# Knuth-Morris-Pratt (KMP) Algorithm and the Failure Function

## KMP

- The Knuth-Morris-Pratt (or KMP) algorithm is used to determine if a search string (p, of length m) occurs in a text (t, of length n).
- For example, if our text is “sorin,” and we want to find the search string ”rin,” the KMP algorithm would give us that the string occurs starting at the 3rd character (index 2) of the text.
- With the KMP algorithm, a worst-case run time of O(n+m) can be achieved. In other words, in the worst-case scenario we would have to look at every character in the text and search string at least once.

# KMP (example)

- Let’s run through how we determine where “rin” occurs in “sorin.” We begin by lining up our pattern string (“rin”) with the text “sorin.”

```
s o r i n
r i n
```

## KMP (example)

- Now, we must compare the first letter of the pattern string to the first letter of the text. They do not match, so we must shift over our pattern string (now lining up the ”r” with the “o”).

```
s o r i n
r i n
    ↓
s o r i n
  r i n
```

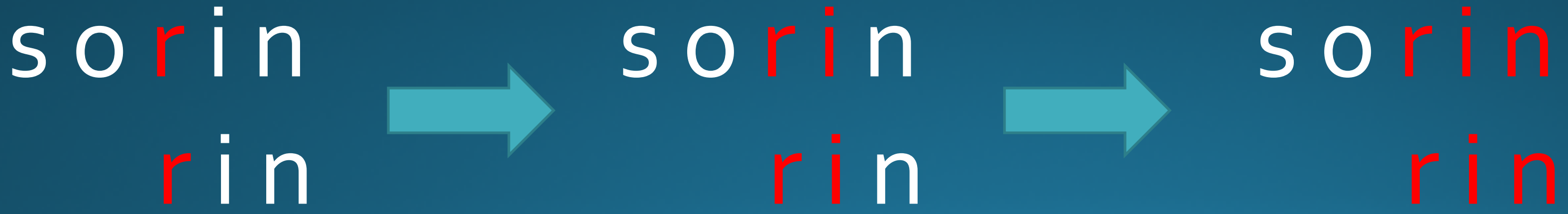
## KMP (example)

- We now compare the second letter of the text with the first letter of the pattern, and again there is no match. Thus, we must shift the search string over by one spot.

```
s o r i n
  r i n
    ↓
s o r i n
    r i n
```

# KMP (example)

- Next, we will compare the third letter, “r,” in ”sorin” to the “r” in ”rin.” This is a match, so we move on to the next letter in the pattern string to see if it matches the next letter in the text. Once again, there is a match, so we move on to the next letter. This one also matches (both are “n”), and we have reached the end of the pattern string, so we are done. The search string has been found in the text at index 3. Note that you could also continue the algorithm to find all of the indices in the text at which the pattern string occurs.



## Failure Function

- In order to for us to implement our searching algorithm more efficiently, we must first derive what is known as the "failure function."
- Given we have a search string (string we are searching for in a given sequence of letters), the failure function tells us the following: if we have character matched x letters of our string, what is the length of the longest suffix of these x letters that is also a prefix of the search string?
- If this can be determined at every index of our search string, we will know how much we should shift the string if a mismatch is found.

# Failure Function (pseudocode)

Input: Pattern p with m characters.

Output: Failure function (f)

This might be a little confusing, so let's run through an example while following the pseudocode. Afterwards, we will analyze our result and see why it makes sense.

```
$i \leftarrow 0$
$f(1) \leftarrow 0$

for every j from 2 to m do
    i = f(j-1)
    while p(j) ≠ p(i+1) and i > 0 do
        i = f(i);
    if p(j) ≠ p(i+1) and i = 0 then
        f(j) = 0
    else
        f(j) = i+1

return f
```

## Failure Function (walkthrough + example)

- Let’s say we have the following pattern (search string): ”abaabc”
- For every index, we want to determine the value that the failure function should return. In other words, we want to know, for every index j, what is the value of f(j) (failure function of j).

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | ? | ? | ? | ? | ? | ? |

# Failure Function (walkthrough + example)

- First, we must set a variable (i) to 0 and set the value of f(1) to 0.
- Next, for every value of j from 2 to 6 (since the length of abaabc is 6).
- First, we look at j=2:
  - i = f(2-1) = f(1) = 0
  - i is not greater than 0, so skip while loop
  - p(2) is not equal to p(1) (a is not equal to b), and i is equal to 0, so we set the value of f(2) to 0.

