Experiment No. 10
Naïve String matching
Date of Performance:03/04/2024
Date of Submission:10/04/2024

EXPERIMENT NO. 10

Title: Naïve String matching

Aim: To study and implement Naïve string-matching Algorithm

Objective: To introduce String matching methods

Theory:

Naive pattern searching is the simplest method among other pattern searching algorithms. It checks for all character of the main string to the pattern.

Naive algorithm is exact string matching (means finding one or all exact occurrences of a pattern in a text) algorithm.

This algorithm is helpful for smaller texts. It does not need any pre-processing phases. We can find substring by checking once for the string. It also does not occupy extra space to perform the operation.

The naïve approach tests all the possible placement of Pattern P [1.....m] relative to text T [1.....n]. We try shift s = 0, 1.....n-m, successively and for each shift s. Compare T [s+1.....s+m] to P [1.....m].

The naïve 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 value of s.

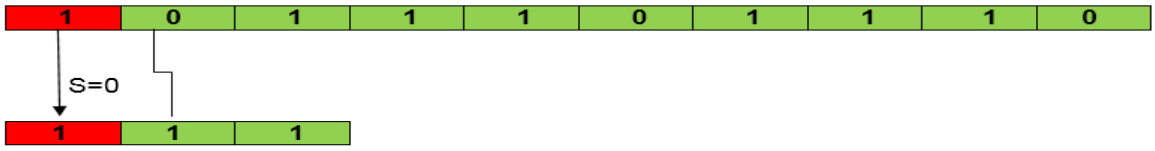
Working of Naive String Matching

The naive-string-matching procedure can be interpreted graphically as sliding a "template" containing the pattern over the text, noting for which shifts all of the characters on the template equal the corresponding characters in the text.

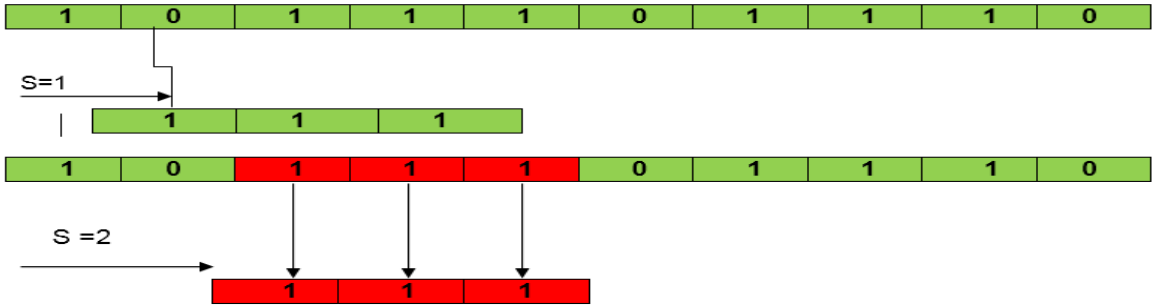
Example:

