Name: Vishal Shashikant Salvi.

**UID:** 2019230069

Class: SE Comps

Batch: C

#### **Experiment No 8**

Aim: To implement the KMP algorithm for string Matching

Theory:

## The Knuth-Morris-Pratt (KMP)Algorithm

Knuth-Morris and Pratt introduce a linear time algorithm for the string-matching problem. A matching time of O (n) is achieved by avoiding comparison with an element of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

# **Components of KMP Algorithm:**

- 1. The Prefix Function ( $\Pi$ ): The Prefix Function,  $\Pi$  for a pattern encapsulates knowledge about how the pattern matches against the shift of itself. This information can be used to avoid a useless shift of the pattern 'p.' In other words, this enables avoiding backtracking of the string 'S.'
- **2. The KMP Matcher:** With string 'S,' pattern 'p' and prefix function ' $\Pi$ ' as inputs, find the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrences are found.

### Algorithm:

```
Algorithm:
prefix table ()
    1. Stast
    2. length = 0 | 1 | 1 | 1 | 1 |
    3. Mps [0] +00 ; 9209001
     4, 1=1;
     5. while (i < 12)
    6. is pattern [i] = pattern [length] , then
             incoease length by 1
         1p3[i] 1= length
     8.
    9.
          else another of all
         if length to then
    10.
         east length = lps [length -1]
    11.
    12. else 9019
          1 ps ti] =0
    13.
         14+ 9000
    14"
         End
    15.
```

```
· kmp()
 4. Stast
 2. prefixtable ()
 3. while ix 11, do
 4. if pattern[i] == 8tr [i] ithen
         increase i and j by 1
 5
    if j==12 ithen 10 3/19/21 3
 7 point the location (i-j) as there
       is the pattern
         j = 1ps [j-1]
 9
          else if pattern [i] + strti7
 10
         if j to then
    deposit and = die poef ALPS [1-1]
11
 12
         else sola.
 13
            increase i by 1
         done
 14
 15
    End
```

Analysis:

· Analysis:
0 1 2 3 4 5 6 1 3 9 10 11 15
The time complexity of the above
algorithm is 0 (n+m).
petitos a b a d a d a
n = The length of text
m: The length of pattern.
slab 'x igod set studened teris the su
phododo arriba edt

Example:

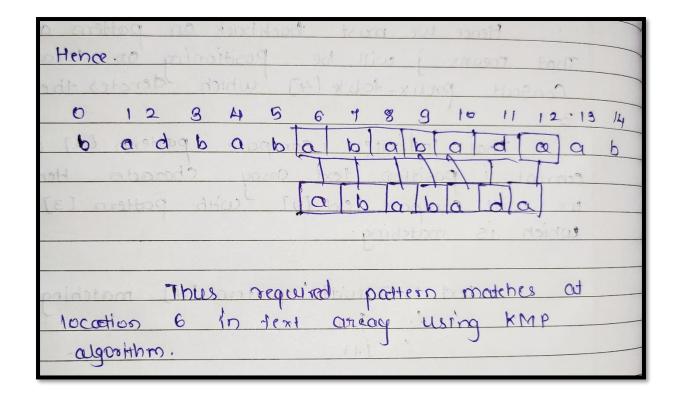
Francis I.	
Example:	
0 1 2 3 4 5 6 7 8 9 10 11 12	13 14
Text badbababada	
olgosition is a continued	
pattern a b a b a d a	
txat to deposit and a C	/
m. The length of pattern.	
we will first compute the prefix table for	) \$
the pattern ababada	
0 1 2 3 H 5 6	
a b a b a d a	
0 0 1 2 3 0 1	
0 1 2 3 4 5 6 7 8 9 10 11 12 13	14
Text badbababadaa	<b>b</b>
pattern a b a b a d a	
compairing b and a as it is not matchi	
we will compare text [1] with pattern [0]	ng
partition of	
0 1 2 8 4 5 6 7 8 9 10 11 12 13	14
Text babababada a	<b>b</b>
pattern o b a b a d a	

now compare Jesting with Pattern 10], we will
now compare Text [2] with pattern [1].
(1).
Debod pd pd pod e
O 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Text b 0 [d] b 0 b
Text boldbabababada ob
Potters a tol a live of the control of the frage of
Pattern a bababa a rangement
As text (2) is not matching with patter (1)
we will backtrack on pattern and compare
pattern [0] with Text [3].
Beacouse we consult prefix- table [1]
which is 0. I have a farmeter
Hence pattern (0) is compared with text(3).
text (7) matches with wathern [3]
0 1 2 8 4 5 6 7 8 9 10 11 12 13 14
rext badbababada a b
fel motted allow solution [8] type
pattern a bobada
partiti
of the principle of the post of the post of
Again Text (3) is not matching with pattern to).
we will then ask prefix-table [0] for the location
we will then ush politically and all the will be will be will be will be will be a second or the will be will
Ot baptens.
AS DRAIN FABRE (B) 18 0 ) SEC WITH COMPANY
pattern (0) with text [4].
or pattern (o) with text [4].  Pattern (o) with text [4].

