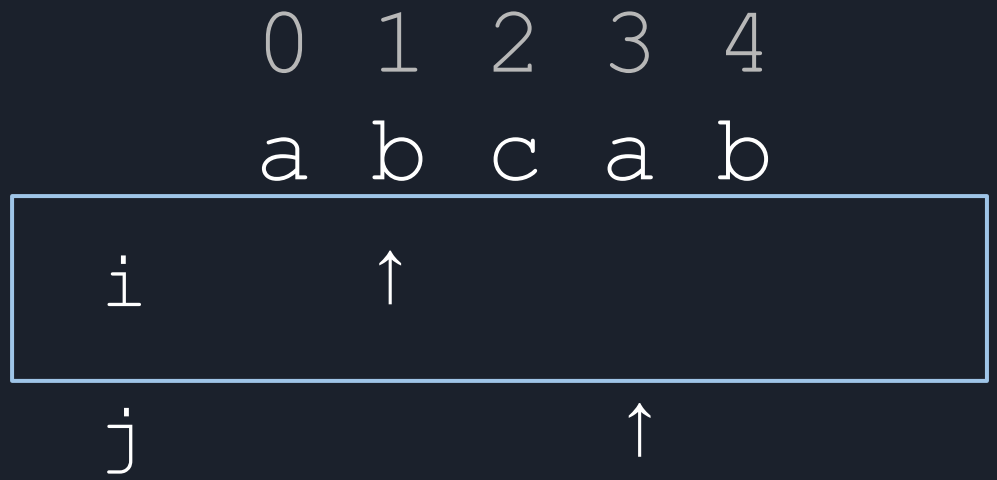


Invalid

Checks: “abca”

Example: Sliding Window

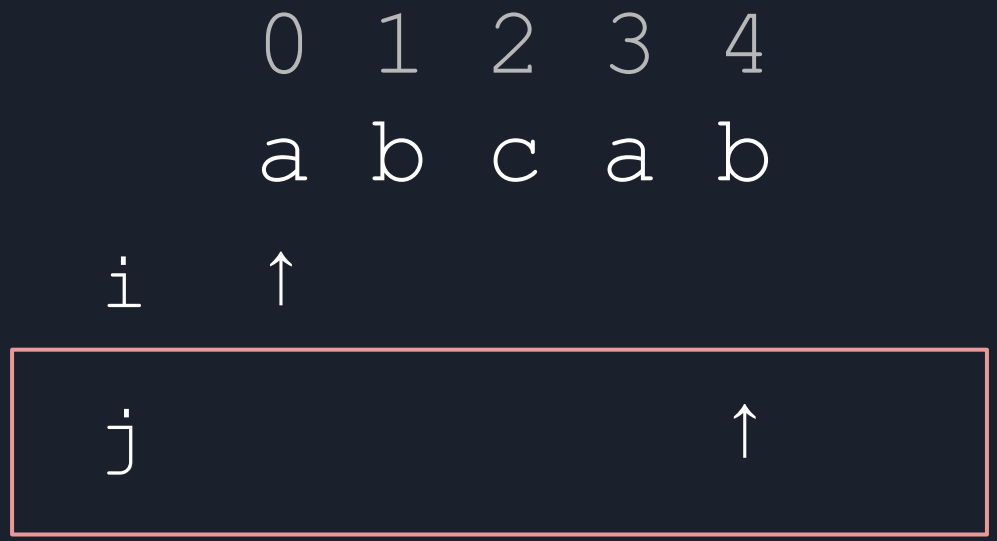
When `i` moves, we finished checking all substrings that begin with `s[0]`



Checks: “bca”

Example: Sliding Window

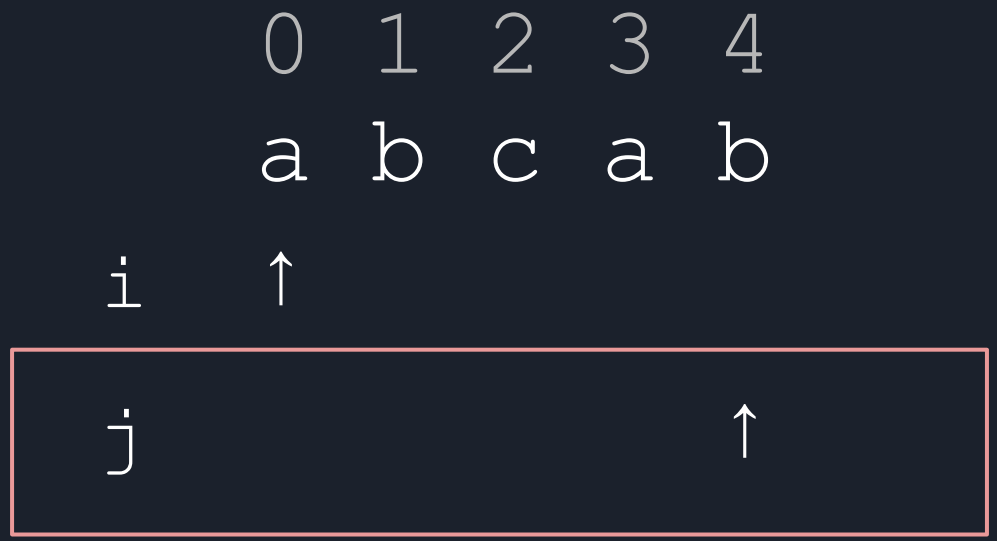
Did we need to check "abcab"?



