|  |  |
| --- | --- |
| **NAME:** | Vaishnavi Borkar |
| **UID:** | 2021300016 |
| **BATCH:** | A |
| **EXP NO:** | 10 |
| **AIM:** | To implement Rabin Karp and Naive String Matching Algorithm |
| **THEORY:** | The Naive String Matching algorithm slides the pattern one by one. After each slide, one by one checks characters at the current shift, and if all characters match then print the match.  Like the Naive Algorithm, the Rabin-Karp algorithm also slides the pattern one by one. But unlike the Naive algorithm, the Rabin Karp algorithm matches the hash value of the pattern with the hash value of the current substring of text, and if the hash values match then only it starts matching individual characters. So Rabin Karp algorithm needs to calculate hash values for the following strings.  Pattern itself  All the substrings of the text of length m  **Algorithm:**  NaiveString  Match(T, P)  1. n ← length[T]  2. m ← length[P]  3. for i ← 0 to n - m  4. j ← 0  5. while j < m and P[j] = T[i+j]  6. j ← j + 1  7. if j = m  8. print "Pattern occurs with shift" i  1. RabinKarp  Match  (text, pattern):  2. n = length(text)  3. m = length(pattern)  4. pattern\_hash = hash(pattern)  5. for i from 0 to n-m:  6. text\_hash = hash(text[i:i+m])  7. if pattern\_hash == text\_hash:  8. if pattern == text[i:i+m]:  10. return i  11. return -1 |
| **PROGRAM:**  **(Rabin-Karp)**  **RESULT:** | *#include* <stdio.h>  *#include* <string.h>  int rabinSearch(char \**t*, char \**p*)  {      int tlength = strlen(*t*);      int plength = strlen(*p*);      int i, j;      int phash = 0;      int thash = 0;      int h = 1;  *for* (i = 0; i < plength - 1; i++)      {          h = (h \* d) % q;      }  *for* (i = 0; i < plength; i++)      {          phash = (d \* phash + *p*[i]) % q;          thash = (d \* thash + *t*[i]) % q;      }  *for* (i = 0; i <= tlength - plength; i++)      {  *if* (thash == phash)          {  *for* (j = 0; j < plength; j++)              {  *if* (*t*[i + j] != *p*[j])                  {  *break*;                  }              }  *if* (j == plength)              {  *return* i;              }          }  *if* (i < tlength - plength)          {              thash = (d \* (thash - *t*[i] \* h) + *t*[i + plength]) % q;  *if* (thash < 0)              {                  thash += q;              }          }      }  *return* -1;  }  int main()  {      char t[1000], p[1000];      printf("Enter the text: ");      fgets(t, 1000, stdin);      printf("Enter the pattern to search for: ");      fgets(p, 1000, stdin);      t[strcspn(t, "\n")] = 0;      p[strcspn(p, "\n")] = 0;      int result = rabinSearch(t, p);  *if* (result == -1)      {          printf("pattern not found in text.\n");      }  *else*      {          printf("Pattern found in text starting at index %d.\n", result);      }  *return* 0;  }  *#include* <stdio.h>  *#include* <conio.h>  *#include* <string.h>  int match(char *st*[100], char *pat*[100]);  int main(int *argc*, char \*\**argv*)  {      char st[100], pat[100];      int status;      printf("\*\*\* Naive String Matching Algorithm \*\*\*\n");      printf("Enter the String.\n");      gets(st);      printf("Enter the pattern to match.\n");      gets(pat);      status = match(st, pat);  *if* (status == -1)          printf("\nNo match found");  *else*          printf("Match has been found on %d position.", status);  *return* 0;  }  int match(char *st*[100], char *pat*[100])  {      int n, m, i, j, count = 0, temp = 0;      n = strlen(*st*);      m = strlen(*pat*);  *for* (i = 0; i <= n - m; i++)      {          temp++;  *for* (j = 0; j < m; j++)          {  *if* (*st*[i + j] == *pat*[j])                  count++;          }  *if* (count == m)  *return* temp;          count = 0;      }  *return* -1;  } |
| **PROGRAM:**  **(Naive approach)** | *#include* <stdio.h>  *#include* <conio.h>  *#include* <string.h>  int match(char *st*[100], char *pat*[100]);  int main(int *argc*, char \*\**argv*)  {      char st[100], pat[100];      int status;      printf("\*\*\* Naive String Matching Algorithm \*\*\*\n");      printf("Enter the String.\n");      gets(st);      printf("Enter the pattern to match.\n");      gets(pat);      status = match(st, pat);  *if* (status == -1)          printf("\nNo match found");  *else*          printf("Match has been found on %d position.", status);  *return* 0;  }  int match(char *st*[100], char *pat*[100])  {      int n, m, i, j, count = 0, temp = 0;      n = strlen(*st*);      m = strlen(*pat*);  *for* (i = 0; i <= n - m; i++)      {          temp++;  *for* (j = 0; j < m; j++)          {  *if* (*st*[i + j] == *pat*[j])                  count++;          }  *if* (count == m)  *return* temp;          count = 0;      }  *return* -1;  } |
| **RESULT:** |  |

|  |  |
| --- | --- |
| **CONCLUSION:** | From this experiment, I understood about Rabin Karp method for string matching which uses hash value to match the string and reduce the time complexity as compared to the naive method. |