|  |  |
| --- | --- |
| **Name** | PARTH GANDHI |
| **UID No.** | 2021300033 |
| **Class & Division** | SE CE A (Batch: B) |
| **Experiment No.** | 9 |

**Aim**: To implement String Matching Algorithm (Rabin- Karp and Naïve)

**Algorithm:**

1. Rabin Karp

RABIN-KARP-MATCHER (T, P, d, q)

1. n ← length [T]

2. m ← length [P]

3. h ← dm-1 mod q

4. p ← 0

5. t0 ← 0

6. for i ← 1 to m

7. do p ← (dp + P[i]) mod q

8. t0 ← (dt0+T [i]) mod q

9. for s ← 0 to n-m

10. do if p = ts

11. then if P [1.....m] = T [s+1.....s + m]

12. then "Pattern occurs with shift" s

13. If s < n-m

14. then ts+1 ← (d (ts-T [s+1]h)+T [s+m+1])mod q

1. Naïve

NAIVE-STRING-MATCHER (T, P)

1. n ← length [T]

2. m ← length [P]

3. for s ← 0 to n -m

4. do if P [1.....m] = T [s + 1....s + m]

5. then print "Pattern occurs with shift" s

**Code:**

**Naïve Approach:**

#include <bits/stdc++.h>

using namespace std;

void naiveSearch(string pattern, string text)

{

    int patternLength = pattern.length();

    int textLength = text.length();

    int i, j;

    for (i = 0; i <= textLength - patternLength; i++) {

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

            if (text[i + j] != pattern[j])

                break;

        }

        if (j == patternLength)

            cout << "\nPattern found at index " << i << endl;

    }

}

int main()

{

    string text;

    string pattern;

   cout<<"\nEnter the string :";

   getline(cin, text);

   cout<<"\nEnter the pattern you want to search :";

   getline(cin, pattern);

    naiveSearch(pattern, text);

    cout<<endl;

    return 0;

}

**Rabin Karp:**

#include <stdio.h>

#include <string.h>

#define d 256  // Number of possible characters in the input alphabet

#define q 101  // A prime number used for hashing

int rabin\_karp(char\* text, char\* pattern) {

    int text\_length = strlen(text);

    int pattern\_length = strlen(pattern);

    int i, j;

    int pattern\_hash = 0;  // Hash value for the pattern

    int text\_hash = 0;  // Hash value for the current text window

    int h = 1;  // Used to calculate hash values for the text windows

    // Calculate the value of h as (d^(m-1)) % q

    for (i = 0; i < pattern\_length - 1; i++) {

        h = (h \* d) % q;

    }

    // Calculate the hash value for the pattern and the first window of text

    for (i = 0; i < pattern\_length; i++) {

        pattern\_hash = (d \* pattern\_hash + pattern[i]) % q;

        text\_hash = (d \* text\_hash + text[i]) % q;

    }

    // Slide the window of text over the input text one character at a time

    for (i = 0; i <= text\_length - pattern\_length; i++) {

        // Check if the hash values of the current window of text and the pattern match

        if (text\_hash == pattern\_hash) {

            // Check if the characters in the current window of text and the pattern match

            for (j = 0; j < pattern\_length; j++) {

                if (text[i+j] != pattern[j]) {

                    break;

                }

            }

            // If the characters match, we have found a match

            if (j == pattern\_length) {

                return i;

            }

        }

        // Calculate the hash value for the next window of text

        if (i < text\_length - pattern\_length) {

            text\_hash = (d \* (text\_hash - text[i] \* h) + text[i+pattern\_length]) % q;

            // Make sure the hash value is positive

            if (text\_hash < 0) {

                text\_hash += q;

            }

        }

    }

    // If we get here, no match was found

    return -1;

}

int main() {

    char text[1000], pattern[1000];

    // Get input from the user

    printf("Enter the text: ");

    fgets(text, 1000, stdin);

    printf("Enter the pattern to search for: ");

    fgets(pattern, 1000, stdin);

    // Remove the newline character at the end of the strings

    text[strcspn(text, "\n")] = 0;

    pattern[strcspn(pattern, "\n")] = 0;

    // Call the Rabin-Karp algorithm and print the result

    int result = rabin\_karp(text, pattern);

    if (result == -1) {

        printf("Pattern not found in text.\n");

    } else {

        printf("Pattern found in text starting at index %d.\n", result);

    }

    return 0;

}

**Output:**

**Naïve Approach:**

Text

Description automatically generated with medium confidence

**Rabin Karp:**

Text

Description automatically generated

**Conclusion:** Successfully wrote a program to implement Rabin Karp and Naïve Approach for string matching algorithm.