Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Name	Tejas Jadhav
UID No.	2022301006
Class	COMPS A (B batch)
Experiment No.	9

Aim: To implement string matching algorithm using the naïve approach and Rabin Karp approach

Theory:

- **String Matching**

String matching is a process of finding all occurrences of a pattern (a sequence of characters) within a text (a larger sequence of characters).

String matching is a fundamental problem in computer science, which involves finding a substring within a larger string. It is an important problem in various applications, including text processing, image processing, and bioinformatics.

- **Rabin Karp**

Rabin Karp is a string matching algorithm that uses hashing to find the occurrence of a pattern string within a text string. It was developed by Michael O. Rabin and Richard M. Karp in 1987. The algorithm works by comparing the hash values of the pattern string and all possible substrings of the text string.

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Algorithm:

NaiveStringMatch(T, P)
1. n ← length[T]
2. m ← length[P]
3. for i ← 0 to n - m
4. j ← 0
5. while j < m and P[j] = T[i+j]
6. j ← j + 1
7. if j = m
8. print "Pattern occurs with shift" i

1. RabinKarpMatch(text, pattern):
2. n = length(text)
3. m = length(pattern)
4. pattern_hash = hash(pattern)
5.
6. for i from 0 to n-m:
7. text_hash = hash(text[i:i+m])
8.
9. if pattern_hash == text_hash:
10. if pattern == text[i:i+m]:
11. return i
12. return -1

Code:

```
#include <bits/stdc++.h>
using namespace std;

int c_count_naive = 0;
int c_count_rbkarp = 0;

int to_int(char c) { return c - 'a' + 1; }

bool is_prime(int n) {
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            return false;
```

```
        }
    }
    return true;
}

int gen_random_prime(int n) {
    int p = rand() % n;
    while (!is_prime(p)) {
        p = rand() % n;
    }
    return p;
}

bool compare(string str, string substr, int start, bool rabin_karp = false)
{
    for (int i = 0; i < substr.length(); i++) {
        if (rabin_karp) {
            c_count_rbkarp++;
        } else {
            c_count_naive++;
        }
        if (str[start + i] != substr[i]) {
            return false;
        }
    }

    return true;
}

int s_hash(string str, int start, int n, int q) {
    int h = 0;
    int pow_10 = n;

    for (int i = 0; i < n; i++) {
        h += to_int(str[start + i]) * pow(10, pow_10);
        pow_10--;
    }
    h = h / 10;

    return h % q;
}
```

```
void naive_matching(string str, string substr) {
    int n = str.length();
    int m = substr.length();

    if (n < m) {
        cout << "Substring cannot be found\nOriginal String is smaller" <<
endl;
        return;
    }

    for (int i = 0; i <= n - m; i++) {
        if (compare(str, substr, i, false)) {
            cout << "Substring found at index " << i << endl;
        }
    }
}

void rabin_karp(string str, string substr, int q) {
    int n = str.length();
    int m = substr.length();

    if (n < m) {
        cout << "Substring cannot be found\nOriginal String is smaller" <<
endl;
        return;
    }

    int h_substr = s_hash(substr, 0, m, q);
    int spurious_hits = 0;

    for (int i = 0; i <= n - m; i++) {
        int h_str = s_hash(str, i, m, q);
        if (h_substr == h_str) {
            if (compare(str, substr, i, true)) {
                cout << "Substring found at index " << i << endl;
            } else {
                spurious_hits++;
            }
        }
    }
```

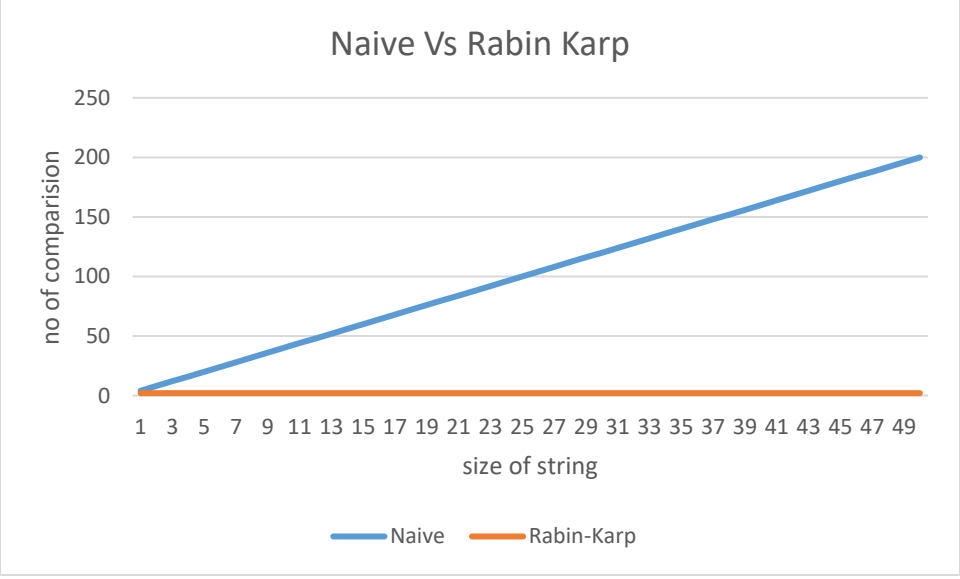
```
    cout << "\nSpurious hits: " << spurious_hits << endl;
}

int main() {
    string str, substr;
    srand(time(0));
    cout << "Enter a string: ";
    cin >> str;

    cout << "Enter a substring: ";
    cin >> substr;

    int prime;
    cout << "Enter a prime number (0 for random): ";
    cin >> prime;

    if (prime == 0) {
        prime = gen_random_prime(100000);
        cout << "Random prime number generated: " << prime << endl;
    } else if (!is_prime(prime)) {
        cout << "Not a prime number" << endl;
        return 0;
    }

    cout << endl << "Applying Naive algorithm" << endl;

    naive_matching(str, substr);

    cout << endl << "Applying Rabin-Karp algorithm" << endl;

    rabin_karp(str, substr, prime);

    cout << endl
         << "Number of comparisons in Naive algorithm: " << c_count_naive
         << endl;
    cout << "Number of comparisons in Rabin-Karp algorithm: " <<
c_count_rbkarp
         << endl;

    return 0;
```

```
}
```

Output:

```
PS D:\Tejas\clg\daa\Experiment 09\code> g++ .\rabinkarp.cpp
PS D:\Tejas\clg\daa\Experiment 09\code> ./a
Enter a string: abcdefabcdefjkjkjkjkjkjklmnopagcagbcaabc
Enter a substring: abc
Enter a prime number (0 for random): 0
Random prime number generated: 11527

Applying Naive algorithm
Substring found at index 0
Substring found at index 6
Substring found at index 37

Applying Rabin-Karp algorithm
Substring found at index 0
Substring found at index 6
Substring found at index 37

Spurious hits: 0

Number of comparisons in Naive algorithm: 47
Number of comparisons in Rabin-Karp algorithm: 9
PS D:\Tejas\clg\daa\Experiment 09\code>
```

Chart:



Conclusion:

Rabin-Karp is a simple and efficient string matching algorithm that can find all occurrences of a pattern within a text in linear time. It uses hashing to compare substrings, and can handle any alphabet size. Spurious hits can be minimized by using a strong hash function.