|  |  |
| --- | --- |
| **Experiment No.** | **10** |
| **Aim** | **Experiment on string matching algorithms** |
| **Name** | **Rucha Sudhir Kulkarni** |
| **UID No.** | **2021300067** |
| **Class & Division** | **SE Computer Engineering (Div:A)(Batch:D)** |
| **Date of Performance:** | **19.04.2023** |
| **Date of Submission:** | **24.04.2023** |

**Aim: – To implement Rabin Karp and Naive String Matching Algorithms.**

**Algorithm:**

**Rabin Karp:**

1. Start
2. Input Text string T[] and Pattern String P[]
3. Initially calculate the hash value of the pattern and text string.
   1. for (i = 0; i < m; i++)
   2. p = ( p + P[i]) % q;
   3. t = ( t + T[i]) % q;
4. Start iterating from the starting of the string:
   1. Calculate the hash value of the current substring having length m.
   2. If the hash value of the current substring and the pattern are same check if the substring is same as the pattern.

if (p == t) {

sphits++;

for (j = 0; j < m; j++) {

if (T[i + j] != P[j])

break;

}

* 1. If they are same, store the starting index as a valid answer. Otherwise, continue for the next substrings.

1. Return the starting indices as the required answer.
2. Stop

Naïve:

1. Start
2. Input Text string T[] and Pattern String P[]
3. Start iterating from the starting of the string:
   1. Slide the pattern over text one by one and check for a match. If a match is found, then slide by 1 again to check for subsequent matches.
   2. if (p == t) {

for (j = 0; j < m; j++) {

if (T[i + j] != P[j])

break;

}

* 1. If they are same, store the starting index as a valid answer. Otherwise, continue for the next substrings.

1. Return the starting indices as the required answer.
2. Stop

**Program:**

**Rabin Karp program:**

#include<stdio.h>

#include<string.h>

int sphits=0;

int validhits=0;

void Rabin(char T[],char P[],int q){

int m = strlen(P);

  int n = strlen(T);

  int i, j;

  int p = 0;

  int t = 0;

  int h = 1;

  for (i = 0; i < m - 1; i++)

    h = (h) % q;

  for (i = 0; i < m; i++) {

    p = ( p + P[i]) % q;

    t = ( t + T[i]) % q;

  }

  for (i = 0; i <= n - m; i++) {

    if (p == t) {

        sphits++;

      for (j = 0; j < m; j++) {

        if (T[i + j] != P[j])

          break;

      }

      if (j == m)

      {

        validhits++;

        if(validhits==1)

          printf("Pattern is found at position:  %d ", i + 1);

          else{

            printf(", %d ", i + 1);

          }

      }

    }

    if (i < n - m) {

      t = ( (t - T[i] \* h) + T[i + m]) % q;

      if (t < 0)

      {

         t = (t + q);

      }

    }

  }

}

int main(){

int q=13;

//char T[]="my name is rucha";

//char P[]="rucha";

char T[20],P[20];

printf("Enter text string\n");

scanf("%[^\n]%\*c",T);

printf("Enter pattern string to be found\n");

scanf("%[^\n]%\*c",P);

Rabin(T,P,q);

printf("\nValid hits are: %d",validhits);

printf("\nSpurious hits are: %d",sphits-validhits);

return 0;

}

**Naive String Matching Algorithm Program:**

#include<stdio.h>

#include<string.h>

void Naive(char\* T,char\* P){

int n=strlen(T);

int m=strlen(P);

for(int i=0;i<=n-m;i++){

for (int j=0; j<m; j++) {

          if(j==m-1){

          printf("Pattern occurs after %d shifts\n",i);

          }

          if (T[i + j] != P[j])

          break;

      }

}

}

int main(){

//char T[]="my name is rucha";

//char P[]="rucha";

char T[20],P[20];

printf("Enter text string\n");

scanf("%[^\n]%\*c",T);

printf("Enter pattern string to be found\n");

scanf("%s",P);

