VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Ex.No: 9 Implementation of Rabin-Karp algorithm. Name : Venkatesan M

Date : 24.10.2024 Reg.No : 22BAI1259

Aim

The goal is to implement the Rabin-Karp algorithm for string matching in C++. We will use the Rabin-Karp algorithm to find the occurrences of a pattern P in a given text T using a rolling hash technique. The algorithm works modulo a prime number q. Additionally, we will analyze the number of spurious hits encountered and measure the time taken to execute the algorithm on different input sizes.

Working modulo q = 11, how many spurious hits does the Rabin-Karp matcher encounter in the text T= 3141592653589793 when looking for the pattern P=26?

Algorithm

Hash Calculation: The algorithm computes the hash value for the pattern and the initial segment of the text. It uses a rolling hash function to efficiently update the hash values for subsequent segments of the text.

Pattern Matching: If the hash values of the current segment of the text and the pattern are equal, it performs a direct comparison to check if the actual strings match (this step helps identify spurious hits).

Spurious Hits Counting: If the hash values match but the strings do not, it counts it as a spurious hit.

Sliding the Window: The algorithm slides the pattern over the text one character at a time, updating the hash value using a rolling hash function.

Pseudocode

1. Input: Pattern P, Text T, Prime number q, Length of P (m), Length of T (n)
2. Compute hash(P) and hash(T) for the first m characters
3. Set spurious_hits = 0
4. for i = 0 to n - m
    a. If hash(P) == hash(T) then
        i. If P == T[i:i+m], then print "Pattern found at index i"
        ii. Else, increment spurious_hits
    b. Update hash(T) for the next window
5. Output the number of spurious_hits

C++ Implementation

```
#include <iostream>
#include <string>
#include <cmath>
#include <chrono>
#include <fstream>

using namespace std;

// Rabin-Karp Algorithm
int rabinKarpMatcher(const string &T, const string &P, int q) {
    int n = T.length();
    int m = P.length();
    int d = 10; // The number of characters in the input alphabet (0-9 for digits)
    int h = 1;  // The value of d^(m-1)
    int pHash = 0; // Hash value for pattern
```

```
int tHash = 0; // Hash value for text
int spuriousHits = 0;

// Precompute h = d^(m-1) % q
for (int i = 0; i < m - 1; i++) {
    h = (h * d) % q;
}

// Compute initial hash values for pattern and text
for (int i = 0; i < m; i++) {
    pHash = (d * pHash + P[i]) % q;
    tHash = (d * tHash + T[i]) % q;
}

// Slide the pattern over text one by one
for (int i = 0; i <= n - m; i++) {
    // Check if the hash values are the same
    if (pHash == tHash) {
        // Verify the actual characters match
        if (T.substr(i, m) == P) {
            cout << "Pattern found at index " << i << endl;
        } else {
            // Count spurious hits if hash matches but pattern does not
            spuriousHits++;
        }
    }

    // Calculate hash value for the next window of text
    if (i < n - m) {
        tHash = (d * (tHash - T[i] * h) + T[i + m]) % q;
        if (tHash < 0) {
```

```
            tHash = (tHash + q);
          }
        }
    }

    return spuriousHits;
}

int main() {
    string T = "3141592653589793";
    string P = "26";
    int q = 11; // Prime number for modulo

    // Measure time
    auto start = chrono::high_resolution_clock::now();
    int spuriousHits = rabinKarpMatcher(T, P, q);
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed = end - start;

    // Output the number of spurious hits and time taken
    cout << "Number of spurious hits: " << spuriousHits << endl;
    cout << "Time taken: " << elapsed.count() << " seconds" << endl;

    // Save results to file for graph plotting
    ofstream out("rabin_karp_times.txt", ios::app);
    out << T.length() << " " << elapsed.count() << "\n";
    out.close();

    return 0;
}
```