Example: Sliding Window

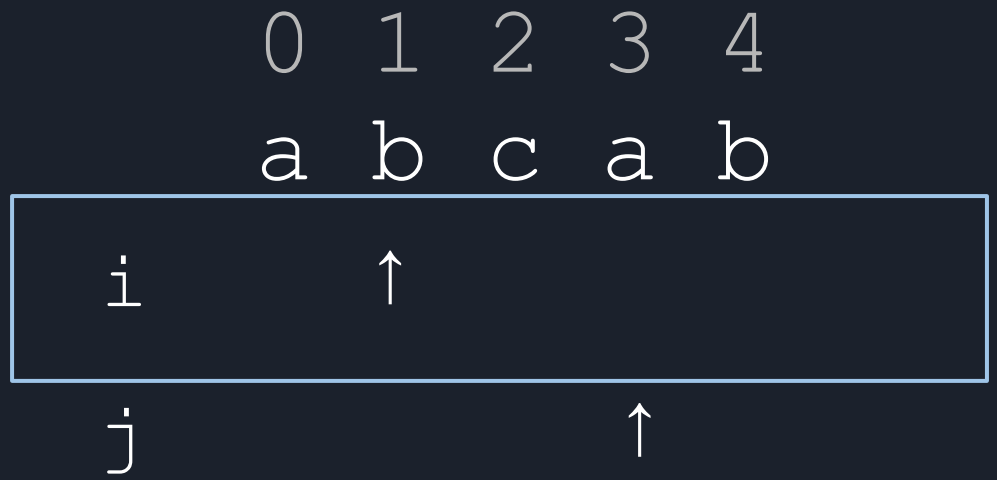
Did we need to check "abcab"?

- ***No***



Example: Sliding Window

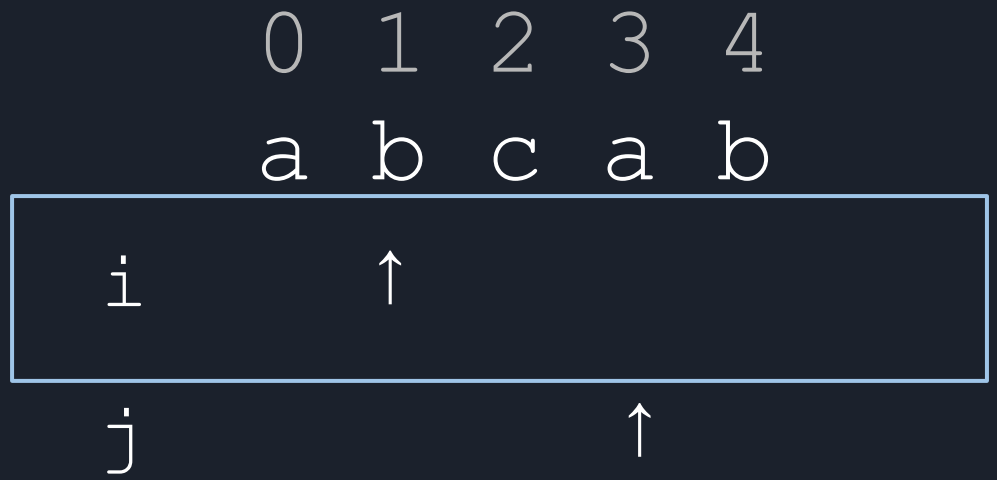
Now we start checking substrings that begin with `s[1]`



Checks: “bca”

Example: Sliding Window

Why didn't we check "b" and "bc"?

