**Practical 8**

**Aim: - String Matching Algorithm**

|  |
| --- |
| **8.1 Naïve String Matching (Implement)**  **8.2 Rabin Karp (Implement)**  **8.1 Naïve String Matching (Implement)**  **Theory: -**  Naïve string matching algorithm simply test all the possible placement of Pattern P[1 . . m] relative to text T[1 . . n]. Specifically, we try shift s = 0, 1, . . . , n - m, successively and for each shift, s. Compare T[s +1 . . s + m] to P[1 . . m].  **Algorithm:**   1. n ← length [T] 2. m ← length [P] 3. for s ← 0 to n-m do 4. j ← 1 5. while j ≤ m and T[s + j] = P[j] do 6. j ← j +1 7. If j > m then 8. return valid shift s 9. return no valid shift exist     // i.e., there is no substring of T matching P.   **Program: -**  **Code: -**  #include<iostream>  #include<conio.h>  #include<string.h>  main()  {  char text[30],pat[30];  int i,j,m,n,l,cnt=0,cn=0;  cout<<"Enter your text:";  cin>>text;  cout<<"Enter your pattern:";  ci>>pat;  m=strlen(text);  n=strlen(pat);  for(i=0;i<=m-n;i++)  {  j=0;  while(j<=m && text[i+j]==pat[j])  {  j++;  }  if(j>=n)  {  l=i+n-1;  cn++;  cout<<"\n%d Match Found"<<cn;  cout<<"\nStarting index:%d"<<i;  cout<<"\nEnding index:%d"<<l;  cnt++;  }  }  if(cnt==0)  {  cout<<"Match not found"<<endl;  }  return 0;  }  **Output: -**  G:\Screenshot (65).png  **8.2 Rabin Karp (Implement)**  **Theory: -**  The Rabin–Karp algorithm or Karp–Rabin algorithm is a string searching algorithm created by Richard M. Karp and Michael O. Rabin (1987) that uses hashing to find any one of a set of pattern strings in a text. For text of length n and p patterns of combined length m, its average and best case running time is O (n+m) in space O(p), but its worst-case time is O(nm).  **Algorithm:**  Function RabinKarp(string s[1..n], string pattern[1..m])   1. hpattern := hash(pattern[1..m]); 2. for i from 1 to n-m+1 3. hs := hash(s[i..i+m-1]) 4. if hs = hpattern 5. if s[i..i+m-1] = pattern[1..m] 6. return i   **Program: -**  **Code: -**  import java.io.BufferedReader;  import java.io.InputStreamReader;  import java.io.IOException;  import java.util.Random;  import java.math.BigInteger;  public class RabinKarp  {  private String pat;  private long patHash;  private int M;  private long Q;  private int R;  private long RM;  public RabinKarp(String txt, String pat)  {  this.pat = pat;  R = 256;  M = pat.length();  Q = longRandomPrime();  RM = 1;  for (int i = 1; i <= M-1; i++)  RM = (R \* RM) % Q;  patHash = hash(pat, M);  int pos = search(txt);  if (pos == -1)  System.out.println("\nNo Match\n");  else  System.out.println("Pattern found at position : "+ pos);  }  private long hash(String key, int M)  {  long h = 0;  for (int j = 0; j < M; j++)  h = (R \* h + key.charAt(j)) % Q;  return h;  }  private boolean check(String txt, int i)  {  for (int j = 0; j < M; j++)  if (pat.charAt(j) != txt.charAt(i + j))  return false;  return true;  }  private int search(String txt)  {  int N = txt.length();  if (N < M) return N;  long txtHash = hash(txt, M);  if ((patHash == txtHash) && check(txt, 0))  return 0;  for (int i = M; i < N; i++)  {  txtHash = (txtHash + Q - RM \* txt.charAt(i - M) % Q) % Q;  txtHash = (txtHash \* R + txt.charAt(i)) % Q;  int offset = i - M + 1;  if ((patHash == txtHash) && check(txt, offset))  return offset;  }  return -1;  }  private static long longRandomPrime()  {  BigInteger prime = BigInteger.probablePrime(31, new Random());  return prime.longValue();  }  public static void main(String[] args) throws IOException  {  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  System.out.println("Rabin Karp Algorithm Test\n");  System.out.println("\nEnter Text\n");  String text = br.readLine();  System.out.println("\nEnter Pattern\n");  String pattern = br.readLine();  System.out.println("\nResults : \n");  RabinKarp rk = new RabinKarp(text, pattern);  }  }  **Output: -** |

**About Complexities:**

|  |  |
| --- | --- |
| **Algorithm** | **Time complexity** |
| Naïve String Matching Algorithm: | O( (n-m+1)m) |
| Rabin Karp Algorithm | O( (n-m+1)m) |

**Conclusion: -**

From this practical we learnt about string matching algorithms and concluded that Rabin Karp is more efficient in most of the cases. A practical application of the algorithm is detecting plagiarism. Given source material, the algorithm can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Because of the abundance of the sought strings, single-string searching algorithms are impractical