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
both abababab
tent bodb. [a.   bababababab
pattern ajbabada
1 2 3 4 5 6 7 8 9 16 11 12 1
a a b a d a d a d a d bl a d d dat
Text (4) matches with pattern (0).
increment i and i da la a action
took to makes with proffer (1)
Text (5) matches with padtern (1)
ancrement i and jones and the
[8 tyst die To] miting
Text (6) matches with pattern [2]
Intrement i and i de de de de de
Hence pattern (o) is compared with
lext [7] matches with pattern [3]
Incrementa is and i
D D D D D D D D D D D D D D D D D D D
7ext [8] matches with pattern [4]
Increment i and j
1
But Text [9] is not matching with pattern [5]
dried ministrate and of letters with pattern (s)
10x1 had back at
11111
0 0 0 0 1
0 1 2 8 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 1 29 38 5 6

Hence we must bucktrack on pattern assay. That means j will be positioning on location 4.

Consult prefix-table [4] which denotes the value That indicates (compare patters [3] with corrent i position text array character. Hence we will compare text [9] with pattern [3]. which is matching. Text [10] with pattern [4], matching silli poith bat at a 1. 144 lext [11] with pattern (5] matching 1 , 1++ 1.11 Text [12] with pattern (6) matching . , 1++ : - j++ Thus we have reached on the last charactes of pattern, at the same time i'is positioned at tocotion 12 in the text array. The last charactes of patteren is also matching with Text [12] hence we can declare that a match of pattern is found in the text oraq at i - length of pattern +1. 1.e. 12-7+1 1.0.6.



#### Code:

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

char str[100],pattern[100];
int lps[100], locationArray[20]={[0 ... 19]=-1},ind;
int j=0,i=0,11,12;

void prefixtable()
{
    int len = 0, i;
    lps[0] = 0;
    i = 1;

    while(i < 12)
```

```
{
    if(pattern[i] == pattern[len])
         len++;
         lps[i] = len;
         i++;
}
else
         if( len != 0 )
     len = lps[len-1];
   else
         lps[i] = 0;
         i++;
     }
printf("\nLongest Prefix Sufix");
printf("\n");
printf("\backslash t-----\backslash n");
printf("\tPattern:|");
for(i=0;i<12;i++)
    printf(" %c |",pattern[i]);
```

```
printf("\n");
  printf("\t-----\n");
  printf("\tLPS: |");
  for(i=0;i<12;i++)
      printf(" %d |",lps[i]);
  printf("\n");
  printf("\backslash t-----\backslash n");
}
void KMP()
      int j=0,i=0;
      prefixtable();
      printf("\n");
      while(i<l1)
       if(pattern[j] == str[i])
        j++;
        i++;
       if(j==12)
```

```
locationArray[ind] = i-j;
          ind++;
        j = lps[j-1];
   }
   else if(pattern[j] != str[i])
        {
       if(j != 0)
         j = lps[j-1];
        }
       else
               i = i+1;
int main()
{
       printf("\nEnter the String:");
       scanf("%s",str);
       printf("\nEnter the Pattern:");
       scanf("%s",pattern);
       11=strlen(str);
       12=strlen(pattern);
```

```
KMP();
  if(locationArray[0]==-1)
{
 printf("\tPattern not found.\n");
 printf("\n***********\n");
}
else{
 for(int i = 0; i < ind; i++) {
  printf( "\tPattern found at index: %d \n",locationArray[i] );
  printf("\n***********\n");
return 0;
```

# **Output**:

Enter the String:vishalsalvi
Enter the Pattern:salvi
**************************************
Pattern:   s   a   l   v   i
LPS:   0   0   0   0
************
Pattern found at index: 6
*************

Enter th	ne String	): V	ish	als	sa.	lv:	Ĺ								
Enter th	ne Patter	n:	SSS	alv	7i										
****	*****	**	* * *	**	* * :	* * :	* * :	* * *	* * :	* * 1	**	* * *	* * *	* * *	****
Longest	Prefix S	uf	ix 												
	Pattern:	:	s   	s	I	5	I	a	I	1	I	v	I	i	I
	LPS:	(	)	1	I	2	I	0	I	0	I	0	I	0	I
****	*****	**	* * *	**	* * :	* * :	* * :	* * *	**	* * *	* * *	* * *	* * *	***	*****
	Pattern	not	t f	our	nd.	•									
****	*******	**	* * *	* * *	* * *	* * *	* * :	* * *	* * :	* * 1	* * *	* * *	* *	* * *	*****

Conclusion:
Thus, the basic idea behind this algorithm is to build a prefix array. This prefix array is built by using the prefix and suffix information of pattern. The overlapping prefix and suffix are used in LMP algorithm. The KMP algorithm achieves the efficiency of O(m+n) which is optimal in worst case.