Jawahar Education Societys Annasaheb Chudaman Patil College of Engineering, Kharghar, Navi Mumbai

NAME: PRIYUSH BHIMRAO KHOBRAGADE

PRN NO: 211112018

Roll No: 52

SUBJECT: Analysis of Algorithms Lab

Implement a C program for the Knuth-Morris-Pratt Pattern matching algorithm.

EXPERMINT: 10

PAGE NO.:
DATE.: / / 20

Experiment No - 10

- **Aim :-** Implement a C program for the kuuth- Morris - Pratt pattern matching pat algorithm.

- **Objectives :-** To implement and analyze complexity of kmp pattern matching algorithm.

- **Outcome :-** Student will be able to compute the complexity of kmp Pattern matching algorithm

- **Hardware/Software Required :-** "Turbo C"

- **Theory :-**

The Kuuth- Morris- pratt string searching algorithm (or kmp algorithm) searches for occurrence of a 'word' W within a main 'text string' S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. The algorithm was conceived in 1970 by Donald kunth and Vaughan pratt, and independently by James H. Morris. This is the first linear o(n) time algorithm for string matching.

- **Procedure / Algorithm :-**

Algorithm PrefixTable(P)
i = 0 & j = 0
Prefixtable(0) = 0
while (i < m) do // we have matched j+1 characters
  Prefixtable[i] = j+1
  i = i+1
  j = j+1

Teachers Signature _____ 1

Implement a C program for the Knuth-Morris-Pratt Pattern matching algorithm.

PAGE NO.:
DATE.: / / 20

```
else
    if (j>0)          // j - index just after a prefix of p that matches
      j = Pretable (j-1)
else
    // we have no match
    Pretable (i) = 0;
    i = i+1
```

- Algorithm KMP (T, P)

```
i=0 , j= 6;
while (i<n)
    if(p[j] = t[i]
      if (j = = m-1)
                return (i-m+1)        // match found
      i = i+1
      j = j+1
else
      if (j>0)
          j- Pretable (j-1)
    else
          i = i+1
return -1                           // no match
```

- **Conclusion :**

Assuming the prior existence of the table T, the search portion of ~~Knuth~~ Knuth-morris-pratt algorithm has complexity O(n), where n is the length S.

Teachers Signature

2

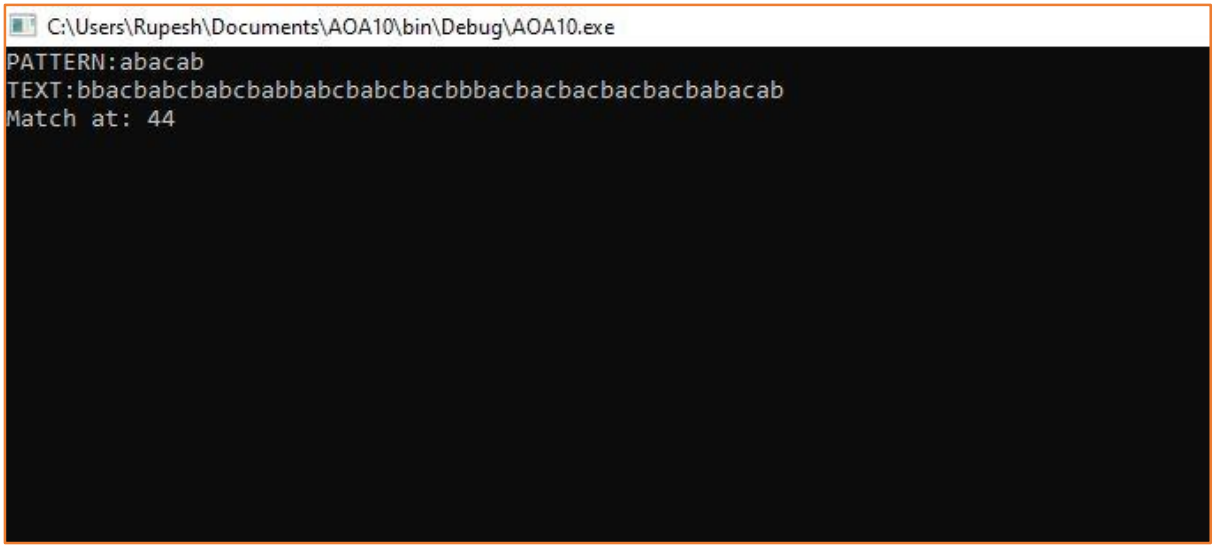
Implement a C program for the Knuth-Morris-Pratt Pattern matching algorithm.

Input:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
void failure(char* pattern, int* f);
int kmp(char* t, char* p);
void failure(char* p, int* f)
{
int i,j,m;
f[0] = 0;
i = 1;
j = 0;
m = strlen(p);
while (i < m)
{
if (p[i] == p[j])
{
f[i] = j + 1; // j+1 matches up to the current character.
i += 1;
j += 1;
}
else if (j > 0)
{
j = f[j - 1];
}
else
{
f[i] = 0;
i += 1;
}
}
}
int* init_array(int size)
{
int* arr = (int*)malloc(size * sizeof(int));
int i;
for(i = 0; i < size; i++)
{
arr[i] = 0;
}
return arr;
}
void main()
{

int match;
char* pattern = "abacab";
char* text = "bbacbabcbabcbabbabcbabcbacbbbacbacbacbacbacbabacab";

printf("PATTERN:abacab");
printf("\nTEXT:bbacbabcbabcbabbabcbabcbacbbbacbacbacbacbacbabacab");
match = kmp(text, pattern);
printf("\nMatch at: %d\n", match);
getch();
}

int kmp(char* t, char* p)
{
int m = strlen(p);
int n = strlen(t);
int* f = init_array(m); // Failure function values.
int i = 0;
int j = 0;
while (i < n)
{
if (t[i] == p[j])
{
if (j == m - 1)
{
return i - j;
}
else
{
i += 1;
j += 1;
}
}
else
{
if (j > 0)
{
j = f[j-1];
}
else
{
i += 1;
}
}
}
return -1;
}
```

Implement a C program for the Knuth-Morris-Pratt Pattern matching algorithm.

Output:



Conclusion: Assuming the prior existence of the table T, the search portion of **kunth-morris-peratt algorithm has complexity o(n)** ,where n the length S.