**Batch: A1 Roll No.: 1911004**

**Experiment No. 10**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of Knuth Morris Pratt String Matching Algorithm** |

**Objective:** To learn the KMP String matching algorithm

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations. |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |
| CO 3 | Analyze and solve problems for   different string matching algorithms. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:** The [algorithm](https://en.m.wikipedia.org/wiki/Algorithm) was conceived by [James H. Morris](https://en.m.wikipedia.org/wiki/James_H._Morris) and independently discovered by [Donald Knuth](https://en.m.wikipedia.org/wiki/Donald_Knuth) "a few weeks later" from [automata theory](https://en.m.wikipedia.org/wiki/Automata_theory). Morris and [Vaughan Pratt](https://en.m.wikipedia.org/wiki/Vaughan_Pratt) published a technical report in 1970.

**New Concepts to be learned:**

The Knuth-Morris-Pratt (KMP)Algorithm:

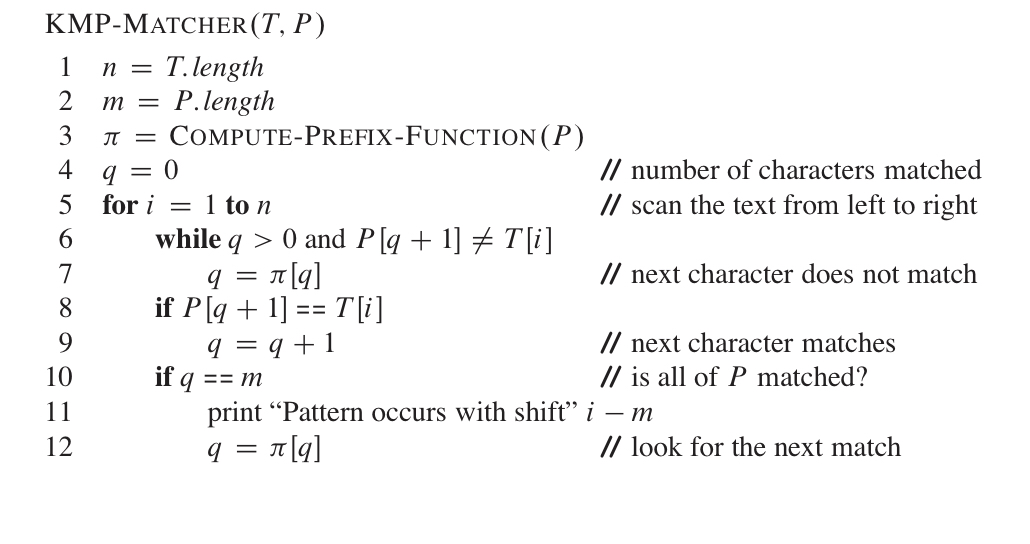
Knuth-Morris and Pratt introduce a linear time algorithm for the string-matching problem. A matching time of O(n) is achieved by avoiding comparison with an element of 'Σ' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string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

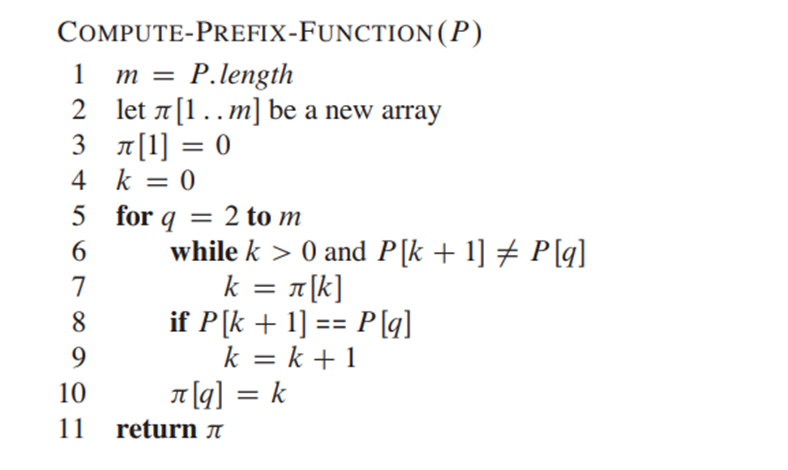
Components of KMP Algorithm:

**1. The Prefix Function (Π):** The Prefix Function, Π 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 'Σ.'

**2. The KMP Matcher:** With string 'S,' pattern 'p' and prefix function 'Π' as inputs, find the occurrence of 'p' in 'Σ' and returns the number of shifts of 'p' after which occurrences are found.

1. The most expensive part of the string-matching automaton method is to build the transition function , which takes O(m3| Σ |) time (or at least O(m| Σ |) time).
2. The KMP algorithm avoids to directly compute d. Instead, it computes an auxiliary function p[1..m] pre-computed from pattern P in O(m) time.
3. The transition function d can be obtained from array p in an efficient amortized constant time when the algorithm runs on a text.



