NAME:	Arnav Hoskote
UID:	2021300044
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	10
DATE OF PERFORMANCE	17/03/2023
DATE OF SUBMISSION	23/04/2023
AIM:	To use Rabin-Karp and Naïve string matching algorithms to match strings.
PROBLEM STATEMENT 1:	**Rabin-Karp and Naïve method to match strings**
ALGORITHM and THEORY:	Hash at the next shift must be efficiently computable from the current hash value and next character in text or we can say hash(txt[s+1 .. s+m]) must be efficiently computable from hash(txt[s .. s+m-1]) and txt[s+m] i.e., hash(txt[s+1 .. s+m]) = rehash(txt[s+m], hash(txt[s .. s+m-1])) and Rehash must be O(1) operation. he number of possible characters is higher than 10 (256 in general) and the pattern length can be large. So the numeric values cannot be practically stored as an integer. Therefore, the numeric value is calculated using modular arithmetic to make sure that the hash values can be stored in an integer variable (can fit in memory words). To do rehashing, we need to take off the most significant digit and add the new least significant digit for in hash value. Rehashing is done using the following formula:

hash(txt[s+1 .. s+m]) = (d (hash(txt[s .. s+m-1]) – txt[s]*h) + txt[s + m]) mod q
hash(txt[s .. s+m-1]) : Hash value at shift s
hash(txt[s+1 .. s+m]) : Hash value at next shift (or shift s+1)
d: Number of characters in the alphabet
q: A prime number
h: d(m-1)

PROGRAM:

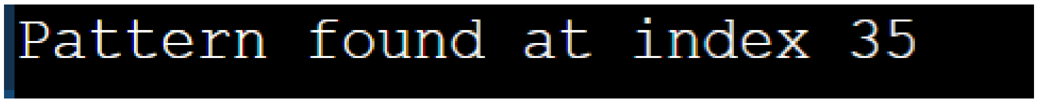
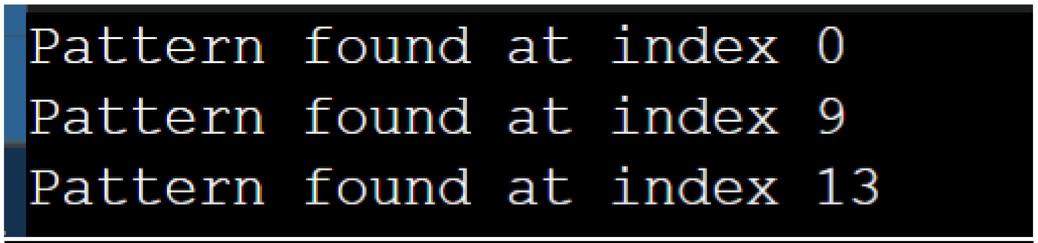
<u>RABIN-KARP:</u>

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

#define d 256

/* pat -> pattern
        txt -> text
        q -> A prime number
*/
void search(char pat[], char txt[], int q)
{
        int M = strlen(pat);
        int N = strlen(txt);
        int i, j;
        int p = 0;
        int t = 0;
        int h = 1;

        for (i = 0; i < M - 1; i++)
                h = (h * d) % q;


        for (i = 0; i < M; i++) {
                p = (d * p + pat[i]) % q;
                t = (d * t + txt[i]) % q;
        }


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

                if (p == t) {

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


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

                if (i < N - M) {
                        t = (d * (t - txt[i] * h) + txt[i + M]) % q;


                        if (t < 0)
                                t = (t + q);
                }
        }
}

int main()
{
        char txt[] = "MY NAME IS ARNAV AND I LIKE PLAYING
TABLA";
        char pat[] = " TABLA";
```

```
	int q = 17;
	search(pat, txt, q);
	return 0;
}
```

NAÏVE:

```
#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[] = "AABAACAADAABAAABAA";
```

	`char pat[] = "AABA";` `search(pat, txt);` `return 0;` `}`
OUTPUT:	<u>RABIN-KARP:</u>  `Pattern found at index 35` <u>NAÏVE:</u>  `Pattern found at index 0` `Pattern found at index 9` `Pattern found at index 13`
CONCLUSION:	By performing above experiment I have understood Rabin Karp and naïve string-matching algorithms and its uses thoroughly.