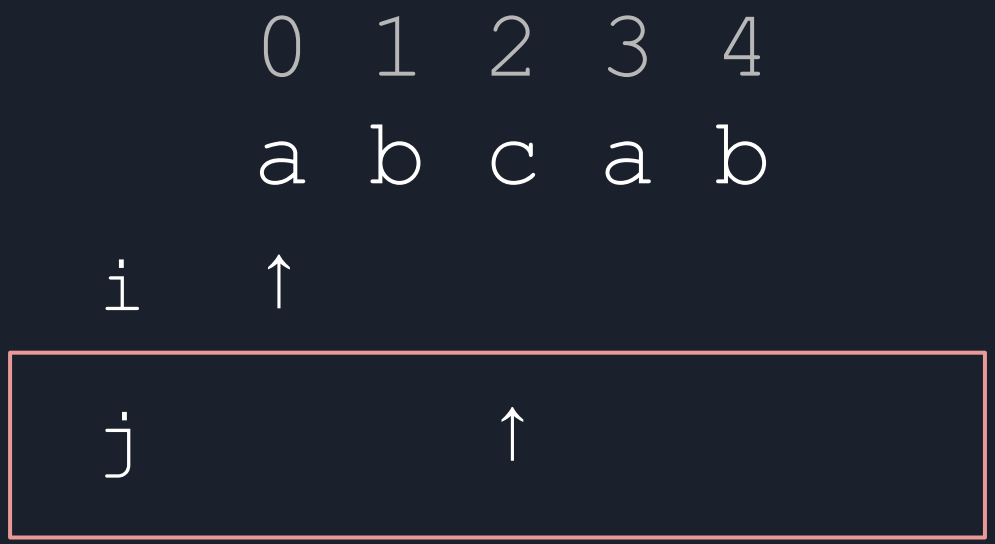


Checks: "bca"

Example: Sliding Window

Why didn't we check "b" and "bc"?

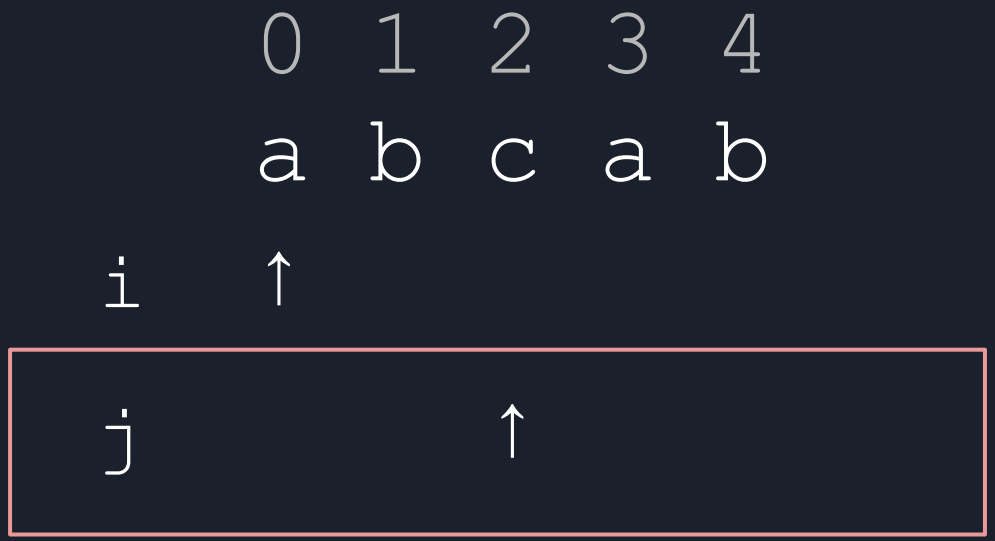
Recall...



Checks: "abc"

Example: Sliding Window

Why didn't we check "b" and "bc"?



Recall...

