KMP Algorithm

Introduction

- Used for searching a pattern in a string.

Ex: 1. String: “this is a **boring** presentation.”
Pattern: “**boring**”
Output: Pattern found at index 10.

2. String: “A**AABA**ABBBAB**AABA**”
Pattern: “**AABA**”
Output: Pattern found at index 1.
Pattern found at index 11.

Introduction

- In naïve approach, the worst case time complexity is O(nm). Using KMP algorithm, the time complexity can be improved to O(n) in the worst case.
- Unlike naïve approach, KMP needs O(m) auxiliary space.

The LPS array

- The key to KMP is the Longest Suffix Prefix(LPS) array or partial match array.
- We create a LPS array with size equal to the size of the pattern.
- For each index ‘i’ in the LPS array store the **length of the longest proper suffix which is also the proper prefix for a substring from zero to ‘i’.**

The LPS array

- Proper Prefixes of “aababaab” : “a” , “aa” , “aab” , “aaba” , “aabab” , “aababa” and “aababaa”.
- Proper Suffixes of “aababaab” : “b” , “ab” , “aab” , “baab” , “abaab”, “babaab” and “ababaab”.
- There is only one common suffix and prefix : “aab”. Length is 3.

Example of creating LPS array:

Pattern: “abcaby”.

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern: “aabaabaaa”.

Index	0	1	2	3	4	5	6	7	8
Pattern	a	a	b	a	a	b	a	a	a
LPS Array	0	1	0	1	2	3	4	5	2

Code for creating LPS Array:

```
int len=0, i=1, lps[0]=0;
while(i<m)  // ’m’ is the pattern length
{   if(pat[i]==pat[len]) {
            lps[i] = ++len;
            i++;
    }
    else {
            if(len!=0)  len=lps[len-1];
            else lps[i++] = 0;
     }
}
```

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0																	

lps[0] initialized to zero

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0																

len=0, i=1

pat[i] != pat[len]

Since len=0, lps[i] = 0
i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1															

len=0, i=2

pat[i] == pat[len]

lps[i] = len + 1
i++, len++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2														

len=1, i=3

pat[i] == pat[len]

lps[i] = len + 1
i++, len++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3													

len=2, i=4

pat[i] == pat[len]

lps[i] = len + 1
i++, len++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3													

len=3, i=5

pat[i] != pat[len]

len=lps[len-1]

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3													

len=1, i=5

pat[i] != pat[len]

len=lps[len-1]

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0												

len=0, i=5

pat[i] != pat[len]

Since len=0, lps[i]=0
i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1											

len=0, i=6

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2										

len=1, i=7

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3									

len=2, i=8

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4								

len=3, i=9

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5							

len=4, i=10

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6						

len=5, i=11

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7					

len=6, i=12

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8				

len=7, i=13

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9			

len=8, i=14

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10		

len=9, i=15

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	

len=10, i=16

pat[i] == pat[len]

lps[i]=len+1
len++, i++

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	

len=11, i=17

pat[i] != pat[len]

len=lps[len-1]

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	

len=5, i=17

pat[i] != pat[len]

len=lps[len-1]

LPS Array. Why it works?

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pattern	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
LPS	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	4

len=3, i=17

pat[i] == pat[len]

lps[i]=len+1
len++, i++

Algorithm of KMP Search:

```
int i=0, j=0, start=0;
while(i<n) {
    if(pat[j]==string[i]) {
            j++;  i++;
     }
     if(j==m) {
            printf("Pattern found at %d" , start);
            j=lps[j-1];
      }
      else if(pat[j] != string[i]) {
            if(j!=0) {     j=lps[j-1];    start=i-lps[j-1];   }
            else {     i++;      start++;    }
       }
}
```