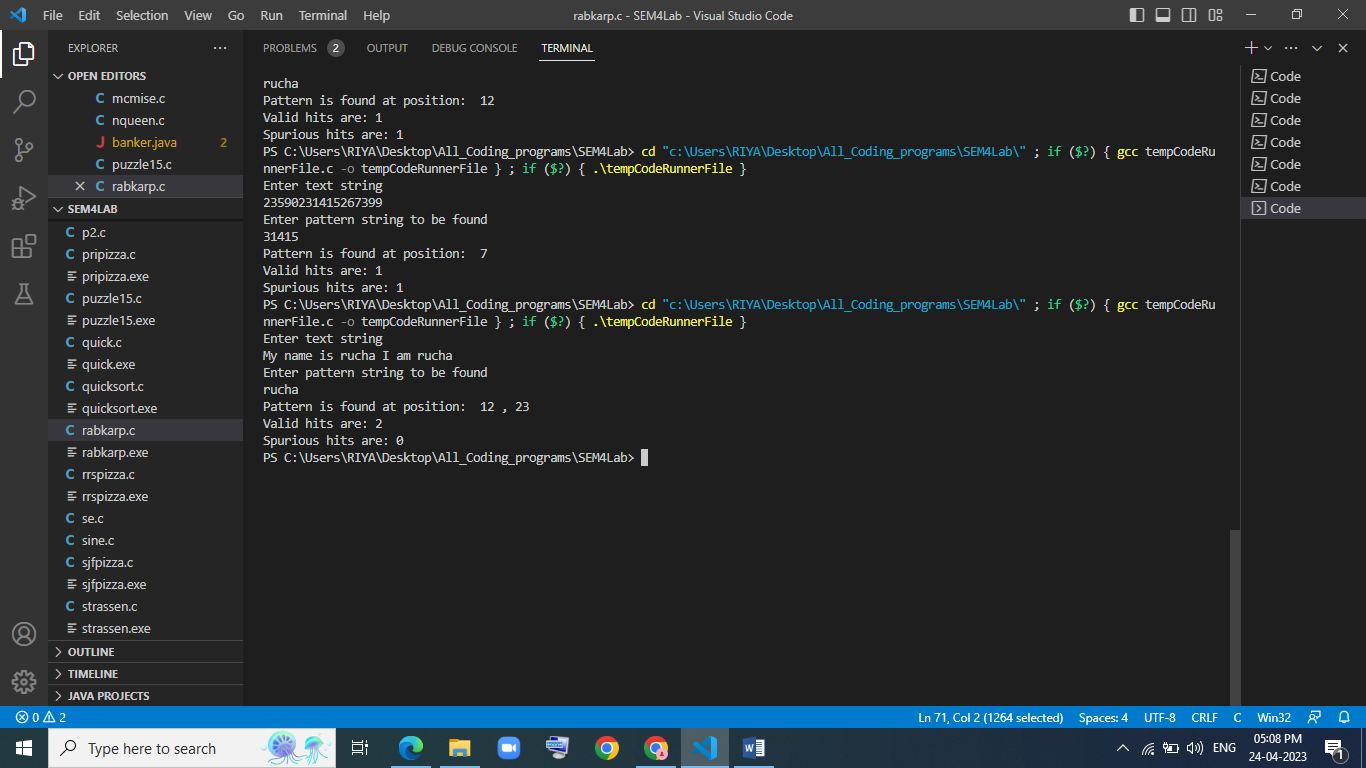
Naive(T,P);

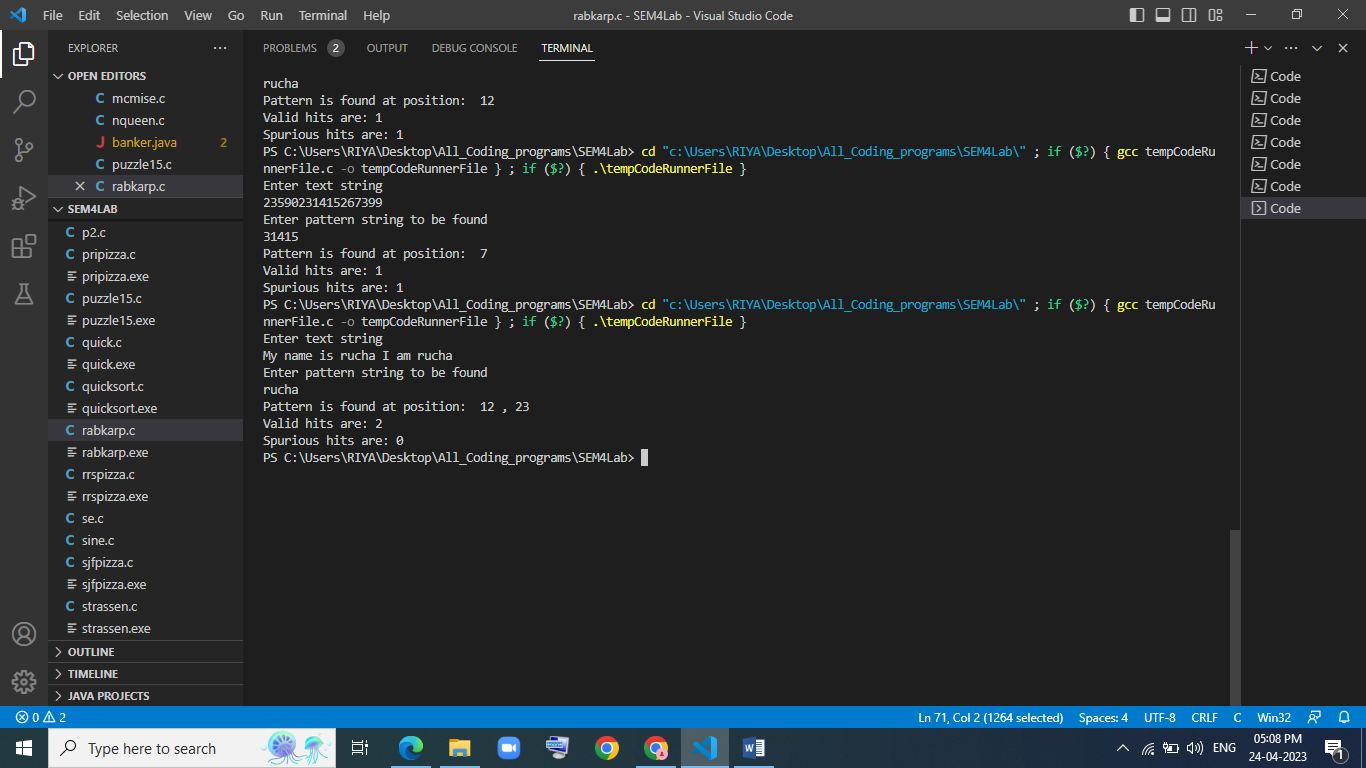
return 0;

