# SCTR's Pune Institute of Computer Technology Dhankawadi, Pune

**A PROJECT REPORT ON**

- Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching. Observe the difference in the working of both the algorithms for the same input.

# SUBMITTED BY

41121

Manas Milind Deshpande

# Under the guidance of

Prof. U. S. Pawar



DEPARTMENT OF COMPUTER ENGINEERING

Academic Year 2023-24



# DEPARTMENT OF COMPUTER ENGINEERING

**SCTR's Pune Institute of Computer Technology Dhankawadi, Pune**

# Maharashtra 411043

**CERTIFICATE**

This is to certify that the SPPU Curriculum-based Mini Project

- Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching. Observe difference in working of both the algorithms for the same input.

# Submitted by

Manas Deshpande

41121

has satisfactorily completed the curriculum-based Mini Project under the guidance of Prof. U. S. Pawar towards the partial fulfillment of the final year

Computer Engineering Semester VII,

Academic Year 2023-24 of Savitribai Phule Pune University.

# Date:

**Place:** PUNE **Name & Sign of Project**

**Guide:**

# Acknowledgment

It gives me great pleasure to present the mini project on - Implementing the Naive string matching algorithm and the Rabin-Karp algorithm for string matching. Observe the difference in the working of both the algorithms for the same input.

First of all, I would like to take this opportunity to thank my guide Prof. U. S. Pawar for giving me all the help and guidance needed. I am grateful for his kind support and valuable suggestions that proved to be beneficial in the overall completion of this project.

I am thankful to our Head of the Computer Engineering Department, Dr. G. V. Kale, for her indispensable support and suggestions throughout the internship work. I would also genuinely like to express my gratitude to the CC, Prof. Samadhan Jadhav, for his constant guidance.

Finally, I would like to thank my mentor, Prof. U. S. Pawar for his constant help and support during the overall process.

# Title:

- Implement the Naive string matching algorithm and Rabin-Karp algorithm for

string matching. Observe the difference in the working of both the algorithms for the same input.

# Problem Statment:

Develop and implement the Naive string matching algorithm and the Rabin-Karp algorithm for string matching. Analyze and compare the performance and efficiency of both algorithms when applied to the same input strings.

# Objective:

Implement the Naive string-matching algorithm.

Implement the Rabin-Karp algorithm for string matching. Select a set of input strings for testing and evaluation.

Ensure the input data includes a mix of short and long strings

Apply the Naive algorithm to the input strings and record the matching results. Apply the Rabin-Karp algorithm to the same input strings and record the matching results.

# Algorithm :

Naive String Matching Algorithm:

The Naive string matching algorithm is a simple and straightforward approach to finding occurrences of a pattern (substring) within a larger text (string). It works by systematically comparing the characters of the pattern with the characters in the text.

Here's the basic algorithm:

* Start by aligning the pattern's leftmost character with the text's leftmost character. Compare each character of the pattern with the corresponding character in the text. If a mismatch is found at any position, shift the pattern one position to the right and repeat the comparison.
* Continue this process until either the pattern is found completely in the text (a match is found), or the end of the text is reached without finding a match.
* If a match is found, record the starting position of the match in the text.
* Repeat the process for all possible starting positions in the text until the entire text is searched.

Rabin-Karp Algorithm:

The Rabin-Karp algorithm is a more efficient string-matching algorithm that utilizes hashing to expedite the search process. It is particularly useful when you need to search for a pattern in multiple texts or for multiple patterns in a single text.

Here's the basic algorithm:

* Compute a hash value (a numerical representation) for the pattern and for the initial substring of the same length in the text.
* Compare the hash values. If they match, it suggests a potential match.
* If the hash values match, perform a character-by-character comparison to confirm the match. If they don't match, proceed to the next substring in the text.
* If a mismatch occurs, update the hash value for the next substring by subtracting the contribution of the first character in the previous substring and adding the contribution of the next character in the current substring.
* Repeat steps 2-4 until a match is found or until you reach the end of the text. If a match is found, record the starting position of the match in the text.
* Repeat the process for all possible starting positions in the text until the entire text is searched.

The Rabin-Karp algorithm is efficient because it reduces the number of character comparisons needed, especially when the hash values of substrings do not match. However, it requires careful handling of hash collisions and the choice of a suitable hash function.

# Input :

# Naive Pattern Searching algorithm = O(N^2)

def search(pat, txt):

    M = len(pat)

    N = len(txt)

    for i in range(N - M + 1):

        j = 0

        while (j < M):

            if (txt[i + j] != pat[j]):

                break

            j += 1

        if (j == M):

            print("Pattern found at index ", i)

txt = "AABAACAADAABAAABAA"

pat = "AABA"

search(pat, txt)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

######################################################

# Rabin Karp Algorithm given in CLRS book  => O(m+n)

# d is the number of characters in the input alphabet

d = 256

# pat -> pattern

# txt -> text

# q -> A prime number

def search1(pat, txt, q):

    M = len(pat)

    N = len(txt)

    i = 0

    j = 0

    p = 0  # hash value for pattern

    t = 0  # hash value for txt

    h = 1

    for i in range(M-1):

        h = (h\*d) % q

    for i in range(M):

        p = (d\*p + ord(pat[i])) % q

        t = (d\*t + ord(txt[i])) % q

    for i in range(N-M+1):

        if p == t:

            # Check for characters one by one

            for j in range(M):

                if txt[i+j] != pat[j]:

                    break

                else:

                    j += 1

            # if p == t and pat[0...M-1] = txt[i, i+1, ...i+M-1]

            if j == M:

                print("Pattern found at index " + str(i))

        if i < N-M:

            t = (d\*(t-ord(txt[i])\*h) + ord(txt[i+M])) % q

            if t < 0:

                t = t+q

txt = "AABAACAADAABAAABAA"

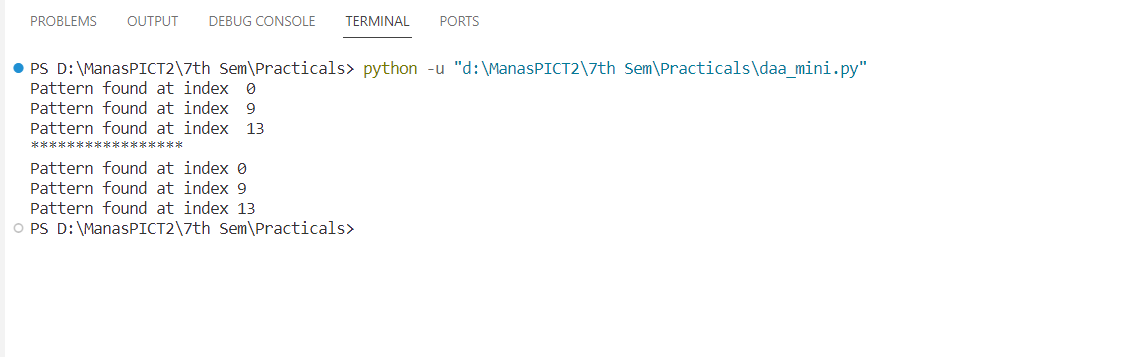
pat = "AABA"

# A prime number

q = 101

search1(pat, txt, q)

# Results:

****

**Conclusion :**

In this study, we implemented and compared the Naive string-matching algorithm and the Rabin-Karp algorithm for string matching.