Pattern Search:

start

i

↓

String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

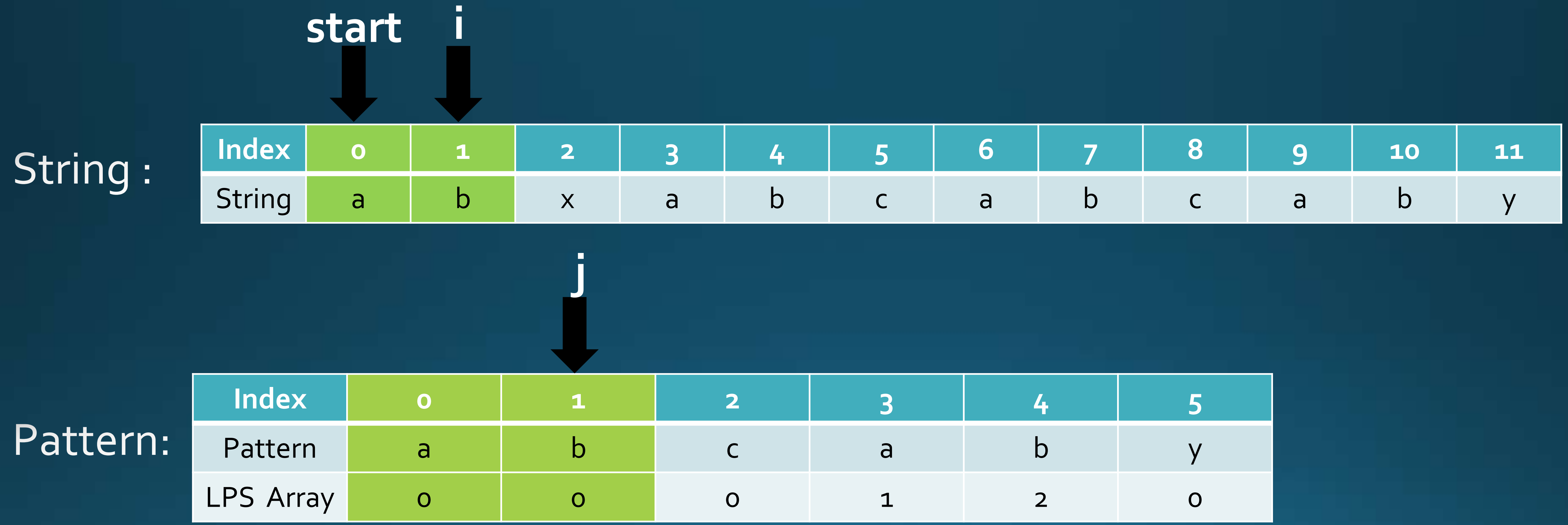
j

↓

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern Search:



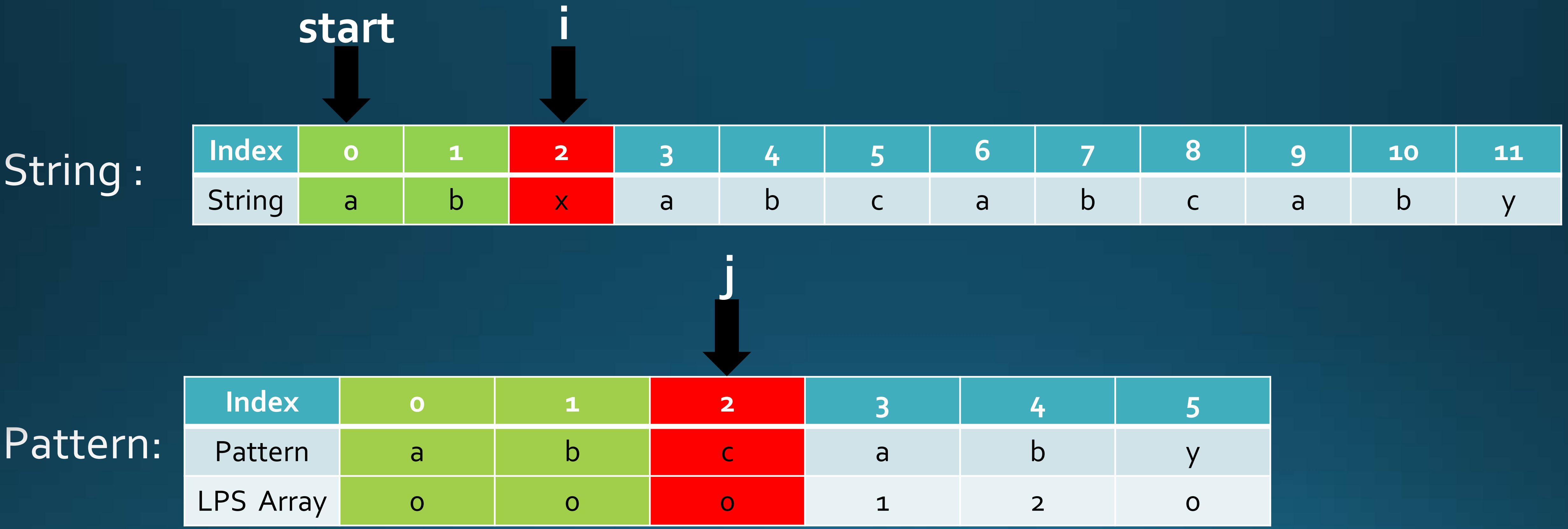
String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern Search:



