EXPERIMENT NO- 09

AIM: Implementation of Naïve String Matching Algorithm.

PROBLEM STATEMENT : Write a program to implement Naïve String Matching Algorithm.

RESOURCE REQUIRED: Pentium IV, Turbo C, Printer, Printout Stationary

THEORY:

In the Naive String matching algorithm, we always slide the pattern by 1. When all characters of pattern are different, we can slide the pattern by more than 1. When a mismatch occurs after j matches, we know that the first character of pattern will not match the j matched characters because all characters of pattern are different. So we can always slide the pattern by j without missing any valid shifts. The naïve approach simply test all the possible placement of Pattern P[1 . . m] relative to text T[1
. . n].

The naïve string-matching procedure can be interpreted graphically as a sliding a pattern P[1 . . m] over the text T[1 . . n] and noting for which shift all of the characters in the pattern match the corresponding characters in the text.

ALGORITHM :

```
NAÏVE_STRING_MATCHER (T, P)
    n ← length [T]
    m ← length [P]
    for s ← 0 to n - m do
        if P[1 . . m] = T[s +1 . . s + m]
            then return valid shift s
```

INPUT:

Main String: “ABAAABCDBBABCDDEBCABC”, pattern: “ABC”

OUTPUT:

Pattern found at position: 4 Pattern found at position: 10 Pattern found at position: 18

CODE:

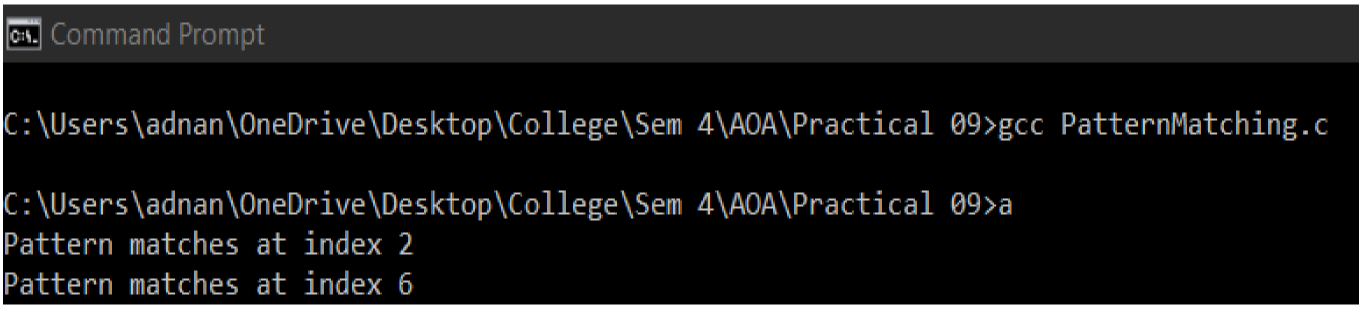
```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main (){
  char txt[] = "1011101110";
  char pat[] = "111";
  int M = strlen (pat);
  int N = strlen (txt);
  int i;
  for(i = 0; i <= N - M; i++){
    int j;
    for (j = 0; j < M; j++)
```

```
      if (txt[i + j] != pat[j])
    break;
    if (j == M)
      printf ("Pattern matches at index %d \n", i);
  }
   getch();

  return 0;

}
```

OUTPUT:



```
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Practical 09>gcc PatternMatching.c

C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\Practical 09>a
Pattern matches at index 2
Pattern matches at index 6
```

CONCLUSION: The running time of the algorithm is O((n - m +1)m), which is clearly O(nm). Hence, in the worst case, when the length of the pattern, m are roughly equal, this algorithm runs in the quadratic time. One worst case is that text, T, has n number of A's and the pattern, P, has (m -1) number of A's followed by a single B.