DAA Lab - Session 11 - Horspool’s String Matching Algorithm

Space and Time Tradeoffs: Implementation of **Horspool’s String Matching** algorithm

**Problem Definition:**

Search for a substring (pattern) in a longer string (text) using Horspool’s 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.

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

**Table[alphabet size] ← ShiftTable(P[0..m-1])**

**i ← m-1**

**while (i < n)**

**j ← 0**

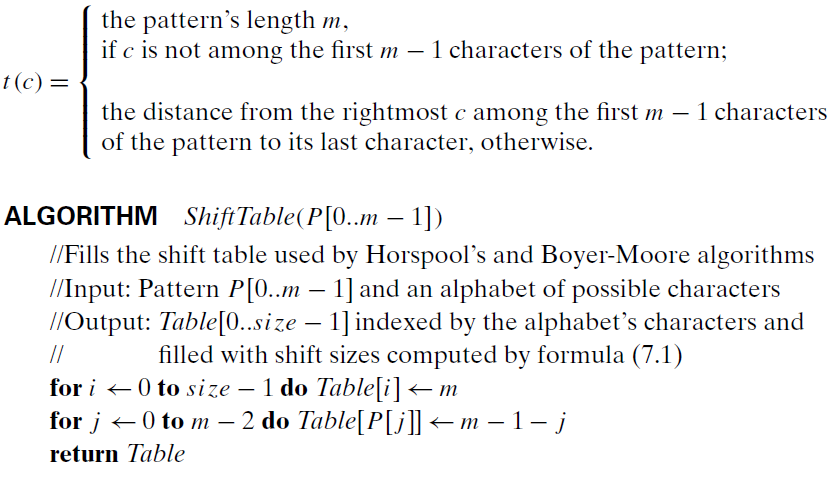
**while (j < m and T[i-j] = P[m-1-j])**

**j ← j + 1**

**if(j = m) return i-(m-1)**

**i ← i + Table[ T[i] ]**

**return -1**



**Practice-Problems:**

1. Search for the last occurrence of the pattern.
2. Search for all occurrences of the pattern.
3. Compare the number of character comparisons made with the Naive string matching algorithm.