String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Now, since Pattern[j] doesn’t match with String[i], **j=LPS[j-1], start=i-LPS[j-1].**

Pattern Search:

start

i

String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

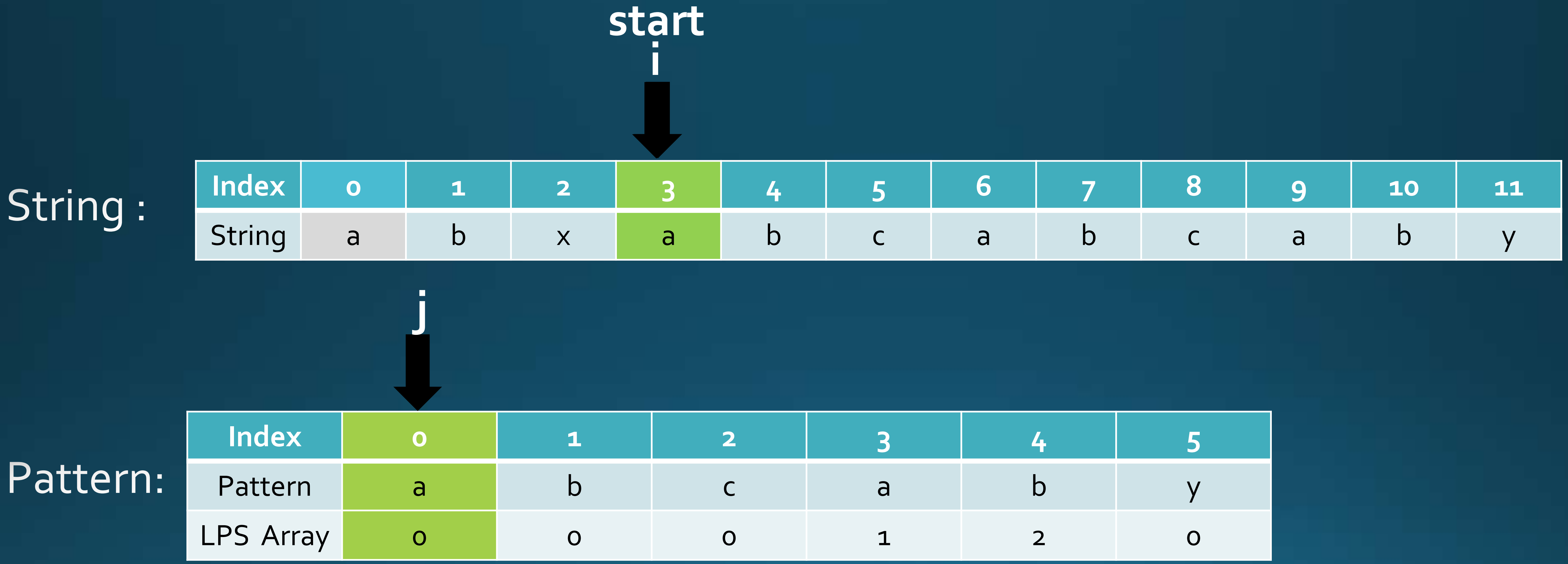
j

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern[j] is still not equal to String[i]. And since j=0, we increment ‘i’ and ‘start’ this time.

Pattern Search:



start

i

String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

j

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern Search:

start