}

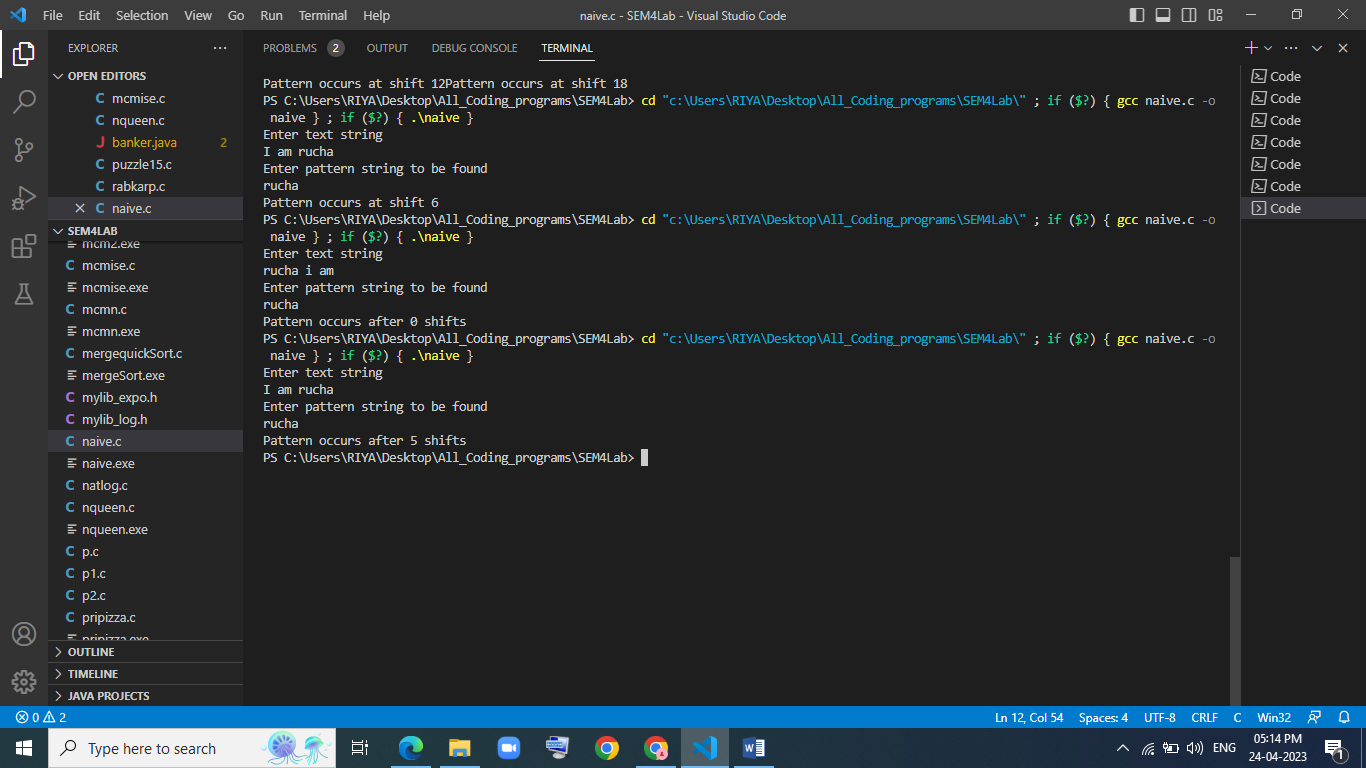
**Output and Observation:**

**Rabin Karp output:**





**Naive String Matching Algorithm output:**



**Conclusion:**

After performing the above experiment, I got to know two string matching algorithms Rabin Karp and Naïve String Matching Algorithm and implement program for them in c.