Suppose T = 1011101110 P = 111

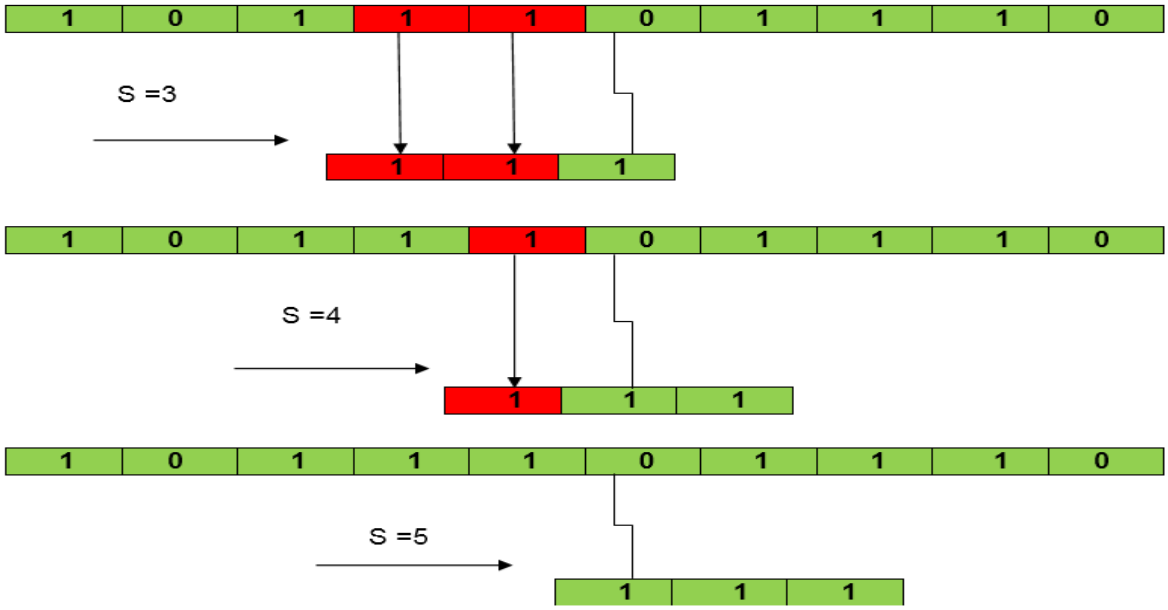
T = Text



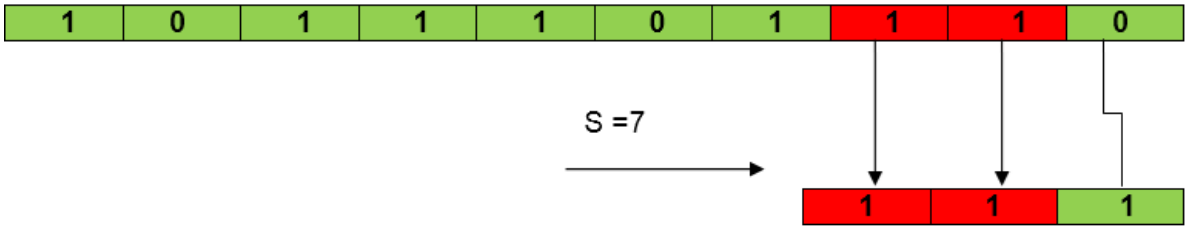
P = Pattern



So, S=2 is a Valid Shift



So, S=6 is a Valid Shift



Example:

Input:

main String: " Hello World!"

pattern: "World!"

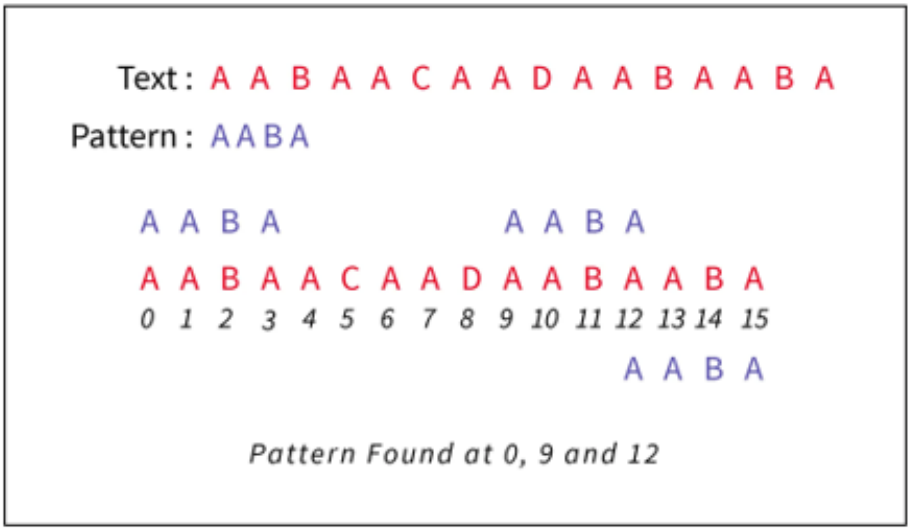
The size of the pattern is 6 (i.e. m) and the size of the input text is 12 (i.e. n).

We can start searching for the pattern in the input text by sliding the pattern over the text one by one and checking for a match.

So, we would start searching from the first index and slowly move our pattern window from index-0 to index-6. At index-6, we can see that both the W's are matching. So, we will search the entire pattern in the window starting with index-6.

Hence, we will find a match and return the starting index of the pattern as the answer.

