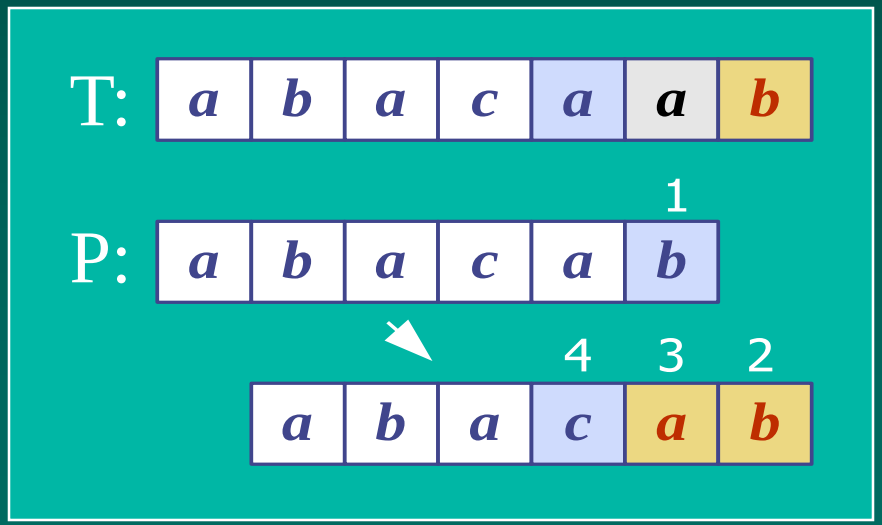
String / Pattern Matching Algorithms



Overview

1. What is Pattern Matching?
2. The Brute Force Algorithm
3. The Boyer-Moore Algorithm
4. The Knuth-Morris-Pratt Algorithm
5. More Information

1. What is Pattern Matching?

- Definition:
  - given a text string T and a pattern string P, find the pattern inside the text
    - T: "the rain in spain stays mainly on the plain"
    - P: "n th"
- Applications:
  - text editors, Web search engines (e.g. Google), image analysis

String Concepts

- Assume S is a string of size m.
- A *substring* S[i .. j] of S is the string fragment between indexes i and j.
- A *prefix* of S is a substring S[0 .. i]
- A *suffix* of S is a substring S[i .. m-1]
  - i is any index between 0 and m-1

Examples

S

- Substring S[1..3] == "ndr"
- All possible prefixes of S:
  - "andrew", "andre", "andr", "and", "an", "a"
- All possible suffixes of S:
  - "andrew", "ndrew", "drew", "rew", "ew", "w"

2. The Brute Force Algorithm

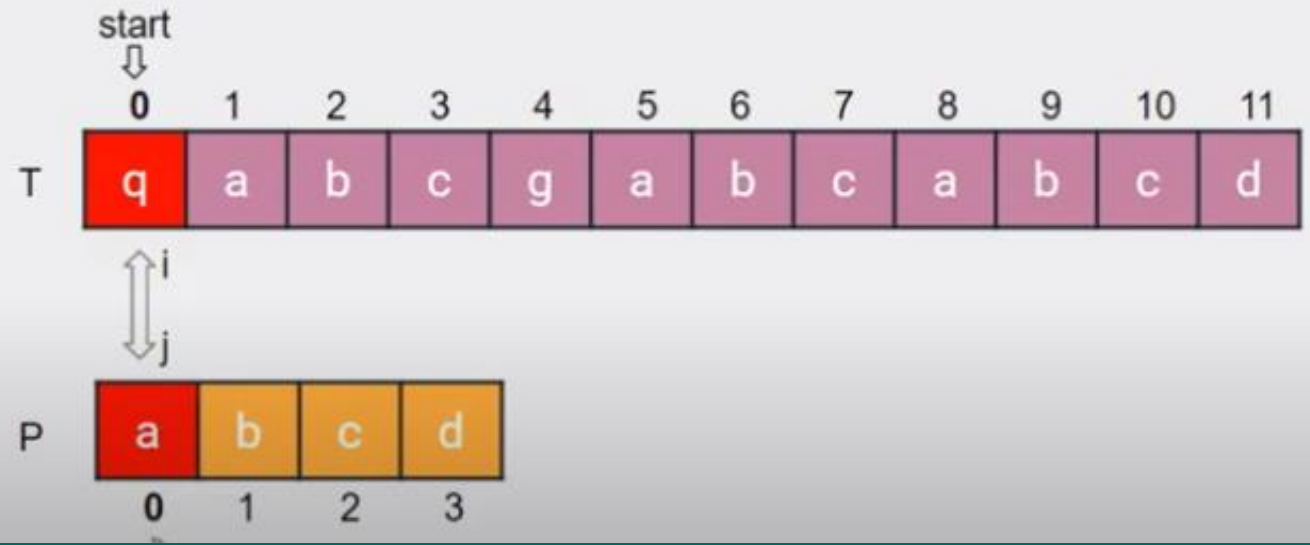
- Check each position in the text T to see if the pattern P starts in that position

P moves 1 char at a time through T

. . . .

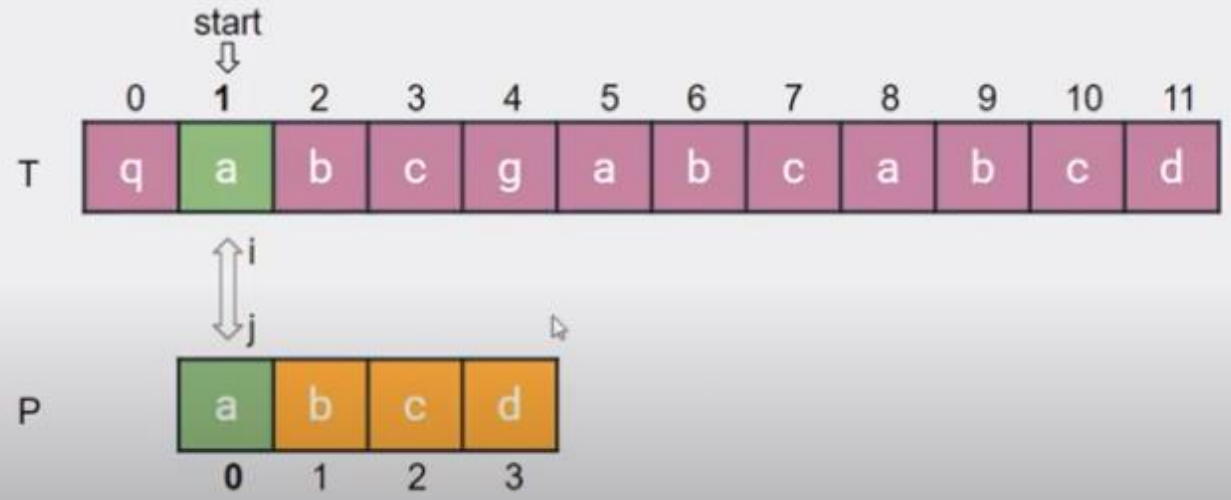
Brute Force Approach (Naive)

Check every possible fit.



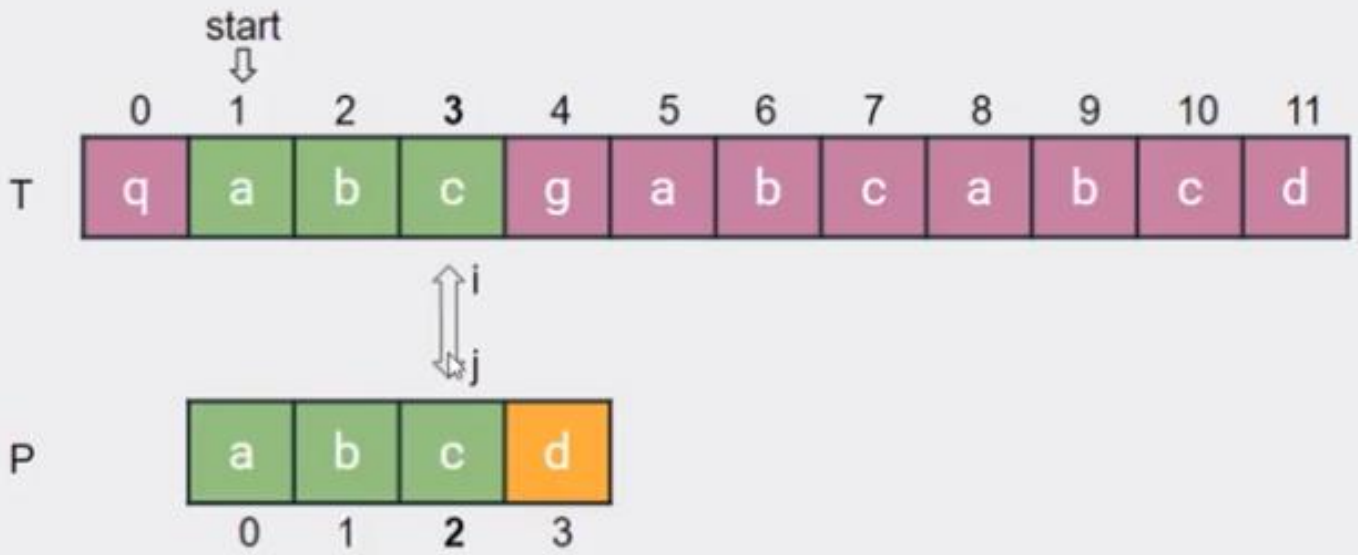
Brute Force Approach (Naive)

Check every possible fit.