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | 0 | 0 | ? | ? | ? | ? |

# Failure Function (walkthrough + example)

- Next, we have j=3:
  - i = f(3-1) = f(2) = 0
  - i is not greater than 0, so skip while loop
  - p(3) is equal to p(1) (p(3) = p(1) = a), so we set the value of f(3) to i+1, or 1.

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | 0 | 0 | 1 | ? | ? | ? |

# Failure Function (walkthrough + example)

- Now we have j=4:
  - i = f(4-1) = f(3) = 1
  - p(4) = a and p(i+1) = p(2) = b, and i is greater than 0, so we enter the while loop.
    - In the first iteration, we set i equal to f(i), so we get i = f(i) = f(1) = 0.
    - Now i is equal to 0, so we break out of the loop.
  - p(4) = a and, since i is now 0, p(i+1) = p(1) = a. Since these are the same, we skip the first condition and instead set f(4) = i+1 = 1.

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | 0 | 0 | 1 | 1 | ? | ? |

# Failure Function (walkthrough + example)

- Next we have j=5:
  - i = f(5-1) = f(4) = 1
  - p(5) = b and p(i+1) = p(2) = b, so we skip the while loop.
  - p(5) = b and p(i+1) = p(2) = b. Since these are the same, we skip the first condition and instead set f(5) = i+1 = 2.

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | 0 | 0 | 1 | 1 | 2 | ? |

# Failure Function (walkthrough + example)

- Lastly, we have j=6:
  - i = f(6-1) = f(5) = 2
  - p(6) = c and p(i+1) = p(3) = a, and i > 0, so we enter the while loop.
    - First iteration: set i = f(i) = f(2) = 0.
    - i is now equal to 0 so we exit the loop.
  - p(6) = c and p(i+1) = p(1) = a. Since these are not same, and i is equal to 0, we set f(6) equal to 0.

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| p(j) | a | b | a | a | b | c |
| f(j) | 0 | 0 | 1 | 1 | 2 | 0 |

# Failure Function (explanation)

- The finished table below represents the failure function, which tells us, for every string of length j from j =2  to j =6, the length of the longest suffix of the j letters that is a prefix of the entire string.
- If we look at our table, this can be seen.
  - At j = 2, we have the suffix “b,” and since this is not a prefix of the entire string, f(2) = 0.
  - At j = 3, we have the suffices “ba” and ”a.” ”a” is also a prefix of the entire string, so we write its length as the value for f(3) (f(3) = 1).  The same is true for j =4.
  - At j=5, we have the suffices “baab,” “aab,” “ab,” and  “b.” “ab” is a prefix of the entire string, and it has length 2, so we set the value of f(5) to 2.
  - At j=6, every suffix ends with “c,” and there is no way to get a prefix of the entire string that has a “c” in it, therefore f(6) = 0.

# Failure Function (purpose)

- Knowing the values for the failure function is important for the KMP algorithm because it allows us to know how much we should "shift" our pattern string over if we encounter a mismatch.
- For instance, in the case of "abaabc," let's say we encounter a mismatch at "c." We now must look at the failure function value for the last index that was matched correctly (index 5, the "b"). This gives us 2.
- This value (2) tells us that the characters at index 4 and 5 are the same as those at 1 and 2, so, when shifting over our pattern string, we must line up the character at index 3 with the character that we just failed to match the "c" with (as opposed to lining it up with the character at index 1, which is done if the failure function value is 0).

## Example

- We will now show the series of comparisons that will be done when we try to find the string “abaabc” in a text “abccabaabaabc.” Red indicates characters that match, green indicates characters currently being compared.