Another Example:



Algorithm:

NAIVE-STRING-MATCHER (T, P)

Step 1. n ← length [T]

Step 2. m ← length [P]

Step 3. for s ← 0 to n -m

Step 4. do if P [1.....m] = T [s + 1....s + m]

Step 5. then print "Pattern occurs with shift" s

Time Complexity Analysis:

Best Case: O(n):

- When the **pattern** is found at the very beginning of the **text** (or very early on).
- The algorithm will perform a constant number of comparisons, typically on the order of **O(n)** comparisons, where n is the length of the **pattern**.
- The best case occurs when the first character of the pattern is not present in text at all.

  txt[] = "BBACCAADDEE";

  pat[] = "HBB";

  The number of comparisons in best case is O(n).

Worst Case: $O(n^2)$:

- When the **pattern** doesn't appear in the **text** at all or appears only at the very end.
- The algorithm will perform **O((n-m+1)*m)** comparisons, where **n** is the length of the **text** and **m** is the length of the **pattern**.
- In the worst case, for each position in the **text**, the algorithm may need to compare the entire **pattern** against the text.
- The worst case of Naive Pattern Searching occurs in following scenarios. When all characters of the text and pattern are same.

  txt[] = "DDDDDDDDDDDD";

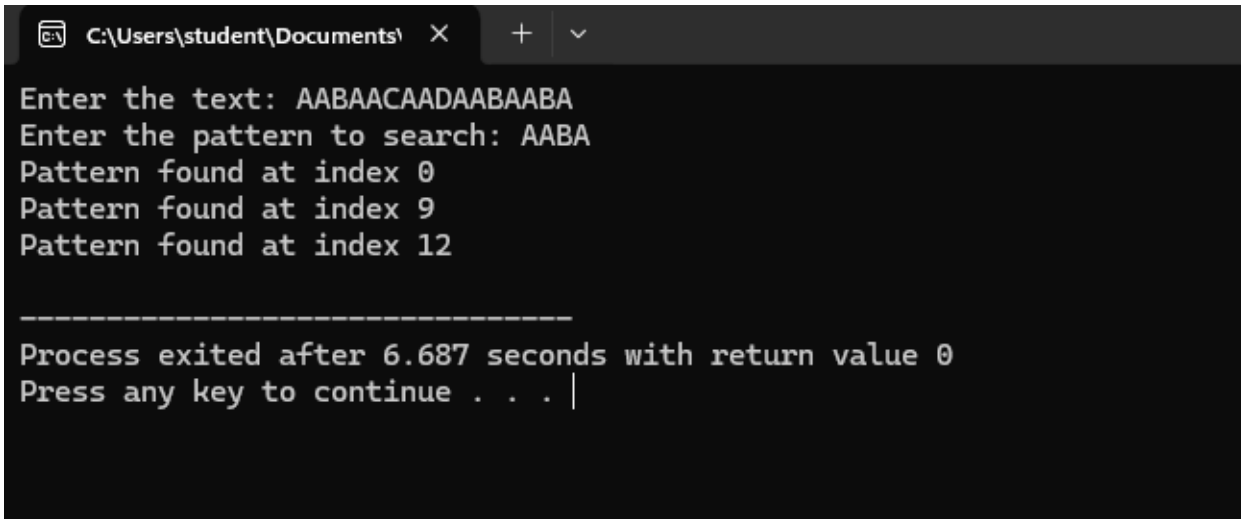
  pat[] = "DDDDD";

Program:

```
#include <stdio.h>
#include <string.h>


void stringMatch(char text[], char pattern[]) {
    int textLen = strlen(text);
    int patternLen = strlen(pattern);

    for (int i = 0; i <= textLen - patternLen; i++) {
        int j;
        for (j = 0; j < patternLen; j++) {
            if (text[i + j] != pattern[j])
                break;
        }
        if (j == patternLen)
            printf("Pattern found at index %d\n", i);
    }
}

int main() {
    char text[100];
    char pattern[100];

    printf("Enter the text: ");
    scanf("%s", text);

    printf("Enter the pattern to search: ");
    scanf("%s", pattern);
```

```
    stringMatch(text, pattern);

    return 0;
}
```

Output:



Conclusion:

In conclusion, the Naive String Matching algorithm is a simple yet effective method for finding occurrences of a pattern within a text. It compares the pattern with every possible substring of the text, making it straightforward to implement. However, its time complexity is relatively high, O((n - m + 1) * m), where 'n' is the length of the text and 'm' is the length of the pattern. This means its efficiency decreases with longer texts and patterns. Despite this drawback, the Naive String Matching algorithm remains a valuable tool for small-scale string matching tasks or as a baseline for more sophisticated algorithms.