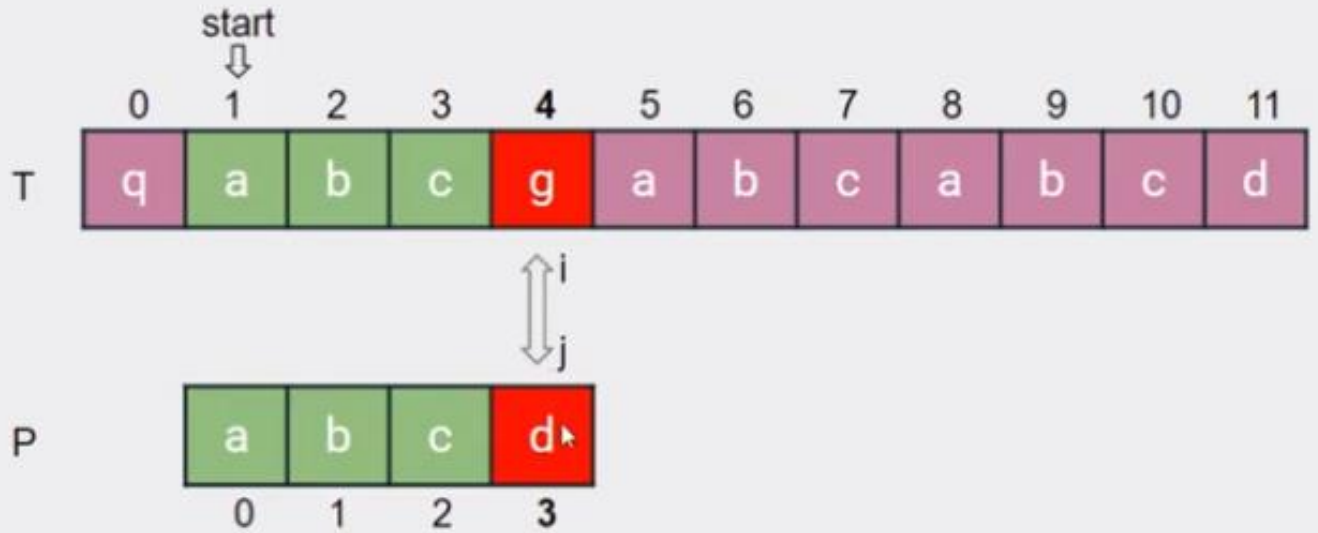
Brute Force Approach (Naive)

Check every possible fit.



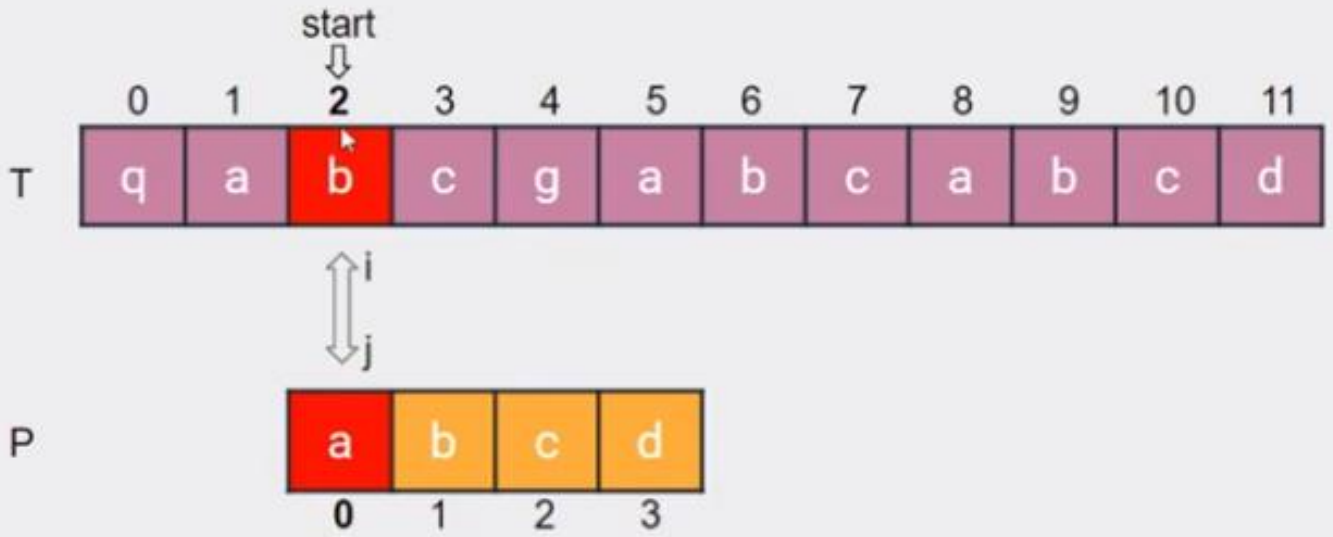
Brute Force Approach (Naive)

Check every possible fit.



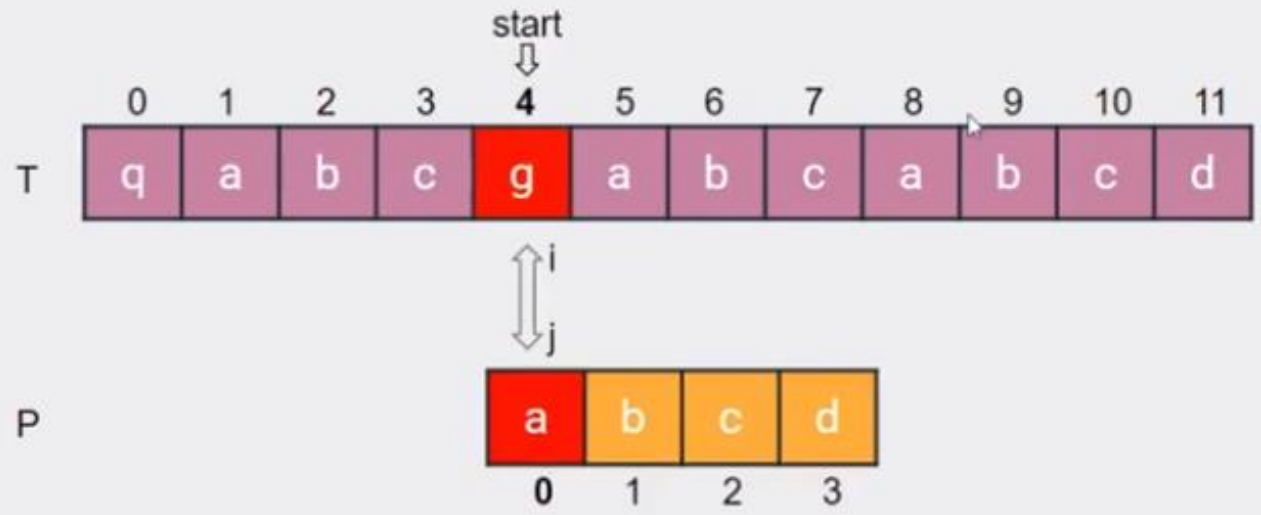
Brute Force Approach (Naive)

Check every possible fit.



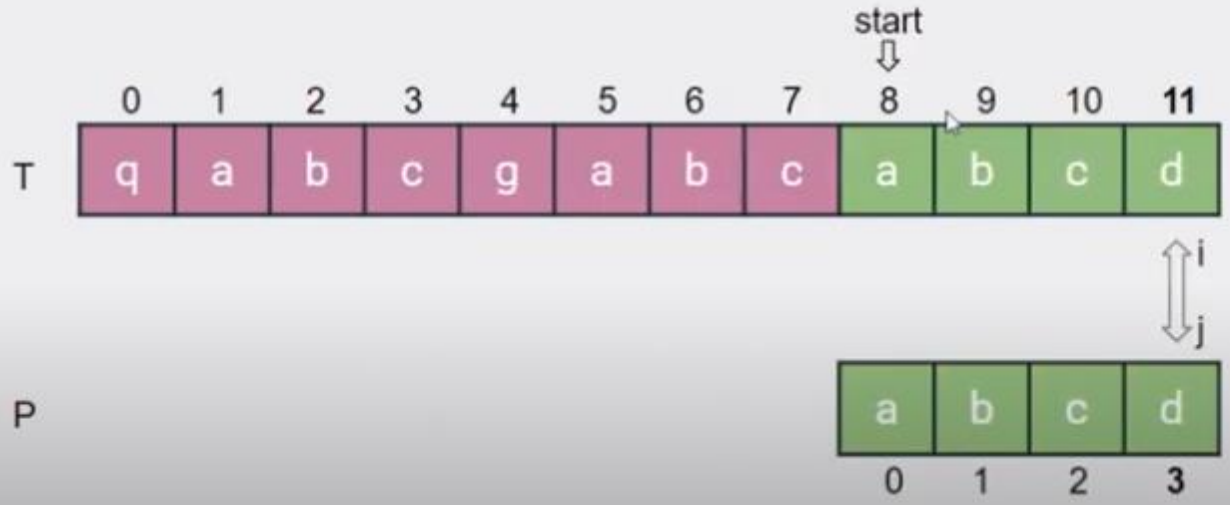
Brute Force Approach (Naive)

Check every possible fit.



