Experiment No : 10

Name: Shawn Louis

Roll No: 31

Batch: B

Problem Statement:	**To implement KMP Algorithm for String Matching**
Objective:	• To be able to implement a string matching algorithm.
Expected Outcome:	• **To find the occurrence of pattern string in the text string.** • **To implement pre-processing and matching step.**
Theory:	The **KMP Algorithm** (or Knuth, Morris and Pratt string searching algorithm) cleverly make use of previous comparison's data. It can search a pattern in O(n) time as it never re-compares a text symbol that has matched a pattern symbol. However, it uses a partial match table to analyze the pattern structure. Construction of partial match table takes O(m) time. Therefore, the overall complexity of the KMP algorithm is O(m + n).
Algorithm:	

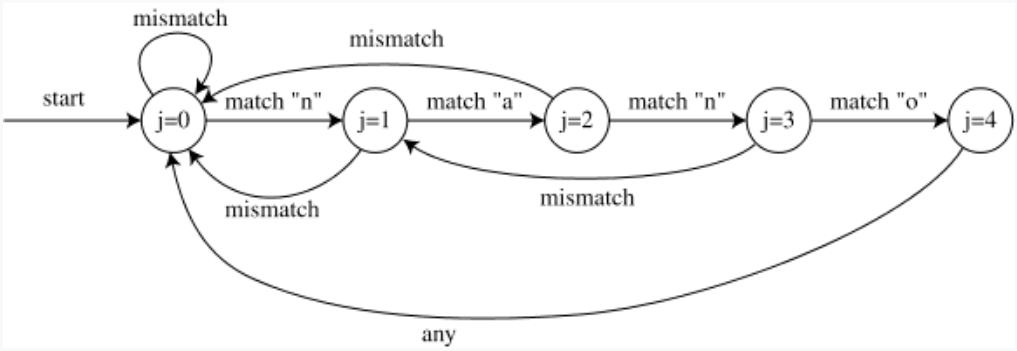
```
algorithm kmp_search:
    input:
        an array of characters, S (the text to be searched)
        an array of characters, W (the word sought)
    output:
        an array of integers, P (positions in S at which W
is found)
        an integer, nP (number of positions)

    define variables:
        an integer, j ← 0 (the position of the current
character in S)
        an integer, k ← 0 (the position of the current
character in W)
        an array of integers, T (the table, computed
elsewhere)

    let nP ← 0

    while j < length(S) do
        if W[k] = S[j] then
            let j ← j + 1
            let k ← k + 1
            if k = length(W) then
                (occurrence found, if only first occurrence
is needed, m ← j - k  may be returned here)
                let P[nP] ← j - k, nP ← nP + 1
                let k ← T[k] (T[length(W)] can't be -1)
```

```
        else
            let k ← T[k]
            if k < 0 then
                let j ← j + 1
                let k ← k + 1
```



Program Code:

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

void longest_prefix_suffix(char P[], int f[])
{
  int i, j, m;
  m = strlen(P);

  j = 0;
  i = 1;
  f[0] = 0;

  while(i < m)
  {
    if(P[i] == P[j])
    {
      f[i] = j + 1;
      i++;
      j++;
    }
    else
    {
      if(j != 0)
                    j = f[j-1];
      else
      {
                    f[i] = 0;
                    i++;
      }
    }
```

```
  }

  printf("\nPrefix-Suffix array is as shown : \n");
  for(i = 0; i < m; i++)
    printf("%d ", f[i]);
  printf("\n");
}


void KMP(char T[], char P[])
{
     int i, j, m, n;
     int f[100];

     longest_prefix_suffix(P, f);

     m = strlen(P);
     n = strlen(T);

     i = 0; j = 0;

     while(i < n)
     {
          if(P[j] == T[i])
    {
      j++;
      i++;
    }

    if(j == m)
    {
      printf("\nPattern found at position %d\n", i-
j);
      j = f[j-1];
    }
    else if(P[j] != T[i])
    {
      if(j != 0)
                    j = f[j-1];
      else
                    i = i + 1;
    }
     }
}
```

```
void main()
{
  char T[100], P[100];
  clrscr();

  printf("\nEnter the Text string    : ");
  gets(T);

  printf("\nEnter the Pattern string : ");
  gets(P);

  KMP(T, P);

  getch();
}
```

Output Snapshot:

```
Enter the Text string    : ababaabaab

Enter the Pattern string : abaa

Prefix-Suffix array is as shown :
0 0 1 1

Pattern found at position 2

Pattern found at position 5
_
```

```
Enter the Text string    : ababcabcabababd

Enter the Pattern string : ababd

Prefix-Suffix array is as shown :
0 0 1 2 0

Pattern found at position 10
```

Application

Its imperative applications are Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics, Digital Forensics and Information Retrieval Systems etc.

Outcome:	Successfully implemented Knuth-Morris-Pratt Algorithm for String (Pattern) Matching in C.