**Pattern Searching Algorithms (Problem Type)**

Naive Pattern Searching (Strategy: Brute Force)

KMP Algorithm (Strategy: Finite Automata)

Rabin-Karp Algorithm (Strategy: Hashing)

**Naive Pattern Searching (Strategy: Brute Force)**

Algorithm

Step 1- sliding

Slide the pattern over the text.

Step 2

Match the occurrence

check if the characters matches with the pattern or not

if matches than

step 3

print the index value from where the pattern starts

step 4

repeat the process until the end of the text.

 Program:

package PatternSearchingAlgorithms;  
  
public class NaivePattern  
{  
 public static void npm(String t, String p)  
 {  
 int m = p.length();  
 int n = t.length();  
 for(int i=0;i<=n-m;i++) {  
 int j;  
 for (j = 0; j < m; j++)  
 {  
 if (t.charAt(i + j) != p.charAt(j))  
 break;  
 }  
  
 if (j == m)  
 System.*out*.println("Pattern mathced at index " + i);  
 }  
 }  
  
 public static void main(String args[])  
 {  
 String text = "programogramgramoprogram";  
 String pattern = "gram";  
 *npm*(text,pattern);  
 }  
}

**KMP Algorithm (knuth morris pratt) algorithm**

It is a pattern matching algorithm , It is  more  efficient than naïve pattern algorithm.

It  compare the pattern with the text window by sliding.

Program:

package PatternSearchingAlgorithms;  
  
public class KnuthMorrisPrattAlgorithm  
{  
 public static void kmp(String pa, String txt) {  
 int m = pa.length();  
 int n = txt.length();  
 int[] ps = new int[m];  
 *computeps*(pa,m,ps);  
 int i=0; // for text array.  
 int j=0; // for pattern array  
 while(i<n)  
 {  
 if(pa.charAt(j)==txt.charAt(i))  
 {  
 i++;  
 j++;  
 }  
 if(j==m)  
 {  
 System.*out*.println("Pattern matched at index: "+(i-j));  
 j= ps[j-1];  
  
 }  
 else if(i<n && pa.charAt(j)!=txt.charAt(i))  
 {  
 if(j!=0)  
 {  
 j= ps[j-1];  
 }  
 else {  
 i++;  
 }  
 }  
 }  
 }  
 public static void computeps(String pa,int m, int []ps)  
 {  
 int len =0;  
 ps[0]=0;  
 int i=1;  
 while(i<m)  
 {  
 if(pa.charAt(i)==pa.charAt(len))  
 {  
 len++;  
 ps[i]=len;  
 i++;  
 }  
 else {  
 if(len!=0)  
 {  
 len= ps[len-1];  
 }  
 else {  
 ps[i] =0;  
 i++;  
 }  
 }  
 }  
 }  
  
 public static void main(String args[])  
 {  
 String text = "aabasaaabbaddaaabffeaad";  
 String pattern = "aab";  
 *kmp*(pattern,text);  
 }  
}

**Rabin Karp Algorithm ( Uses Hashing)**

IT is a pattern searching algorithm which uses hashing.

In this if the hash value of the pattern is matched to the hash value of text than it go with the check of individual characters.

Steps----

1. Hashing - Calculate the hash values for both the text and the pattern.

Compare the hash values.

1. Sliding

Slide the pattern over the Text.

1. Hashing calculation

Choose a prime number as the modulus and a base b which is character set size that is 256.

Program:

package PatternSearchingAlgorithms;  
  
public class RabinKarpAlgorithm  
{  
 static final int *prime* =101;  
 public static void rka(String pat, String txt)  
 {  
 int m = pat.length();  
 int n = txt.length();  
 int patHash=0, txtHash =0;  
 for(int i=0;i<m;i++)  
 {  
 patHash = (patHash \* *prime* + pat.charAt(i))% *prime*;  
 txtHash = (patHash \* *prime* + txt.charAt(i))% *prime*;  
 }  
 for(int s=0;s<=n-m;s++) {  
 if (patHash == txtHash) {  
 int j;  
 for (j = 0; j < m; j++) {  
 if (txt.charAt(s + j) != pat.charAt(j)) {  
 break;  
 }  
 }  
 if (j == m)  
 System.*out*.println("Pattern matched at the index: " + s);  
 }  
 if (s < n - m)  
 {  
 txtHash =(txtHash - txt.charAt(s)\**pow*(*prime*, m-1))% *prime*;  
 txtHash = (txtHash \* *prime* + txt.charAt(s+m))% *prime*;  
 if(txtHash<0)  
 {  
 txtHash += *prime*; // txtHash =txtHash+prime;  
 }  
 }  
 }  
 }  
 static int pow(int base , int ex)  
 {  
 int r = 1;  
 while(ex>0)  
 {  
 if(ex % 2 == 1)  
 {  
 r = (r\* base)%*prime*;  
 }  
 base = (base \* base)%*prime*;  
 ex/=2 ; // ex = ex/2;  
 }  
 return r;  
 }  
 public static void main(String args[])  
 {  
 String text = "This is java world";  
 String pattern = "java";  
 *rka*(pattern,text);  
 }  
}