Brute Force Approach (Naive)

Check every possible fit.



Return index where pattern starts, or -1

The Naive string-matching algorithm

The naive algorithm finds all valid shifts using a loop that checks the condition $P[1 . .m] = T[s + 1 . . s + m]$ *for each of the* $n - m + 1$ *possible values of* s.

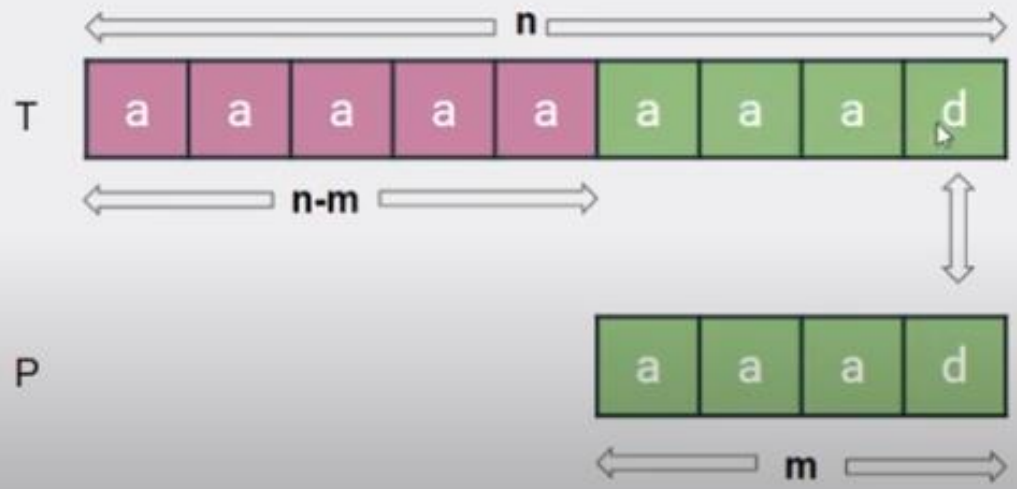
```
NAIVE-STRING-MATCHER(T, P)
1  n ← length[T]
2  m ← length[P]
3  for s ← 0 to n − m
4       do if P[1..m] = T[s + 1..s + m]
5            then print "Pattern occurs with shift" s
```

Procedure NAIVE-STRING-MATCHER takes time $O((n - m + 1)m)$, *and this* bound is tight in the worst case.

Time Complexity

Brute Force Approach (Naive)

Worst Case



Time Complexity

O([n-m+1]*m)

~ O(nm)

when n >>> m

- The brute force algorithm is fast when the alphabet of the text is large
  - e.g. A..Z, a..z, 1..9, etc.
- It is slower when the alphabet is small
  - e.g. 0, 1 (as in binary files, image files, etc.)

continued

- Example of a worst case:
  - T: "aaaaaaaaaaaaaaaaaaaaaaaaaaab"
  - P: "aaab"
- Example of a more average case:
  - T: "a string searching example is standard"
  - P: "store"

4. The KMP Algorithm

- The Knuth-Morris-Pratt (KMP) algorithm looks for the pattern in the text in a *left-to-right* order (like the brute force algorithm).

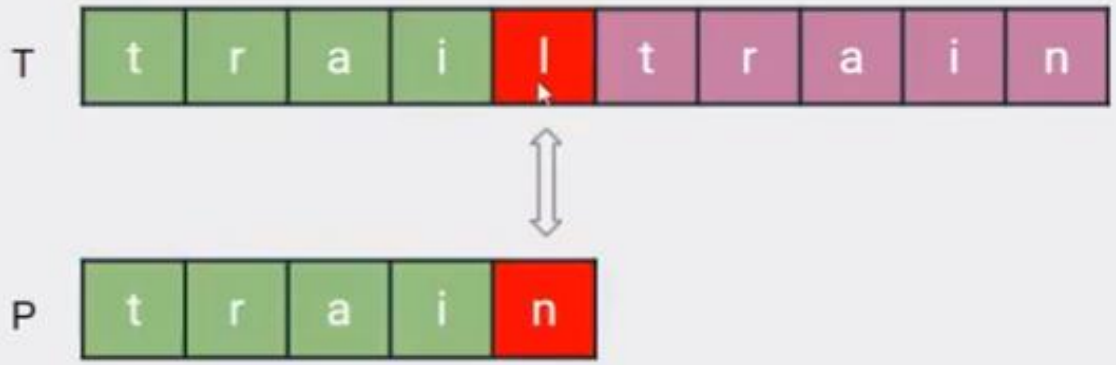
- But it shifts the pattern more intelligently than the brute force algorithm.

continued

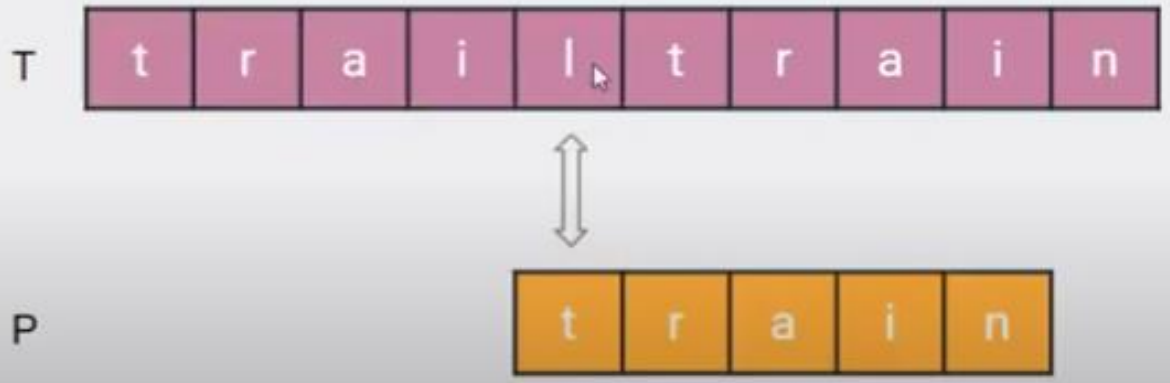
- If a mismatch occurs between the text and pattern P at P[j], what is the *most* we can shift the pattern to avoid wasteful comparisons?

- *Answer*: the largest prefix of P[0 .. j-1] that is a suffix of P[1 .. j-1]

