String Matching Algorithms

Author	Enes Kızılcan
Link	https://github.com/eneskzlcn
Publishing/Release Date	@November 11, 2021

String Matching Algorithms

In this work, I will search and implement (if possible) the string matching, string searching algorithms in below.

- Boyer Moore Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm
- Naive Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm
- KMP Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm
- Rabin-Karp Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm
- Aho-Corasick Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm
- Z Algorithm
    - How this algorithm works?
    - Implementation Of Algorithm

Naive Algorithm

How this algorithm works?

This algorithms works in a brute force approach. Let think you have a text and pattern like in example below:

> Text: A B C D E D G H
>
> Pattern: D G

The algorithm compares the first character in the text with the first character in pattern firstly.

If any match, then second character of text and pattern will be comparised. This operation will go over an over for each character in pattern. If all the character matches, then you found this pattern at this index of the text.

If any mismatch, then you shift pattern 1 index and continue to compare operation again. Let show in example above.

1.Step: Mismatch A ≠ D . So shift the pattern

> A B C D E D G H
> D G

2.Step: Mismatch B ≠ D . So shift the pattern

> A B C D E D G H
>   D G

3.Step: Mismatch C ≠ D . So shift the pattern

> A B C D E D G H
>   D G

4.**Step: Match In D= D. So look next character. But next is mismatch E ≠ G . So shift the pattern**

> A B C D E D G H
>   D G

5.Step: Match In E≠D So shift the pattern .

> A B C D E D G H
>   D G

6.Step: Match In D = D . And all the other characters in pattern matches too. So you found a match at that index. It is okey to continue to found other ones in exactly the same way.

> A B C D E D G H
>   D G

Implementation of algorithm in javascript

```
function naive_string_matching(pattern, text) // 'text' is the string that we search for given 'pattern' string.
{
    //the time complexity of naive string matching algorithm is  O( m * (n - m + 1)) where n is the text length and m is the pattern length

    //we need to find the length of pattern and text firstly.
    //This variables for just for understanding and reading clearly. String in js have a length property simply.
    //You do not need to declare them in your code.
    var text_length = text.length;
    var pattern_length = pattern.length;

    let pattern_found_indexes = [];
    //we need to loop from 0th index of text to the ( text_length - pattern_length)th index. This is the last index can pattern start from.
    let last_possible_patern_index = (text_length - pattern_length);
    let matched_character_count;

    for (let i = 0; i <= last_possible_patern_index; i++) {
        matched_character_count = 0;
        for (let j = 0; j < pattern_length; j++) {
            if (text.charAt(i + j) != pattern.charAt(j)) break; //if any mismatch directly break.
            matched_character_count++;
        }
        if (matched_character_count == pattern_length) // if there is no break in upper loop that means fullmatch. So push it.
```

```
            pattern_found_indexes.push(i);
    }
    return pattern_found_indexes;
}
```