NAME:	SANIYA BANGARE
UID:	2021300009
BATCH:	A
EXP NO:	9
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:

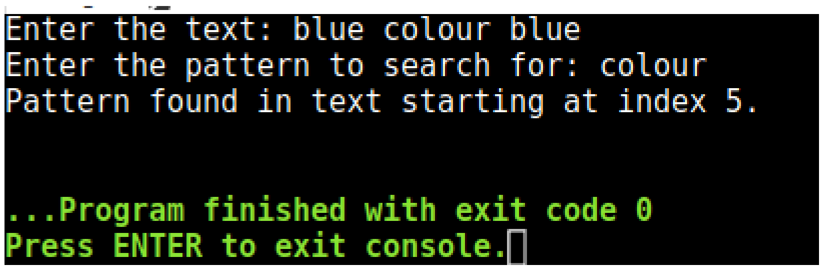
```
 NaiveString
Match(T, P)
 1. n ← length[T]
2. m ← length[P]
 3. for i ← 0 to n - m
4. j ← 0
5. while j < m and P[j] = T[i+j]
6. j ← j + 1
7. if j = m
```

	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)	see code below

```
#include <stdio.h>
#include <string.h>

#define d 256
#define q 101

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;
}
```

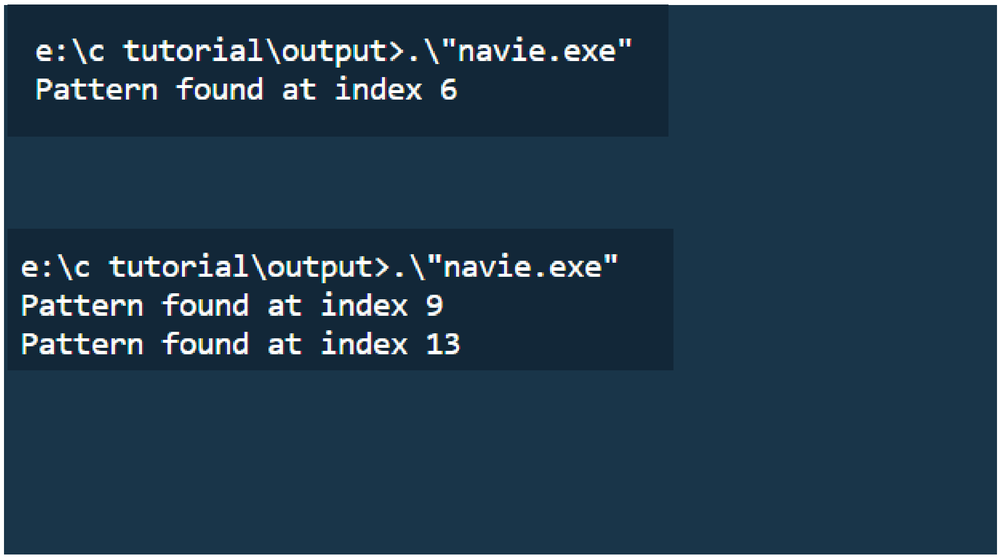
RESULT:



```
Enter the text: blue colour blue
Enter the pattern to search for: colour
Pattern found in text starting at index 5.


...Program finished with exit code 0
Press ENTER to exit console.
```

PROGRAM:
(Naive approach)

```c
#include <stdio.h>
#include <string.h>

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++) {
        int j;

        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j == M)
            printf("Pattern found at index %d \n", i);
    }
}

int main()
{
    char txt[] = "BAAAACAADAABAAABAA";
    char pat[] = "AABA";

    search(pat, txt);
    return 0;
}


//
int main()
{
    char txt[] = "MYNAMEISSANIYASANI";
    char pat[] = "ISSANI";

    search(pat, txt);
    return 0;
}
```

RESULT:	

```
e:\c tutorial\output>.\"navie.exe"
Pattern found at index 6
```

```
e:\c tutorial\output>.\"navie.exe"
Pattern found at index 9
Pattern found at index 13
```

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.