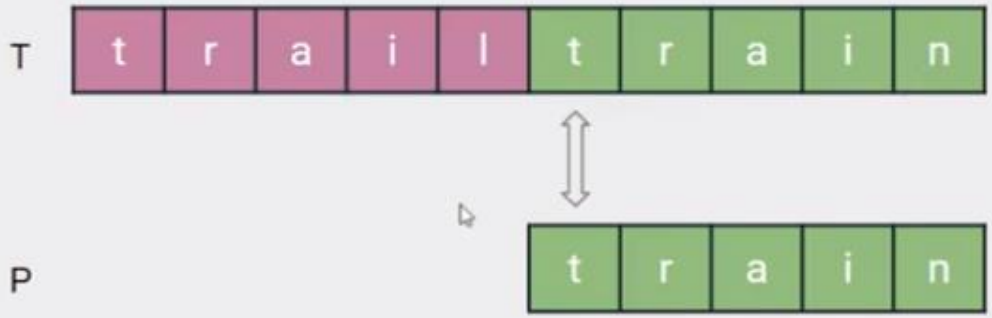
KMP algorithm



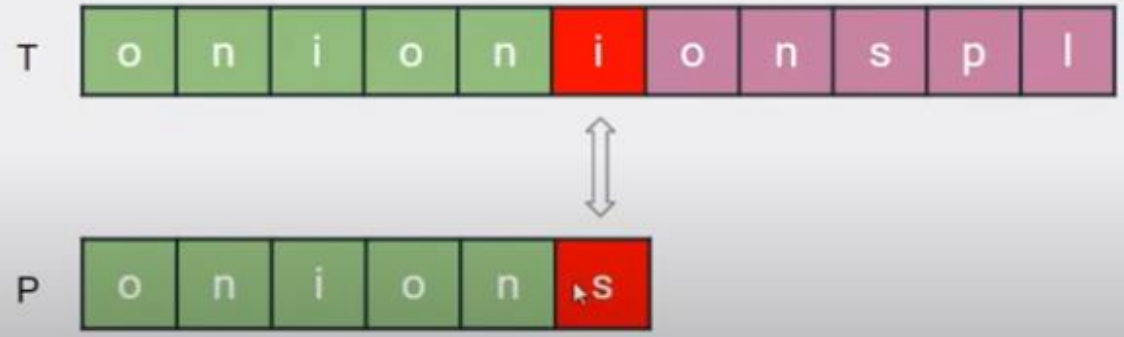
KMP algorithm



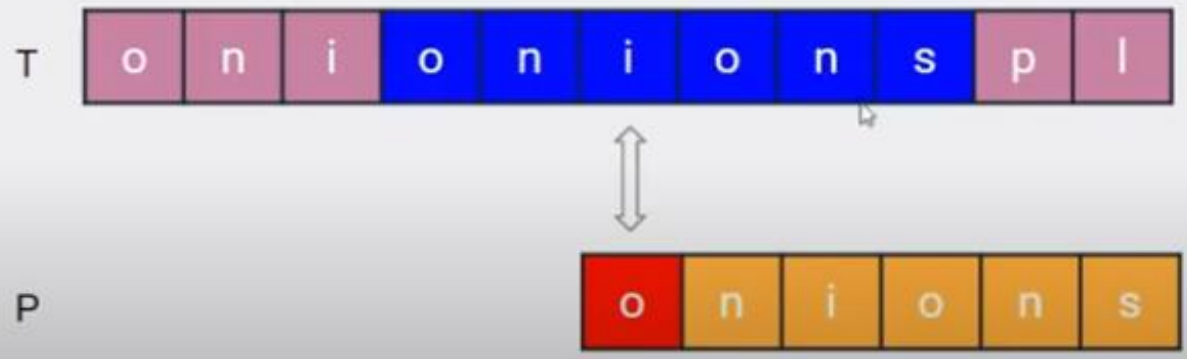
KMP algorithm



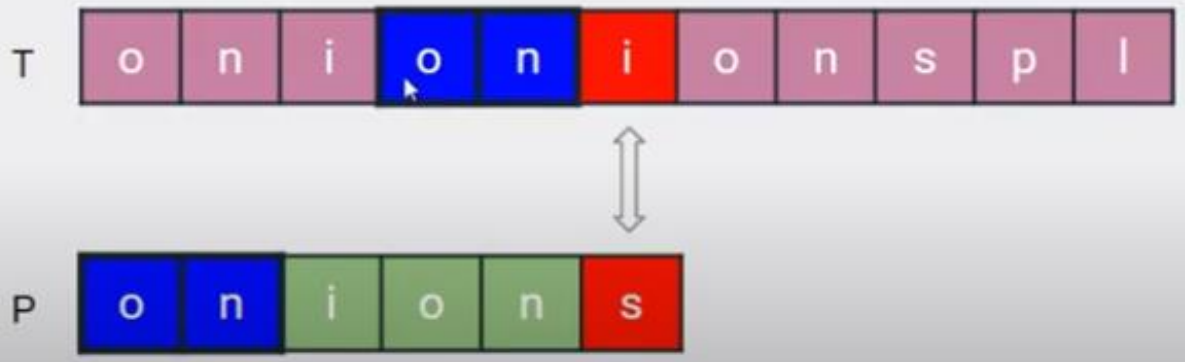
KMP algorithm



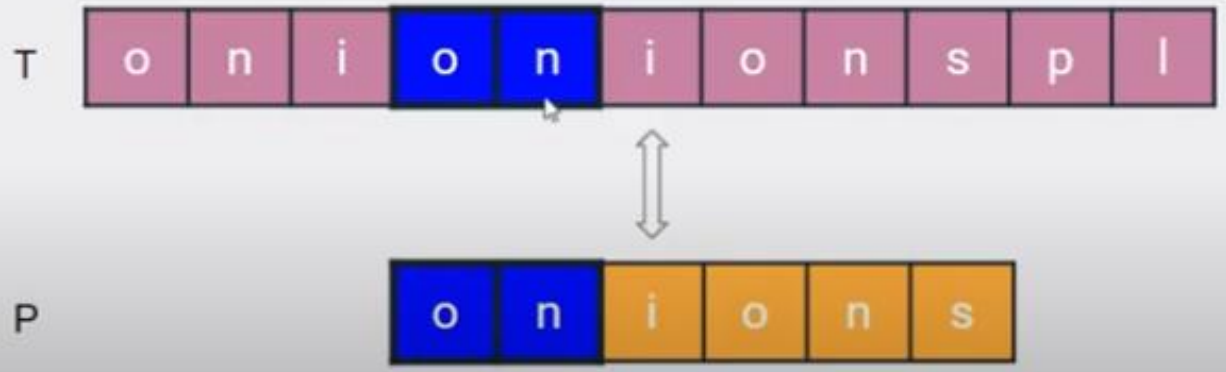
KMP algorithm



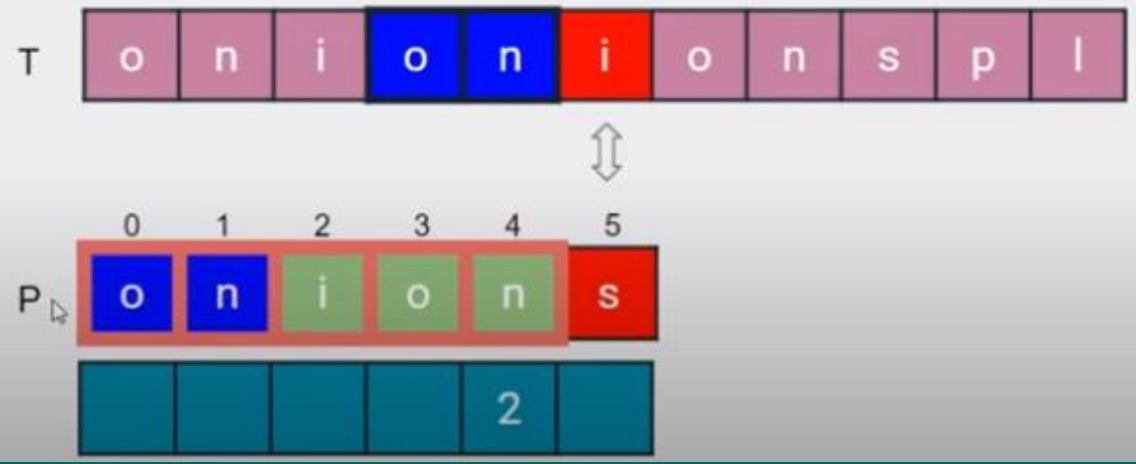