i

String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

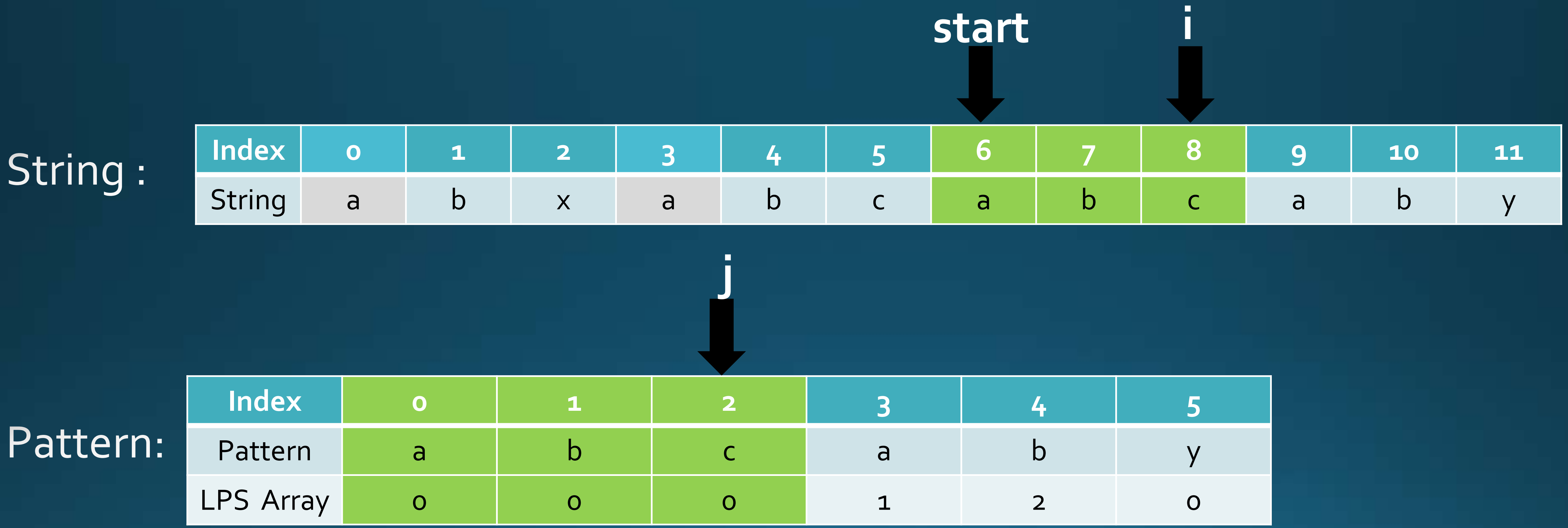
j

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Again String[i]!=Pattern[j]. Therefore j=LPS[j-1]. start=i-LPS[j-1].

Pattern Search:



String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

Pattern Search:

start i

String :

Index	0	1	2	3	4	5	6	7	8	9	10	11
String	a	b	x	a	b	c	a	b	c	a	b	y

j

Pattern:

Index	0	1	2	3	4	5
Pattern	a	b	c	a	b	y
LPS Array	0	0	0	1	2	0

We reached the end of the pattern. Hence the pattern is found at index 'start'.

That’s It.

Thank You!!