DAA Lab - Session 2 - Naive String Matching Algorithm

Brute Force: Implementation of **String Matching** algorithm

**Problem Definition:**

Search for a substring (pattern) in a longer string (text) using naive string matching algorithm.

**Input:** Input begins with a text (1 ≤ length of the text ≤ 220) in a single line. The second line has t (1 ≤ t ≤ 100), the number of patterns ((1 ≤ length of the pattern ≤ 220)) to be searched in the text. The following t number of lines to have one pattern per line. The text and pattern could have whitespaces like spaces and tabs as characters.

**Output:** For each pattern, print the index (0 ≤ index < length of the text) of the beginning the pattern in the text in a new line. Print ‘-1’ if the substring is not found in the text. Print the total time taken in seconds.

**Sample Input:**

If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident.

2

chose

programming

**Sample Output:**

10

-1

0.123456 sec.

**Algorithm:**

**Algorithm NaiveStringMatching(T[0..n-1],P[0..m-1])**

//Implements brute-force string matching.

//Input: An array T[0..n-1] of n chars representing a text

// and an array P[0..m-1] of m chars representing a pattern.

//Output: The index of the **first** character in the text

// that starts a matching substring

// or -1 if the search is unsuccessful.

**for i ← 0 to n-m**

**j ← 0**

**while (j < m) and (P[j] ≠ T[i+j] do**

**j ← j + 1**

**endwhile**

**if (j = m) return i**

**return -1**

**Practice-Problems:**

1. Search for the last occurrence of the pattern.
2. Search for all occurrences of the pattern.
3. Search for the first occurrence of the pattern. If not the pattern is not present, search for the longest prefix of the pattern. If not not even the first character of the pattern is present in the text, then return -1.