KMP algorithm



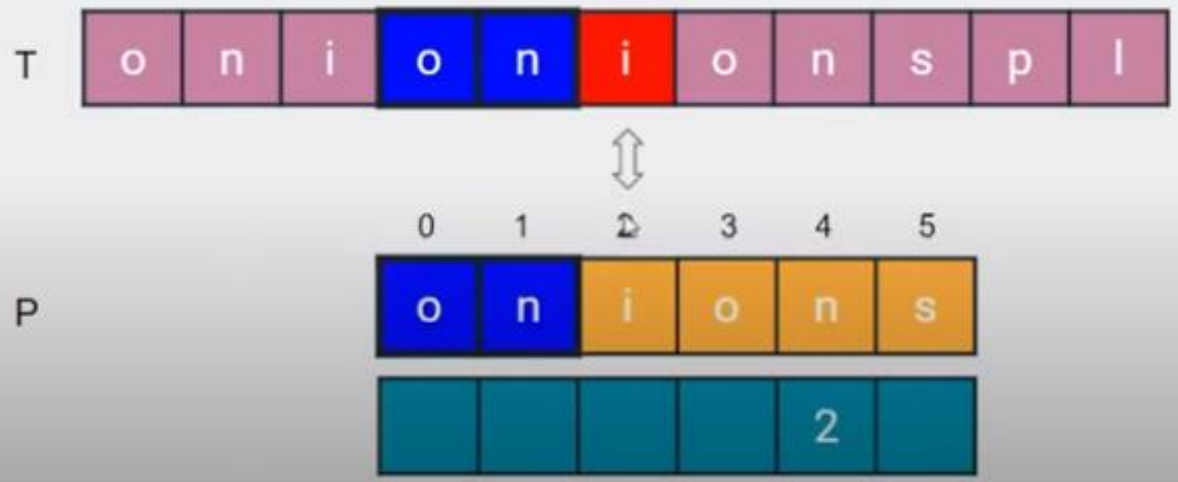
KMP algorithm



KMP algorithm

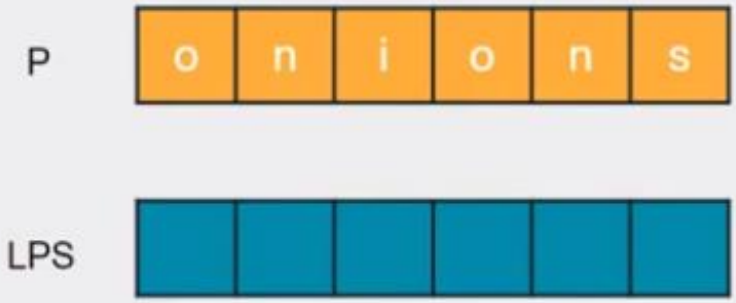


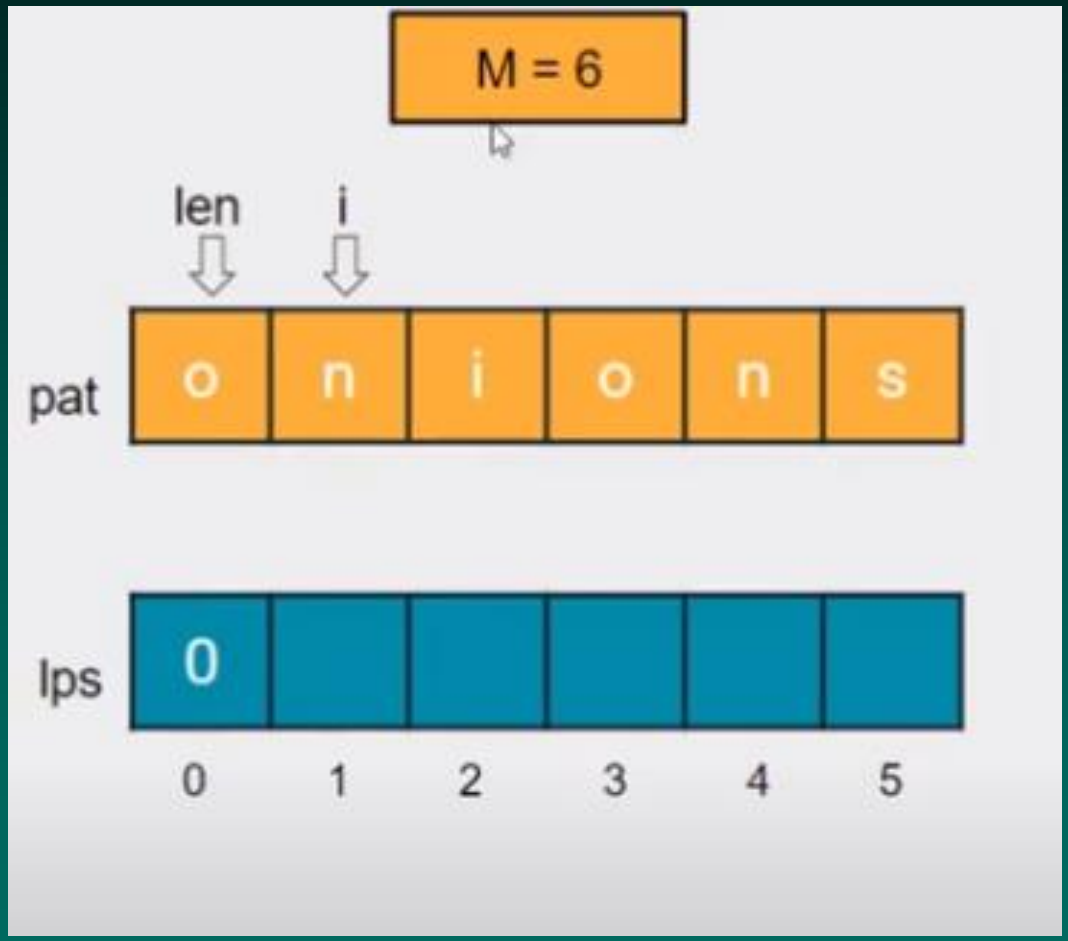
KMP algorithm

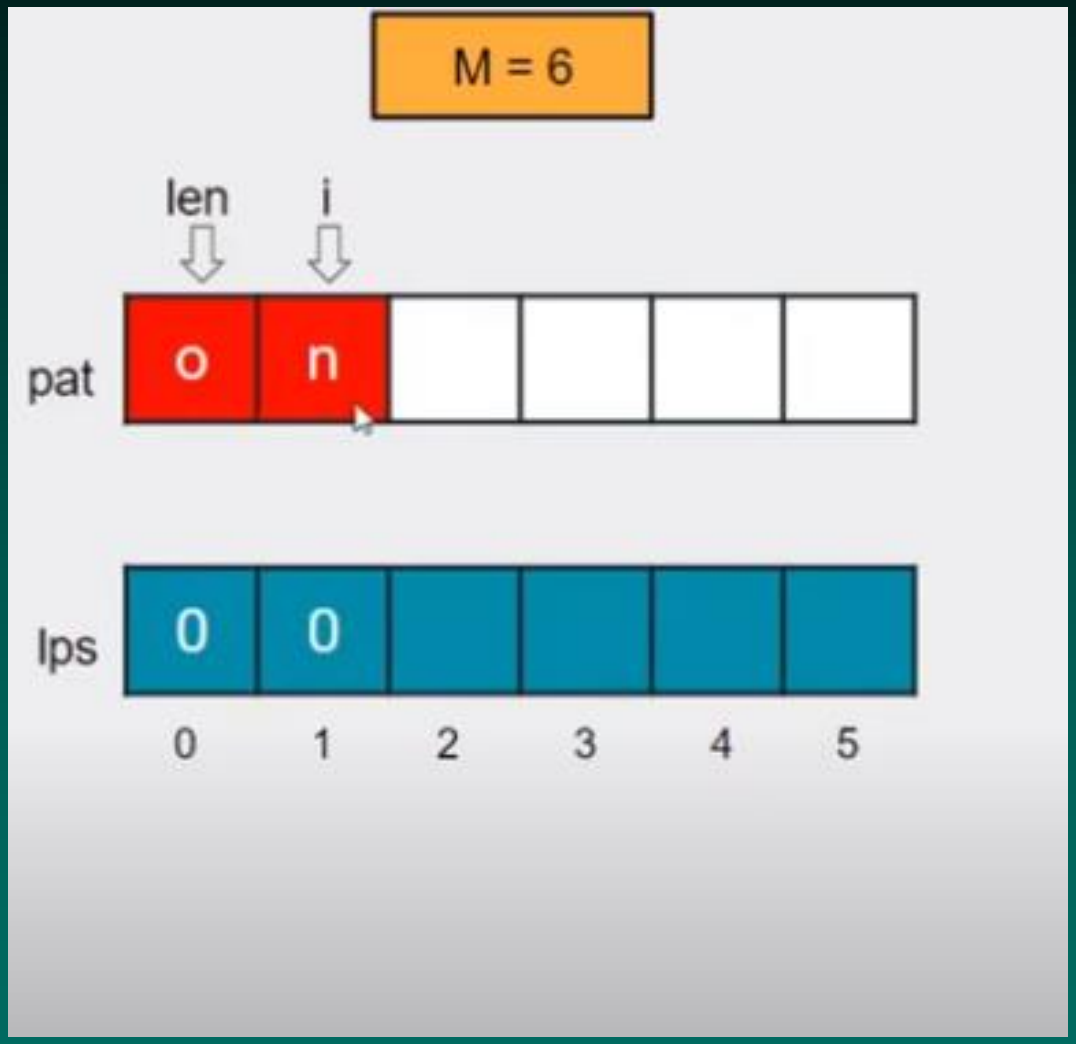


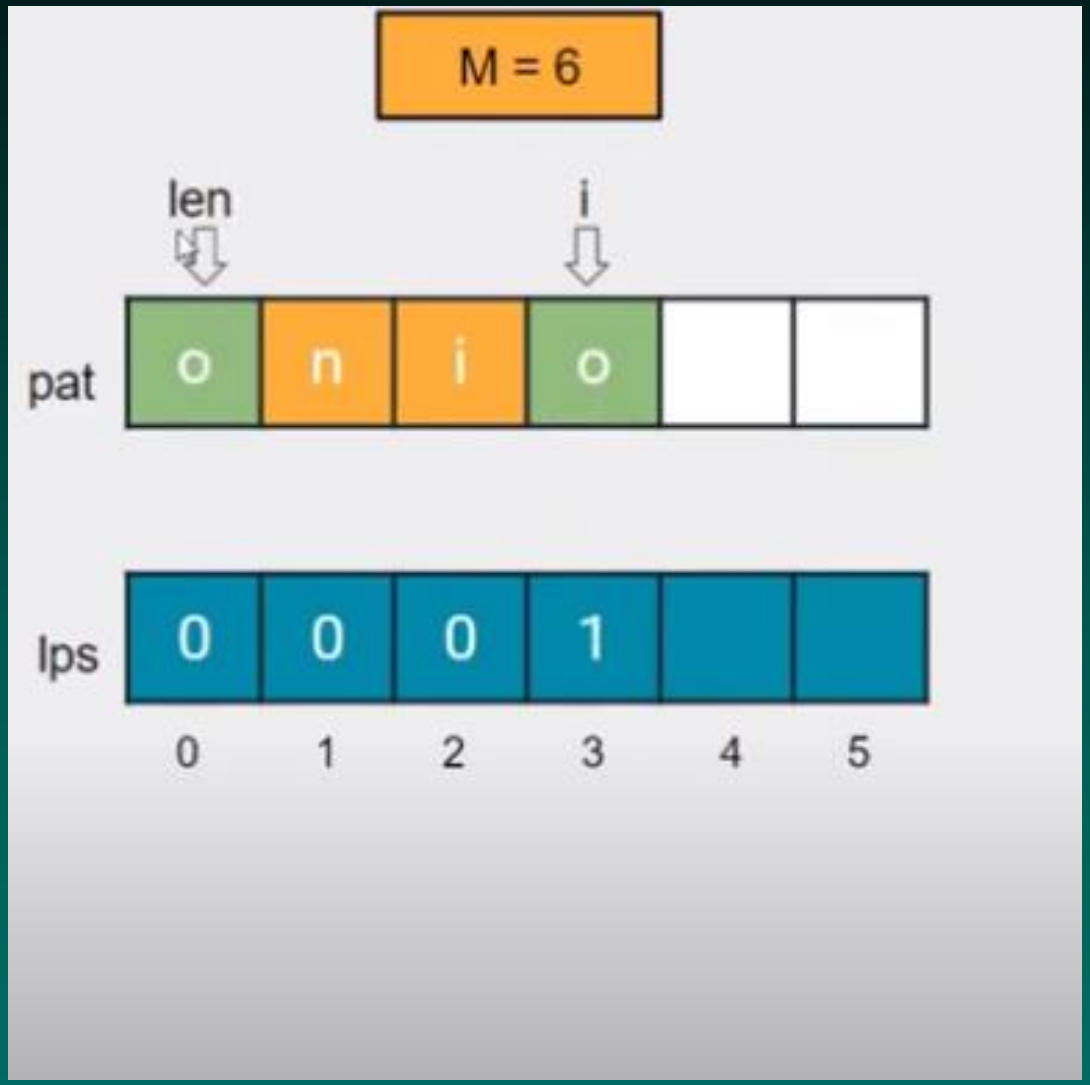
LPS array

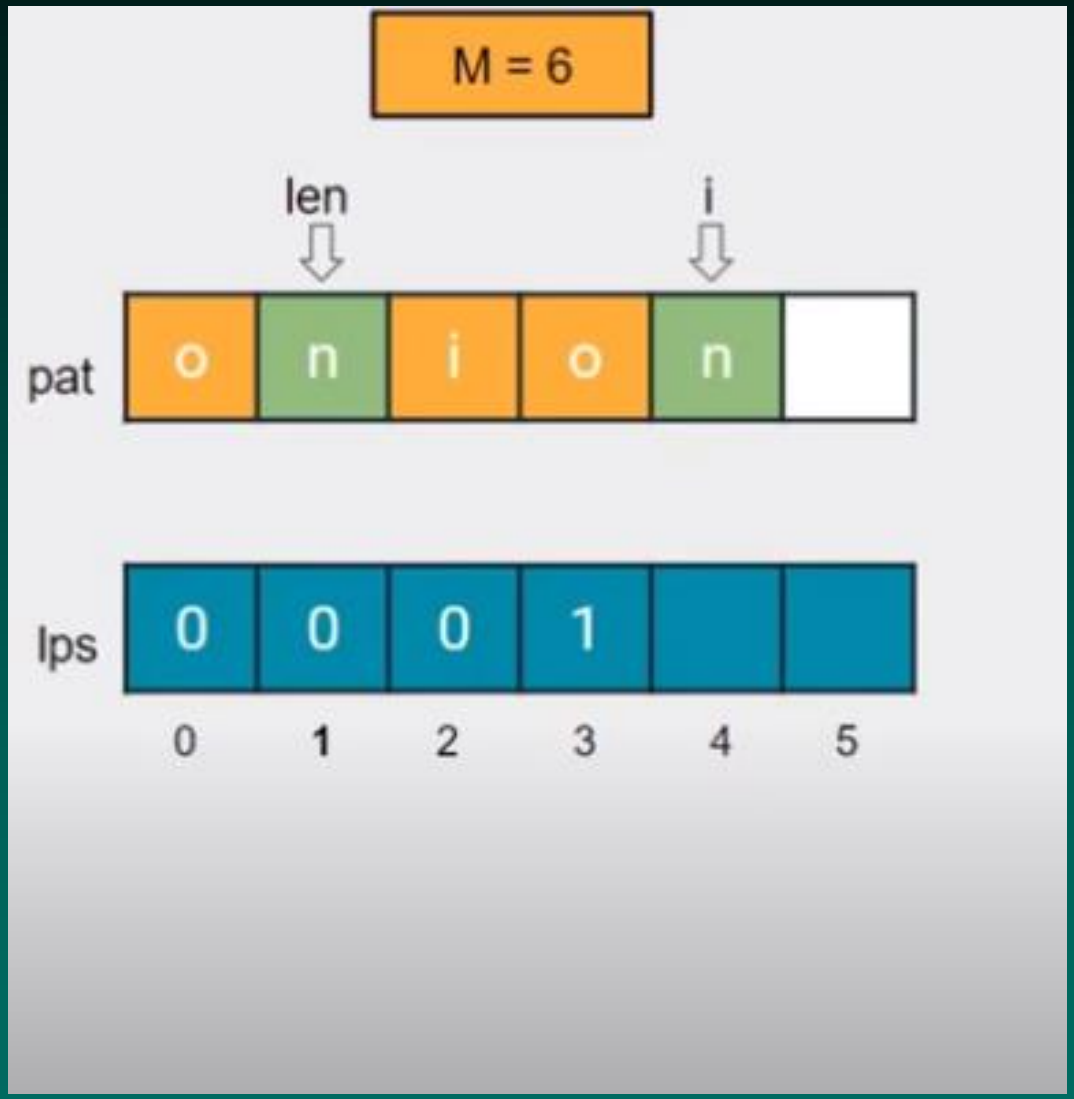
