**Artificial Intelligence and Data Science Department.**
AOA / Even Sem 2021-22 / Experiment 10.

YASH SARANG.
47 / D6AD.
EXPERIMENT - 10.

---

**Aim:** Write a program String matching using Rabin Karp Algorithm

---

**THEORY:**
<u>Rabin Karp Algorithm</u>
Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m.
Input:
txt[] = "THIS IS A TEST TEXT"
pat[] = "TEST"

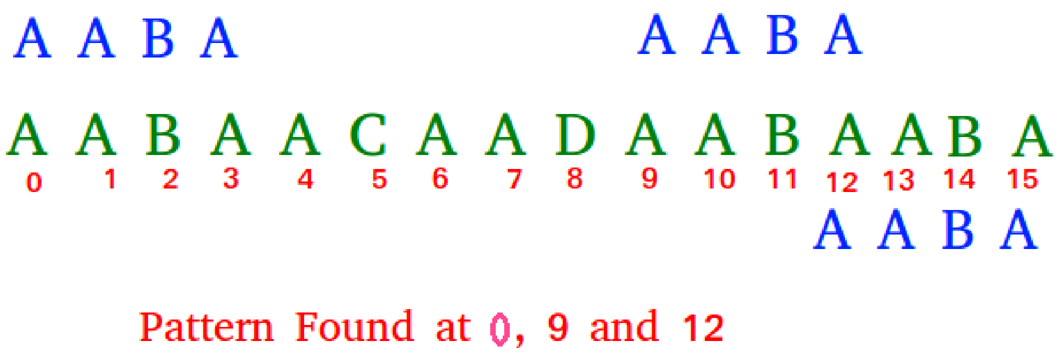
Output: Pattern found at index 10

Input:
txt[] = "AABAACAADAABAABA"
pat[] = "AABA"

```
Output:
Pattern found at index 0
Pattern found at index 9
Pattern found at index 12
```

Text : A A B A A C A A D A A B A A B A

Pattern : A A B A



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.

1) Pattern itself.

2) All the substrings of the text of length m.

Since we need to efficiently calculate hash values for all the substrings of size m of text, we must have a hash function that has the following property.

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.

The hash function suggested by Rabin and Karp calculates an integer value. The integer value for a string is the numeric value of a string.
For example, if all possible characters are from 1 to 10, the numeric value of "122" will be 122. The 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)

**Time Complexity:** Average : O(n+m)
Worst Case : O(nm)

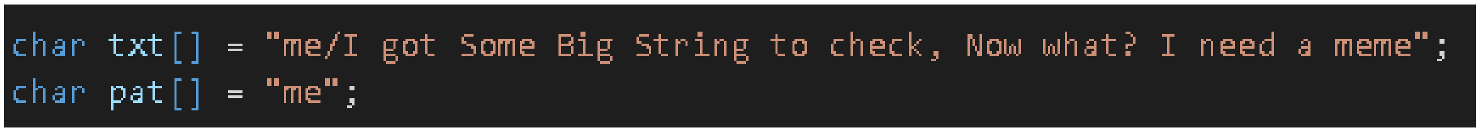
---

## CODE:

Code is in the Rabin_Karp.c file attached along with this doc.

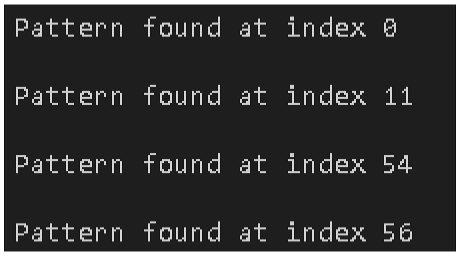
---

## INPUT:



```
char txt[] = "me/I got Some Big String to check, Now what? I need a meme";
char pat[] = "me";
```

## OUTPUT:



```
Pattern found at index 0

Pattern found at index 11

Pattern found at index 54

Pattern found at index 56
```

---

## CONCLUSION:

By performing this experiment, I can conclude that Although the worst-case time complexity of Rabin-Karp is O(nm) which is almost the same as the Naive method, the average time complexity O(n+m) provides a significant difference in time.

The average and best-case running time of the Rabin-Karp algorithm is O(n+m), but its worst-case time is O(nm).
The worst case of the Rabin-Karp algorithm occurs when all characters of pattern and text are the same as the hash values of all the substrings of txt[] match with the hash value of pat[]. For example pat[] = “AAA” and txt[] = “AAAAAAA”.

---