****

**Implementation (Code):**

import java.util.\*;

class kmp {

public static int count = 0,p[],pos;

public static void main(String[] args) {

System.out.println("KMP String Matching");

Scanner ob = new Scanner(System.in);

String S = "",P = "";

System.out.print("Enter String = ");

S=ob.next();

System.out.print("Enter Pattern = ");

P=ob.next();

lps(P);

KMP(S,P);

System.out.println("The pattern occurs at index "+pos);

}

public static void lps(String P){

int[] lps = new int[P.length()];

lps[0] = 0;

for (int i = 1; i < P.length(); i++) {

int j = lps[i - 1];

while (j > 0 && P.charAt(i) != P.charAt(j))

j = lps[j - 1];

if (P.charAt(i) == P.charAt(j))

j++;

lps[i] = j;

}

for(int i=0;i<lps.length;i++){

System.out.print(" "+lps[i]);

}

System.out.println(" ");

}

public static void KMP(String S,String P){

if(P=="" ||P.length()<1)

System.out.println("Null Pattern");

if(S=="" ||S.length()<1)

System.out.println("Null String");

char C[] = P.toCharArray();

int i,j;

p=new int[P.length()+1];

p[0]=0;

for ( i = 1; i < P.length(); i++){

j = p[i + 1];

while (j > 0 && C[j] != C[i])

j = p[j];

if (j > 0 || C[j] == C[i])

p[i + 1] = j + 1;

}

for (i = 0, j = 0; i < S.length(); i++){

if (j < P.length() && S.charAt(i) == P.charAt(j)){

pos=i;

if (++j == P.length()){

pos=i-j+1;

break;

}

}

else if (j > 0){

j = p[j];

i--;

}

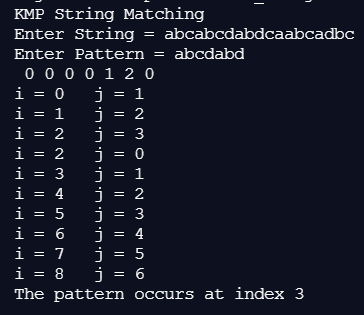
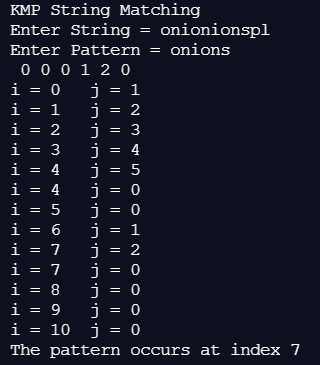
System.out.println("i = "+i+"\tj = "+j);

}

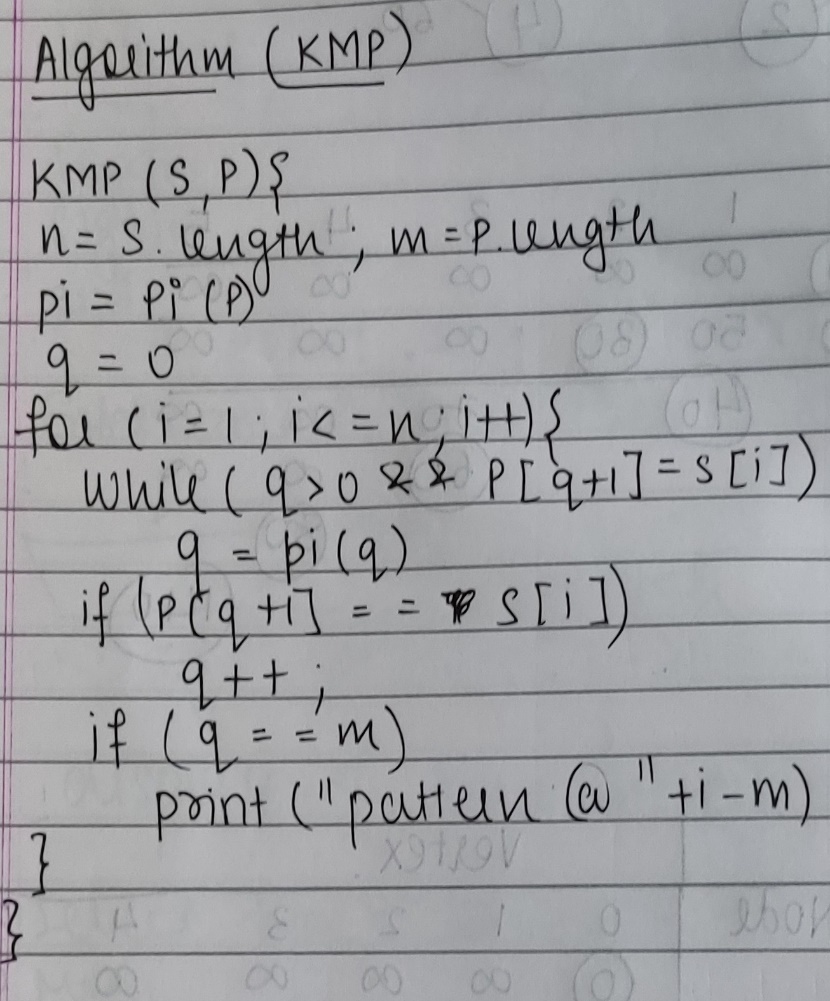
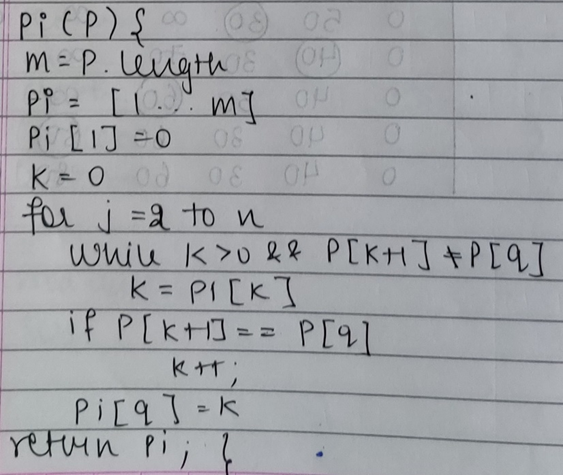
}