Longest prefix that is also a suffix

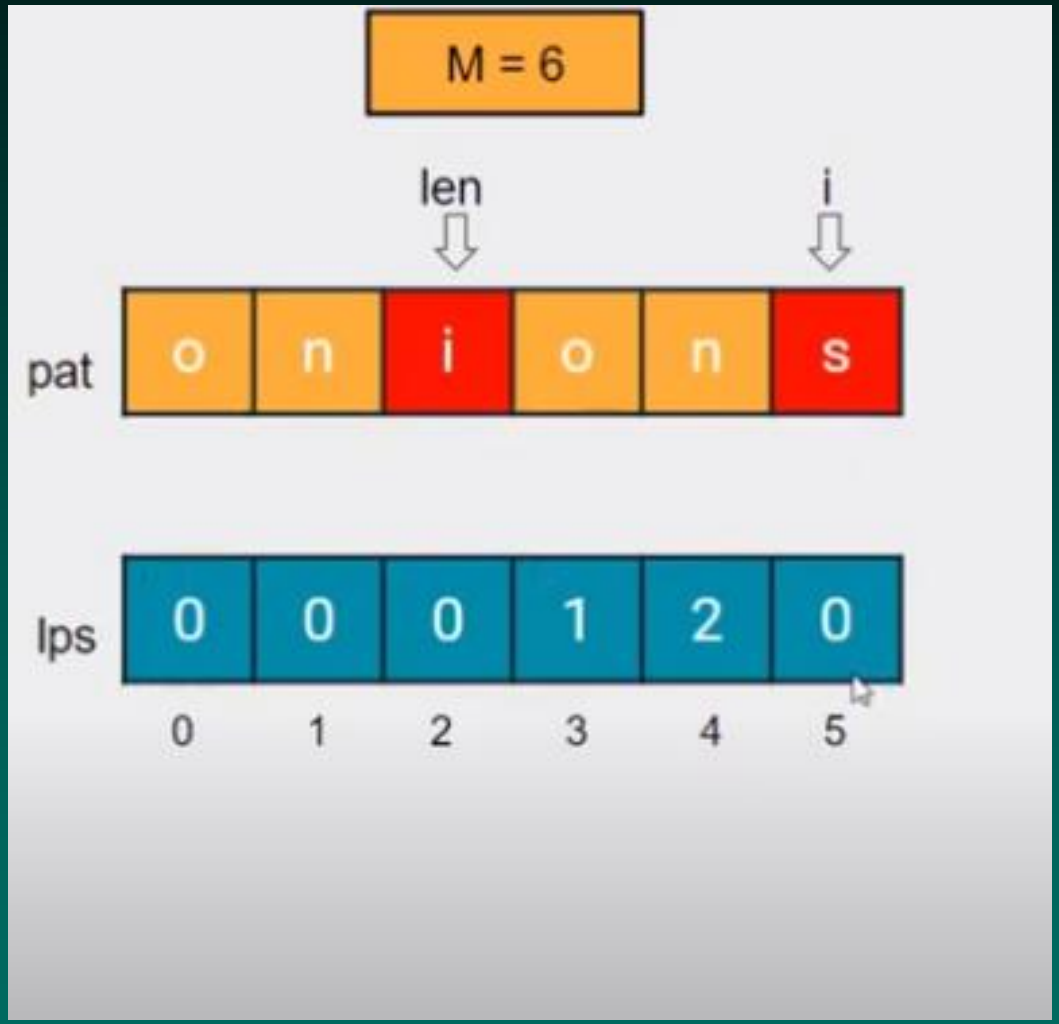


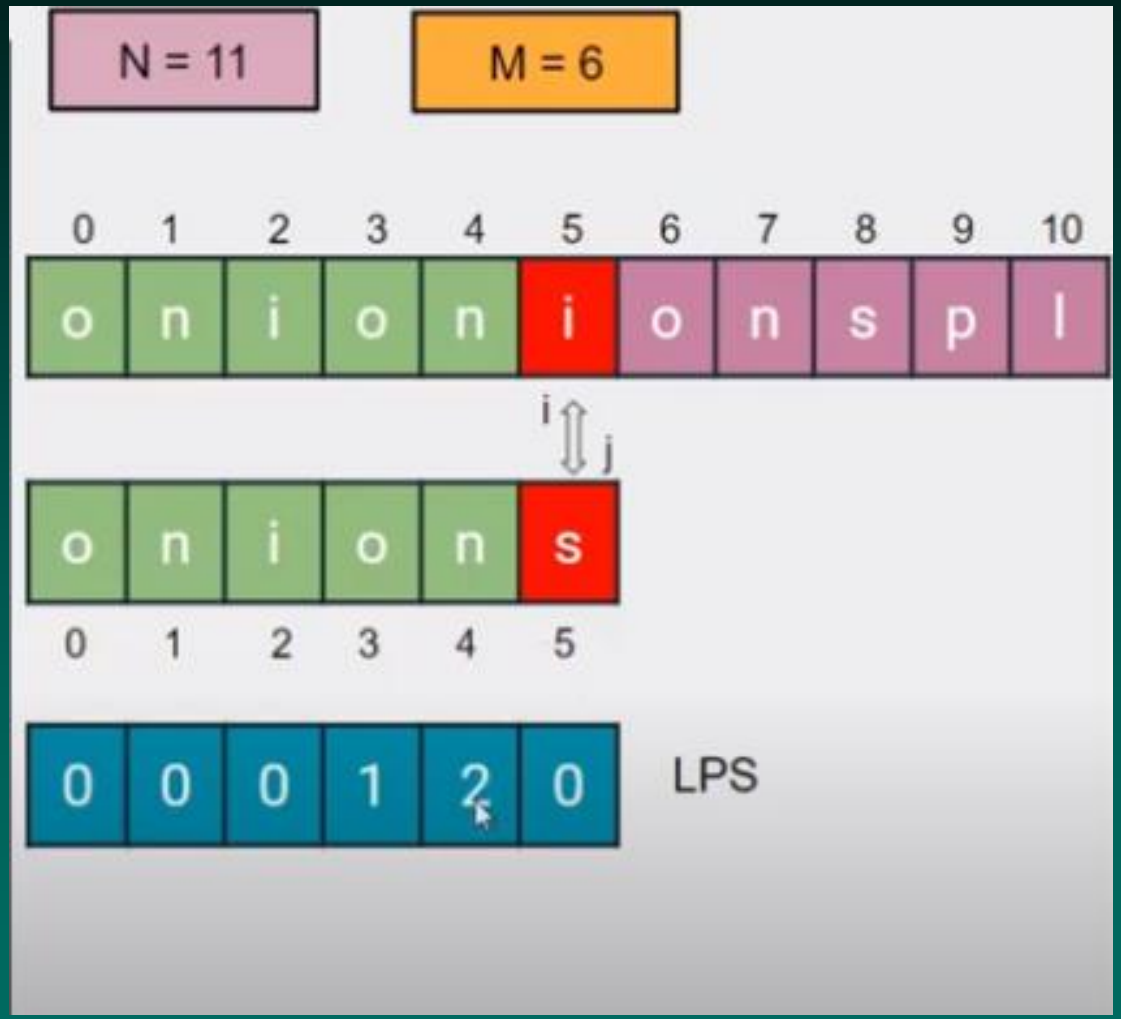

M = 6
len
i
pat
o
n
i
o
n
s
lps
0
0
1
2
3
4
5

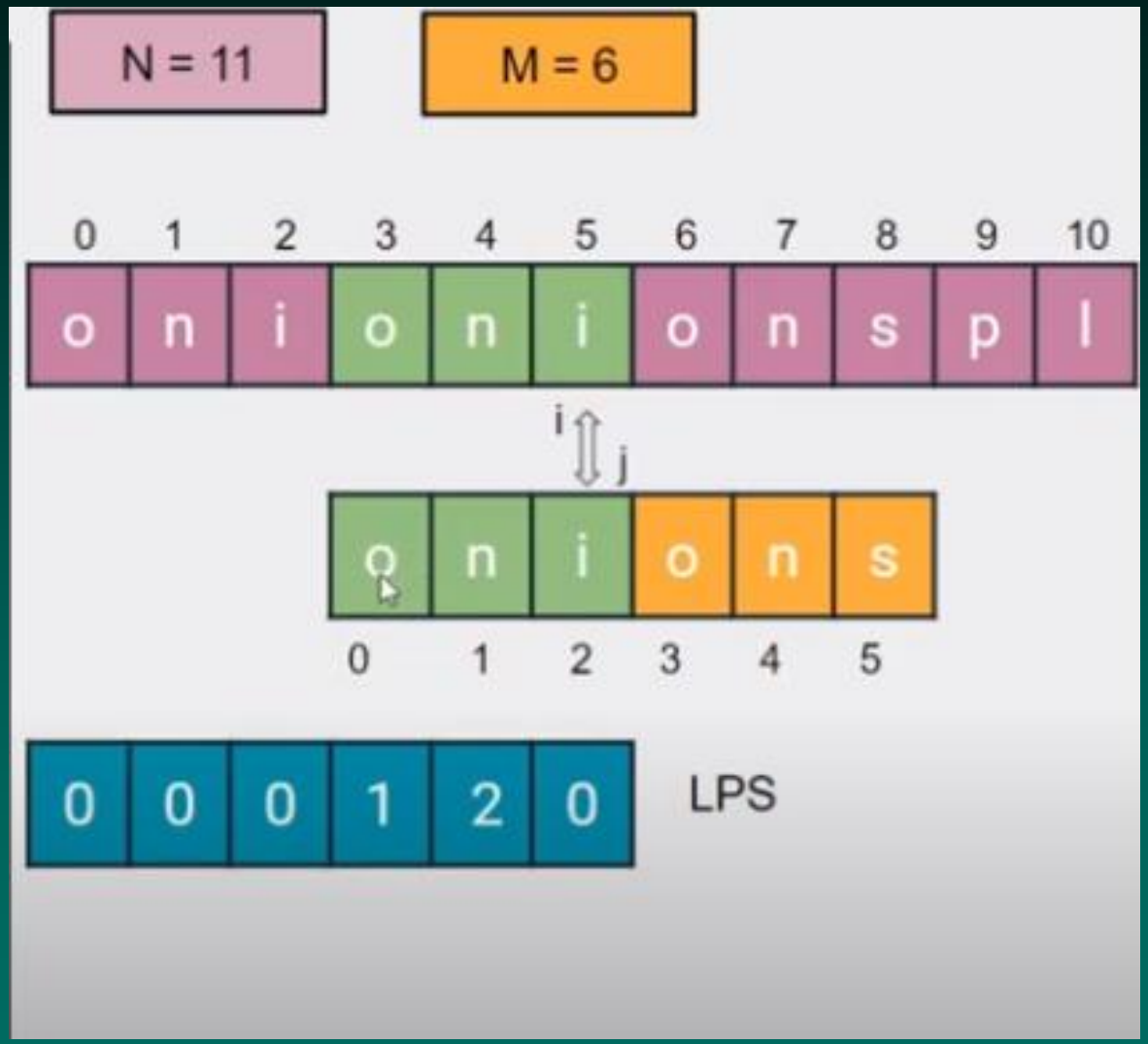

M = 6
len
i
pat
o
n
lps
0
0
0
1
2
3
4
5

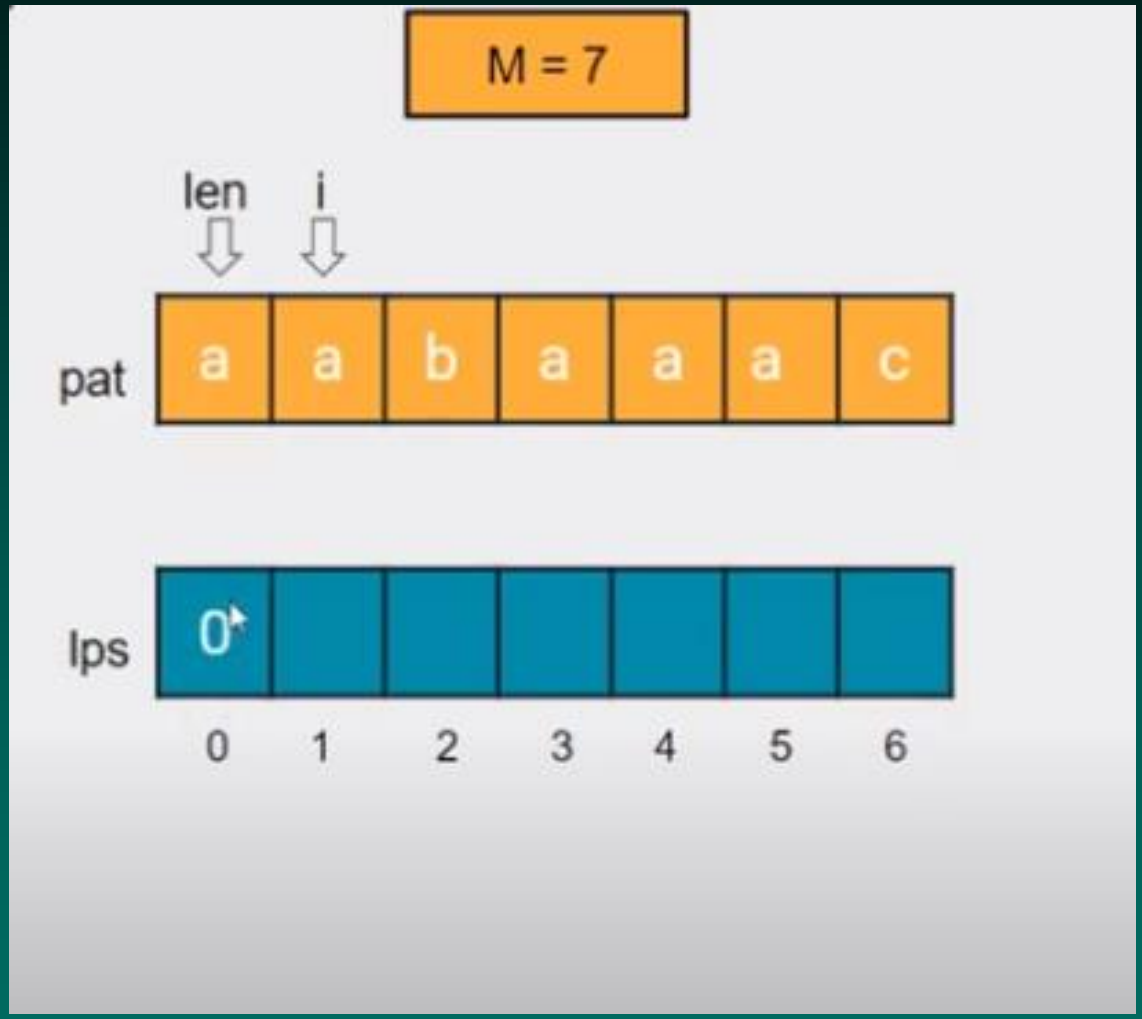

M = 6
len
i
pat
o
n