Explanation

Pattern found at index 6

Number of spurious hits: 3

Time taken: 5.3031e-05 seconds

Random Sampling

```
#include <iostream>
#include <string>
#include <chrono>
#include <fstream>
#include <cstdlib>
#include <ctime>

using namespace std;

// Rabin-Karp Algorithm
int rabinKarpMatcher(const string &T, const string &P, int q) {
   int n = T.length();
   int m = P.length();
   int d = 10; // The number of characters in the input alphabet (0-9 for digits)
   int h = 1;  // The value of d^(m-1)
   int pHash = 0; // Hash value for pattern
   int tHash = 0; // Hash value for text
   int spuriousHits = 0;

   // Precompute h = d^(m-1) % q
   for (int i = 0; i < m - 1; i++) {
      h = (h * d) % q;
   }

   // Compute initial hash values for pattern and text
   for (int i = 0; i < m; i++) {
      pHash = (d * pHash + P[i]) % q;
      tHash = (d * tHash + T[i]) % q;
   }
```

```
    // Slide the pattern over text one by one
    for (int i = 0; i <= n - m; i++) {
        // Check if the hash values are the same
        if (pHash == tHash) {
            // Verify the actual characters match
            if (T.substr(i, m) == P) {
                // Pattern found
            } else {
                // Count spurious hits if hash matches but pattern does not
                spuriousHits++;
            }
        }

        // Calculate hash value for the next window of text
        if (i < n - m) {
            tHash = (d * (tHash - T[i] * h) + T[i + m]) % q;
            if (tHash < 0) {
                tHash = (tHash + q);
            }
        }
    }

    return spuriousHits;
}

// Function to generate random text of given length
string generateRandomText(int length) {
    string text;
    for (int i = 0; i < length; i++) {
        text += '0' + rand() % 10; // Random digit between 0 and 9
    }
    return text;
}

int main() {
    srand(time(0));
    ofstream out("rabin_karp_times.txt");

    // Pattern to search for
    string P = "26";
    int q = 11; // Prime number for modulo

    // Varying input sizes
    for (int length = 1000; length <= 10000; length += 1000) {
        // Generate random text
```

```
        string T = generateRandomText(length);

        // Measure time
        auto start = chrono::high_resolution_clock::now();
        int spuriousHits = rabinKarpMatcher(T, P, q);
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<double> elapsed = end - start;

        // Output the number of spurious hits and time taken
        cout << "Text length: " << length << ", Time taken: " << elapsed.count() << " seconds, Spurious
hits: " << spuriousHits << endl;

        // Save results to file for graph plotting
        out << length << " " << elapsed.count() << "\n";
    }

    out.close();
    return 0;
}
```

Plotting

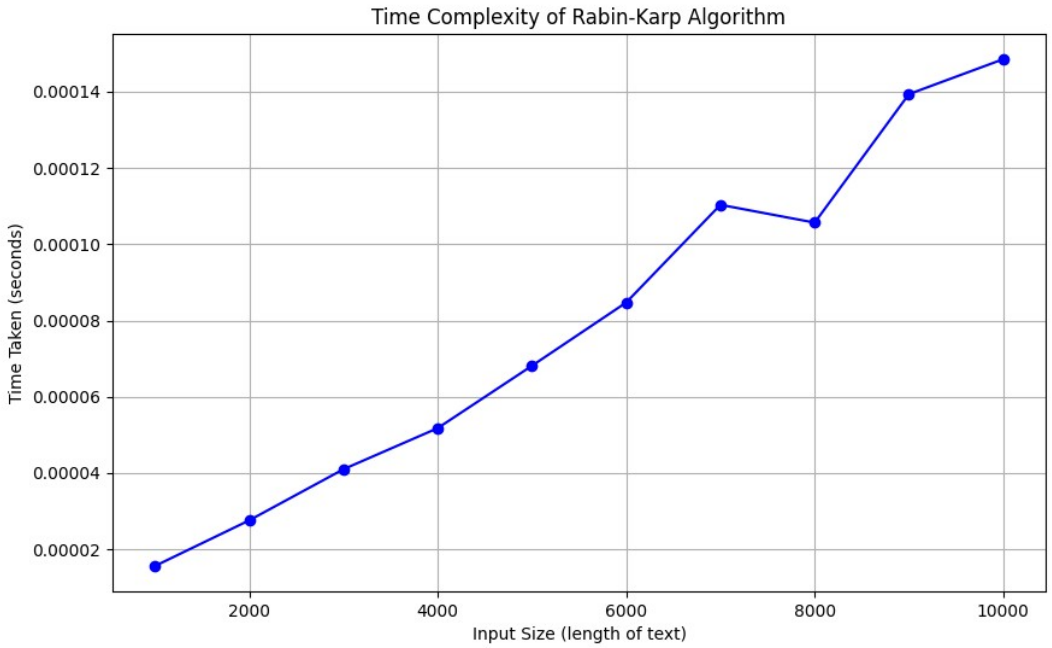
```
import matplotlib.pyplot as plt


# Read data from file
input_sizes = []
times = []


with open("rabin_karp_times.txt", "r") as file:
    for line in file:
        size, time = line.split()
        input_sizes.append(int(size))
        times.append(float(time))


# Plotting
plt.figure(figsize=(10, 6))
plt.plot(input_sizes, times, marker='o', linestyle='-', color='b')
```

```
plt.title('Time Complexity of Rabin-Karp Algorithm')
plt.xlabel('Input Size (length of text)')
plt.ylabel('Time Taken (seconds)')
plt.grid(True)

# Save the plot as a PNG file
plt.savefig("rabin_karp_time_complexity.png")

# Show the plot
plt.show()
```

Time Complexity Analysis

Input size	Time (microseconds)
1000	1.561e-05
2000	2.758e-05
3000	4.099e-05
4000	5.177e-05
5000	6.813e-05

Graph between varying size of inputs and time



Complexity Analysis:

Preprocessing: Calculating the initial hash values takes O(m), where m is the length of the pattern.

- **Pattern Matching:** The matching loop executes n−m+1 times, where n is the length of the text. In each iteration, updating the hash value takes O(1), making the overall complexity O(n).
- **Verification:** In the worst case, all hash matches might require O(m) character comparisons.

Thus, the average case time complexity is O(n+m), while the worst case, considering spurious hits, could be O(nm).