**RABIN KARP ALGO**

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

// Function to implement Rabin-Karp algorithm

void rabinKarp(string pattern, string text) {

int prime = 101; // A prime number for hashing

int patternLength = pattern.length();

int textLength = text.length();

int patternHash = 0; // Hash value for pattern

int textHash = 0; // Hash value for current window of text

int count = 0; // Operation count

// Calculate hash of pattern and initial window of text

for (int i = 0; i < patternLength; i++) {

patternHash += pattern[i] \* pow(prime, i);

textHash += text[i] \* pow(prime, i);

count += 2; // Counting hash calculations

}

// Slide the pattern over text one by one

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

if (patternHash == textHash) {

// Check character by character

int j;

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

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

break;

}

count++; // Counting comparison operations

}

if (j == patternLength) {

cout << "Pattern found at index " << i << endl;

}

}

// Calculate hash value for next window of text

if (i < textLength - patternLength) {

textHash = (textHash - text[i] + text[i + patternLength] \* pow(prime, patternLength - 1));

count += 2; // Counting hash calculations

}

}

cout << "Number of operations: " << count << endl;

}

int main() {

string text = "ABABDABACDABABCABAB";

string pattern = "ABABCABAB";

cout << "Text: " << text << endl;

cout << "Pattern: " << pattern << endl;

cout << "Rabin-Karp Algorithm:\n";

rabinKarp(pattern, text);

return 0;

}

**KNUTH MORRIS PRATT**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Function to compute the LPS (Longest Prefix which is also Suffix) array

void computeLPSArray(string pattern, vector<int>& lps) {

int len = 0; // Length of the previous longest prefix suffix

int M = pattern.length();

int i = 1;

lps[0] = 0; // lps[0] is always 0

// Calculate lps[i] for i = 1 to M-1

while (i < M) {

if (pattern[i] == pattern[len]) {

len++;

lps[i] = len;

i++;

} else {

if (len != 0) {

len = lps[len - 1];

// No increment of i here

} else {

lps[i] = 0;

i++;

}

}

}

}

// Function to implement KMP algorithm

void KMPSearch(string pattern, string text) {

int M = pattern.length();

int N = text.length();

vector<int> lps(M); // LPS array for pattern

computeLPSArray(pattern, lps);

int i = 0; // Index for text[]

int j = 0; // Index for pattern[]

int count = 0; // Operation count

while (i < N) {

count++; // Counting comparison operation

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

j++;

i++;

}

if (j == M) {

cout << "Pattern found at index " << i - j << endl;

j = lps[j - 1];

} else if (i < N && pattern[j] != text[i]) {

if (j != 0) {

j = lps[j - 1];

} else {

i = i + 1;

}

}

}

cout << "Number of operations: " << count << endl;

}

int main() {

string text = "ABABDABACDABABCABAB";

string pattern = "ABABCABAB";

cout << "Text: " << text << endl;

cout << "Pattern: " << pattern << endl;

cout << "Knuth-Morris-Pratt (KMP) Algorithm:\n";

KMPSearch(pattern, text);

return 0;

}