i
o
lps
0
0
0
1
0
1
2
3
4
5

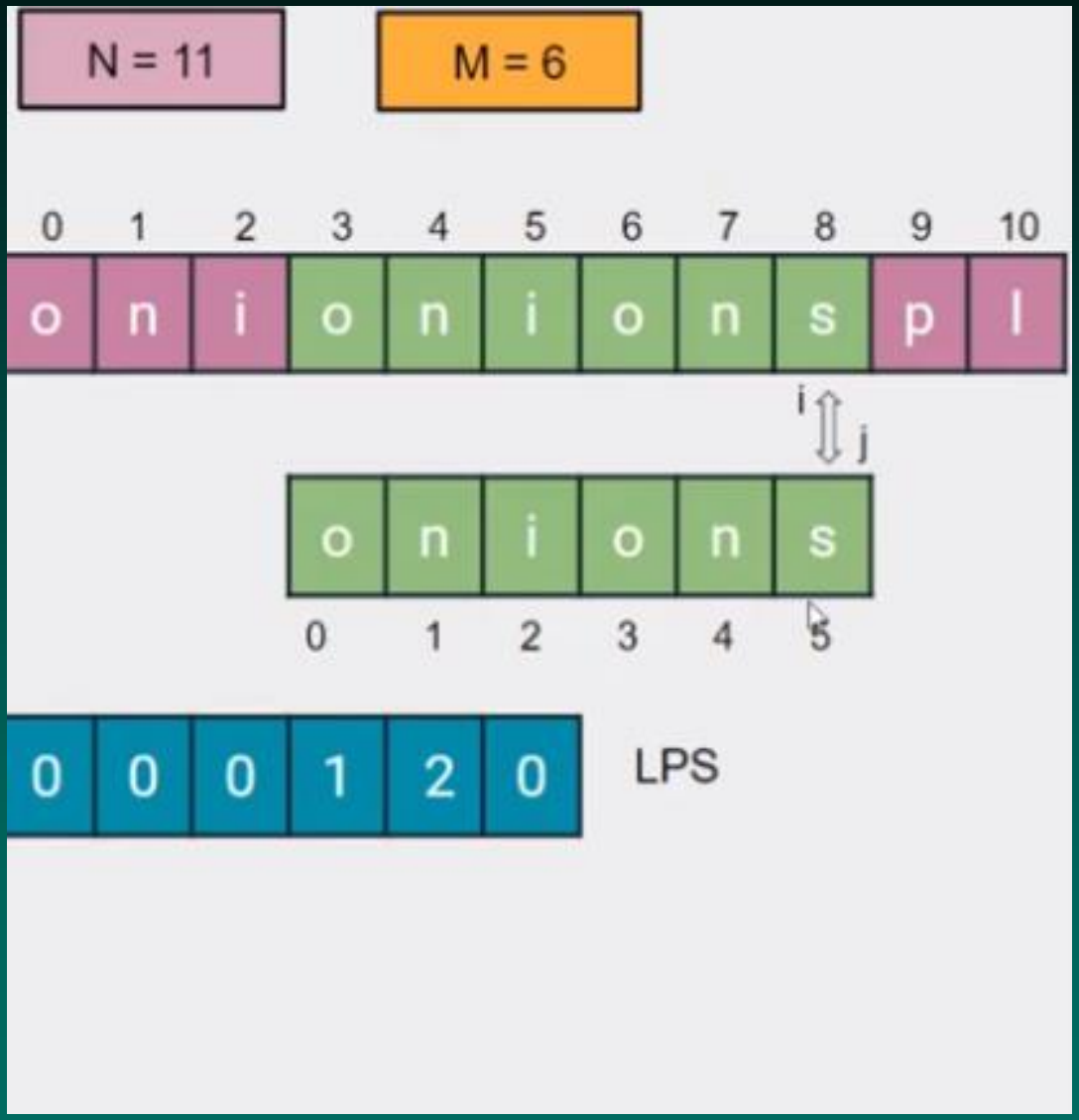

M = 6
len
i
pat
o
n
i
o
n
lps
0
0
0
1
0
1
2
3
4
5


M = 6
len
i
pat
o
n
i
o
n
s
lps
0
0
0
1
2
0
0
1
2
3
4
5


N = 11
M = 6
0 1 2 3 4 5 6 7 8 9 10
o n i o n i o n s p l
i
j
o n i o n s
0 1 2 3 4 5
0 0 0 1 2 0
LPS


N = 11
M = 6
0 1 2 3 4 5 6 7 8 9 10
o n i o n i o n s p l
i
j
o n i o n s
0 1 2 3 4 5
0 0 0 1 2 0
LPS


M = 7
len
i
pat
a a b a a a c
lps
0
0 1 2 3 4 5 6


N = 11
M = 6
0 1 2 3 4 5 6 7 8 9 10
o n i o n i o n s p l
i
j
o n i o n s
0 1 2 3 4 5
0 0 0 1 2 0
LPS

Time complexity

Overall -> O(n+m)

=> O(n) for n>>m

Space complexity

O(m) extra space for LPS array