Experiment No. 12
Naïve String matching
Date of Performance:
Date of Submission:

Experiment No. 12

Title: Naïve String matching

Aim: To study and implement Naïve string matching Algorithm

Objective: To introduce String matching methods

Theory:

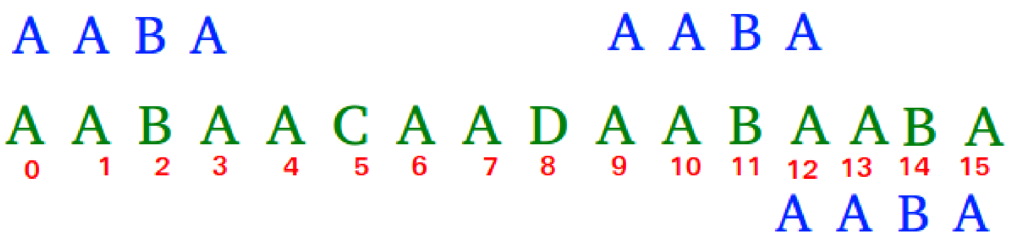
The naïve approach tests all the possible placement of Pattern P [1.......m] relative to text T [1......n]. We try shift s = 0, 1.......n-m, successively and for each shift s. Compare T [s+1.......s+m] to P [1......m].

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.

Example:

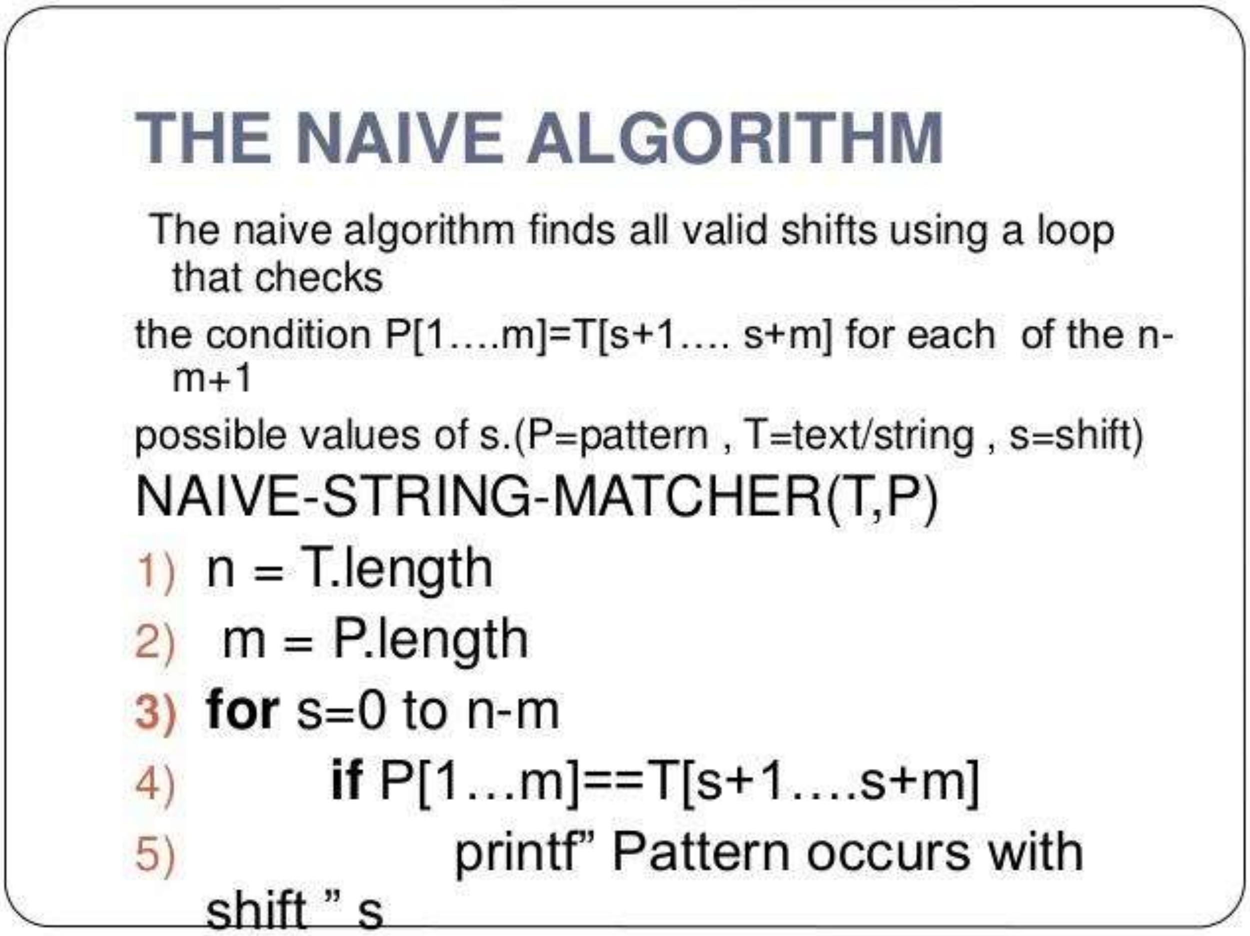
Text : A A B A A C A A D A A B A A B A

Pattern : A A B A



Pattern Found at 0, 9 and 12

Algorithm:



THE NAIVE 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.(P=pattern , T=text/string , s=shift)

NAIVE-STRING-MATCHER(T,P)

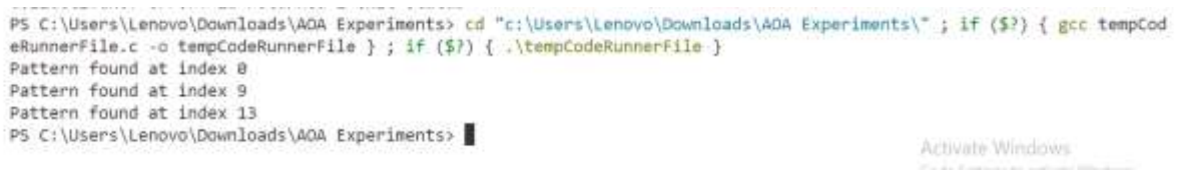
```
1) n = T.length
2)  m = P.length
3) for s=0 to n-m
4)        if P[1…m]==T[s+1….s+m]
5)              printf" Pattern occurs with
   shift " s
```

Implementation:

Code:

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

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++) {
        int j;
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j== M)
            printf("Pattern found at index %d \n", i);
    }
}

int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";
    search(pat, txt);
    return 0;
}
```

1. **Output:**



```
PS C:\Users\Lenovo\Downloads\AOA Experiments> cd "c:\Users\Lenovo\Downloads\AOA Experiments\" ; if ($?) { gcc tempCod
eRunnerFile.c -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
PS C:\Users\Lenovo\Downloads\AOA Experiments> 
```

Conclusion:

In conclusion, the Naive String Matching algorithm efficiently locates occurrences of a given pattern within a text string. This algorithm, also known as the Brute Force algorithm, systematically compares the pattern with all substrings of the text to identify matches.

The implementation in this program iterates through the text string, comparing substrings of the same length as the pattern with the pattern itself. It employs a simple nested loop structure, where for each position in the text, it checks if the characters starting from that position match the characters of the pattern.