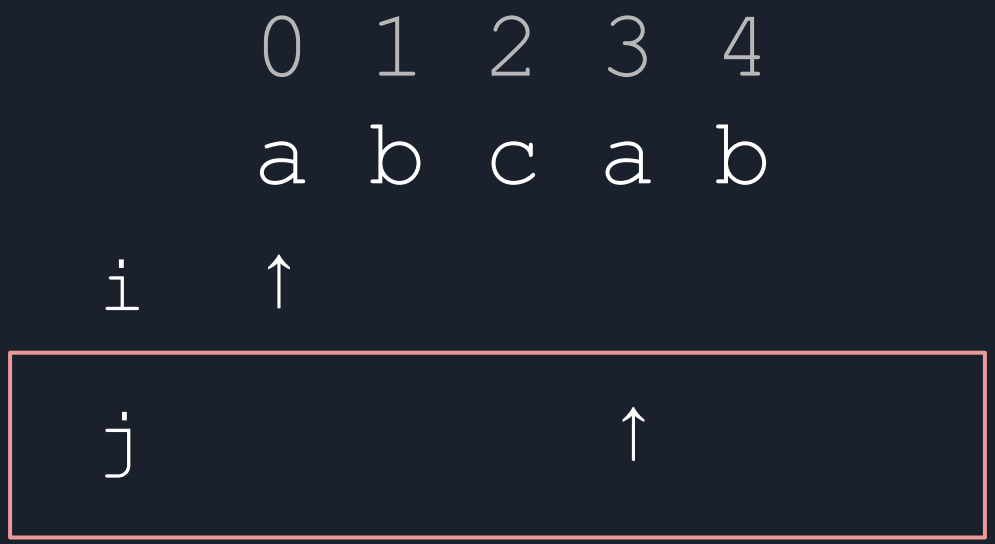
Valid

Checks: "abc"

Example: Sliding Window

Why didn't we check "b" and "bc"?

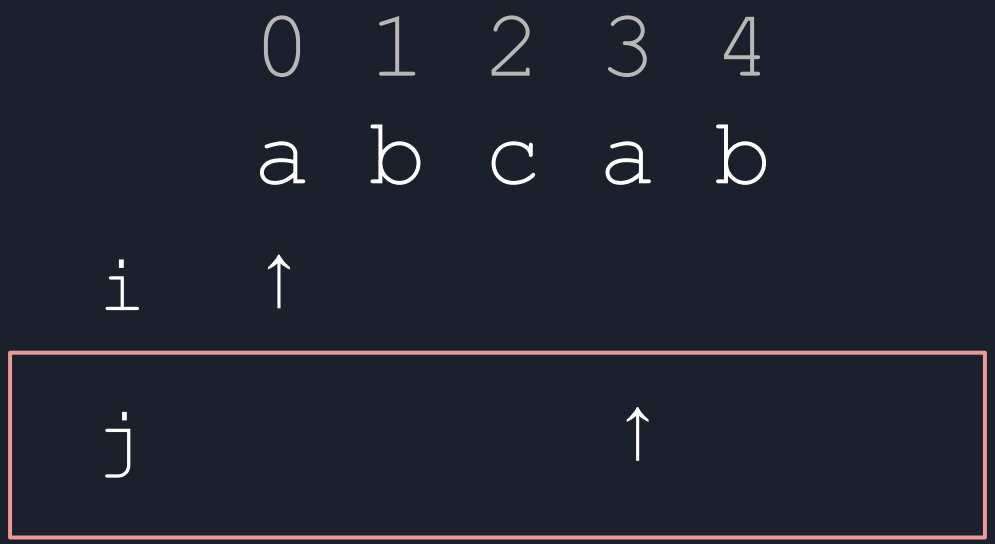
Recall...



Checks: "abca"

Example: Sliding Window

Why didn't we check "b" and "bc"?



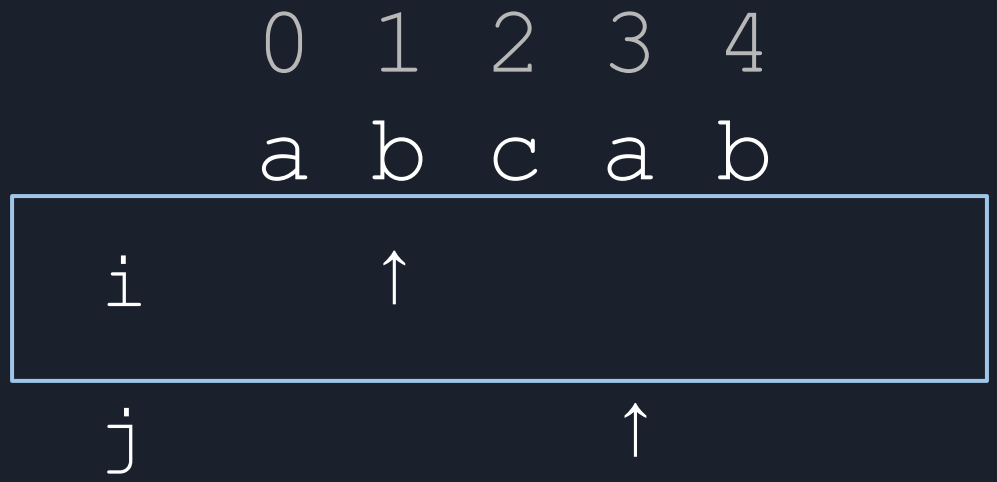
Recall...

Invalid

Checks: "abca"

Example: Sliding Window

Why didn't we check "b" and "bc"?



Recall...

Checks: "bca"

Example: Sliding Window

- We knew there was a valid substring with length 3 from previous moves