

Alternative Homework 4: String Search

Note. This is a parody of CMU and MIT OCW content. The original assignments and related materials can be retrieved from <https://bit.ly/3i1vnHB>, <https://bit.ly/31k7uVI>, <https://bit.ly/37Wgyl4>.

Question 1.

Knuth-Morris-Pratt string search algorithm makes use of a DFA (Deterministic Finite Automaton) to keep track of the longest feasible prefix of S that has been matched so far in the input text T . The transitions between the states in this DFA automaton can be represented using the *prefix function* or the `memo[i]` table, where $i \in \{0, \dots, \ell - 1\}$.

These concepts are defined in Section 1 of <https://bit.ly/3fXWnWT>.

Assume that somebody wants to build a DFA automaton that reads text T once from left to right. S/he searches for either one of the two strings: either $S_1 = 1110111101$ or $S_2 = 10101$ (after the first match of either S_1 or S_2 is found, the automaton reaches its final accepting state and stops).

How many states would that DFA have, and what data structure (instead of the table `memo[i]`) can be used to encode the transitions in this DFA?

Question 2. Suppose you are given a searchable text $T[0..n-1]$ of length n , consisting of symbols **a** and **b**. Suppose further that you are given a pattern string $S[0..\ell-1]$ of length ℓ much smaller than n , consisting of symbols **a**, **b**, and *****, representing a pattern to be found in the original text T . The symbol ***** is a “wild card” symbol, which matches a single symbol, either **a** or **b**. The other symbols must match exactly. The problem is to output a sorted list M of valid “match positions”, which are positions j in S such that pattern P matches the substring $S[j..j+\ell-1]$. For example, if $T = ababbab$ and $S = ab*$, then the output should be $[0, 2]$.

Consider an equivalent of Knuth-Morris-Pratt algorithm that reads the searchable text T once from left to right and (upon every mismatch between the text T and pattern S) shifts the pattern S forward by the smallest feasible amount. We want to search for the pattern $S = 11 * 0111 * 01$.

(A) Write the pseudocode for this modified KMP algorithm.

(B) Fill in the table of shifts (also called *prefix function* or `memo[i]`) describing the shifts, if the T does not match the search pattern $S = 11 * 0111 * 01$ at some position.

This is a parody of Problem 2.1, see p.1 in <https://bit.ly/2XKX5AB>.

Question 3.

Consider the following prehashing function converting strings S of length ℓ into integer “keys”:

$$k(S) = (S[0] \cdot b^{\ell-1} + S[1] \cdot b^{\ell-2} + \dots + S[\ell-1]) \bmod q. \quad (1)$$

Here b (called the “number base”) is a number larger than the alphabet size used for S .

The same prehashing function can be applied to a substring of a longer text T (processing exactly ℓ consecutive symbols):

$$\begin{aligned} k(T[i..i+\ell-1]) &= \\ &= (S[i] \cdot b^{\ell-1} + S[i+1] \cdot b^{\ell-2} + \dots + T[i+\ell-1]) \bmod q, \end{aligned} \quad (2)$$

Consider also the following hashing function (See *multiplication method* in <https://bit.ly/2V8UfDF>)

$$h(k) = [(a \cdot k) \bmod 2^w] \gg (w - r), \quad (3)$$

In this expression:

- \gg denotes the bitwise “shift right” operator,
- $2^r = m$ is the hash table size,
- w is the bit-length of a “machine word” (choose whatever is convenient; regardless of the hardware architecture).
- a is an odd integer between 2^{w-1} and 2^w .

Our goal is to use the hashing method:

$$h(k(T[i..i+\ell-1]))$$

in the Rabin-Karp string search. We want to find patterns S (of length $\ell \leq 100$ symbols) in a long text T . Both S and T are written in ASCII, using its 128 symbol alphabet.

(A) How would you compute the `r.append(c)` and `r.skip(c)` operations in the rolling hash ADT, if the prehashing and hashing are implemented using the expressions (1), (2) and (3).

The Rolling Hash abstract datatype (ADT) and Rabin-Karp algorithm are explained in <https://bit.ly/2BxpNMu> (starting from 38:00).

(B) Write the time complexity of the overall Rabin-Karp algorithm using $O(\dots)$ (the Big-O notation) for this hashing method. Denote the length of the text T by $n = |T|$, and the length of the search pattern S by $\ell = |S|$.

Note. This was inspired by Problem 4-1 from <https://bit.ly/3dyNtgK>.