a b c c a b a a b a a b c

a b a a b c

↓ Match

a b c c a b a a b a a b c

a b a a b c

↓ Match

## Example

a b c c a b a a b a a b c

a b a a b c

Mismatch, shift by length of pattern string since f(2) = 0

a b c c a b a a b a a b c

a b a a b c

Mismatch, shift by length of pattern string since f(1) = 0

a b c c a b a a b a a b c

a b a a b c

Match

## Example

a b c c a b a a b a a b c

a b a a b c

Match

a b c c a b a a b a a b c

a b a a b c

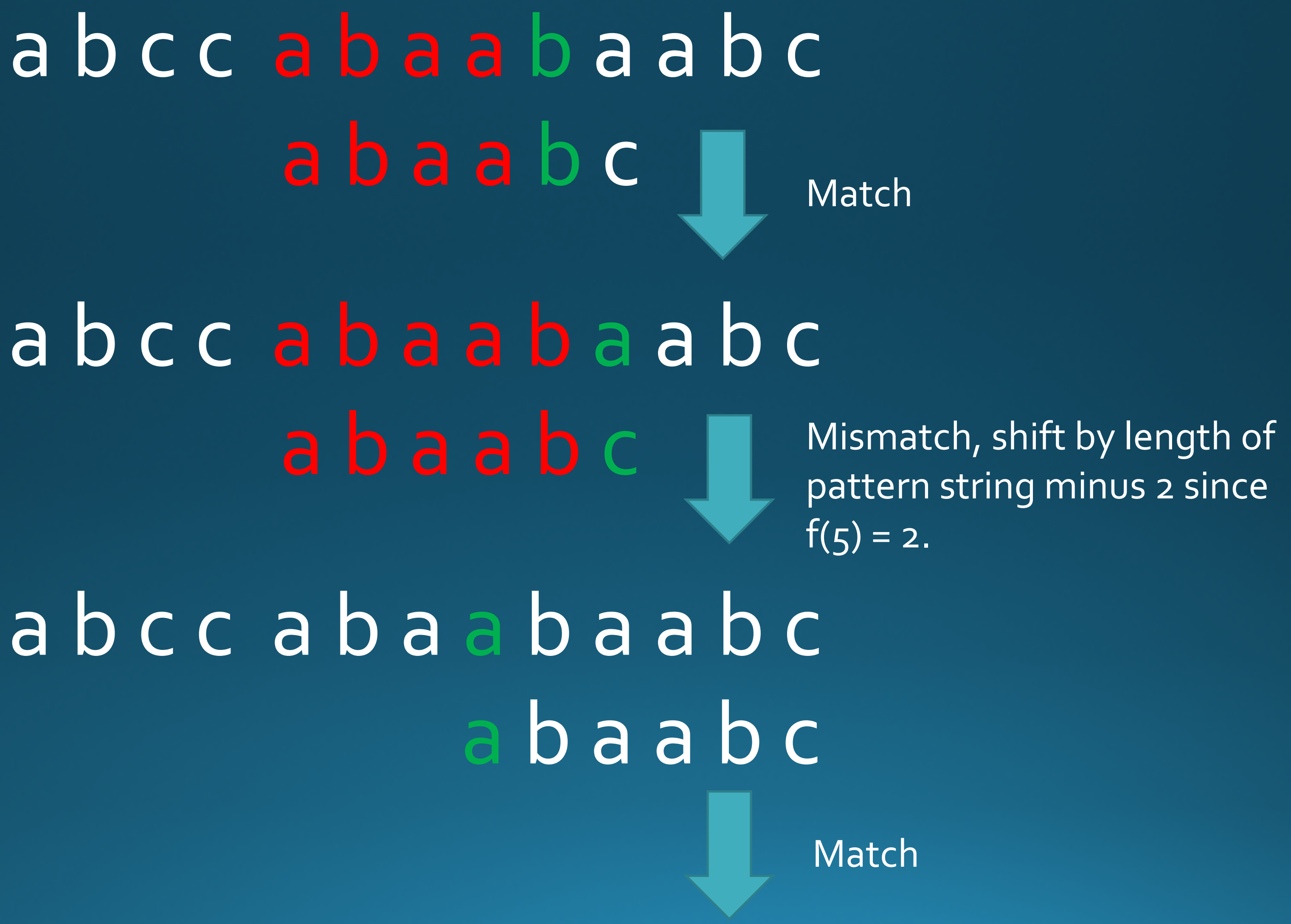
Match

a b c c a b a a b a a b c

a b a a b c

Match

# Example



a b c c a b a a b a a b c
a b a a b c
Match
a b c c a b a a b a a b c
a b a a b c
Mismatch, shift by length of pattern string minus 2 since f(5) = 2.
a b c c a b a a b a a b c
a b a a b c
Match

# Example

a b c c a b a a b a a b c

a b a a b c

↓ Match

a b c c a b a a b a a b c

a b a a b c

↓ Match

a b c c a b a a b a a b c

a b a a b c

↓ Match

## Example

a b c c a b a a b a a b c

a b a a b c

Match

a b c c a b a a b a a b c

a b a a b c

Match

a b c c a b a a b a a b c

a b a a b c

Pattern found