}

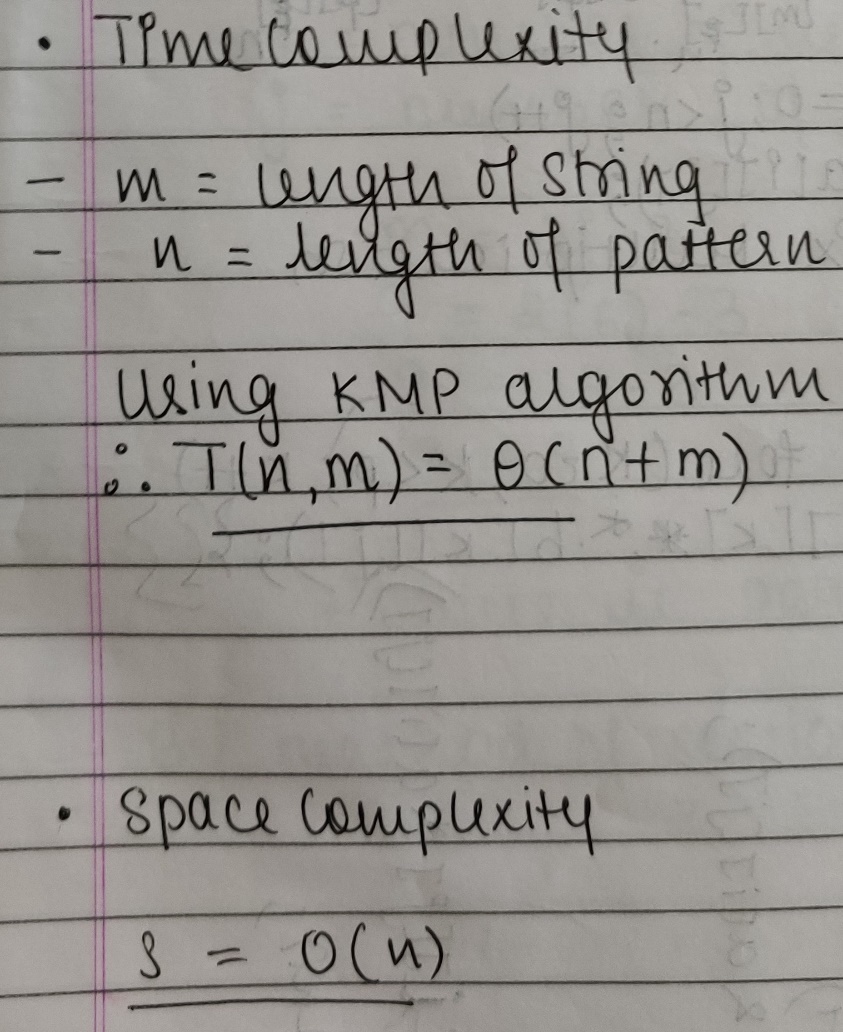
**OUTPUT:**

**Algorithm:**

 ****

**Analysis of KMP Algorithm:**



**CONCLUSION:**

We successfully understood and implemented the concept of Knuth Morris Pratt String Matching Algorithm; thus obtained the index of string where pattern is found in string in java.