Formal Languages & Automata Theory Project

String Matching

This project is about the string matching problem. String matching is the problem of finding the number of occurrences of a pattern in a given text.

We formalize the string-matching problem as follows:

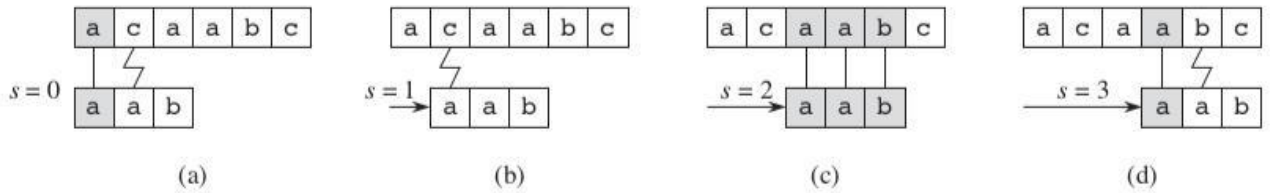
We assume that the text is an array $T[1 . . n]$ of length n and that the pattern is an array $P[1 . . m]$ of length $m \leq n$. We further assume that the elements of P and T are characters drawn from a finite alphabet Σ. For example, we may have $\{0, 1\}$ or $\{a, b, .., z\}$. The character arrays P and T are often called strings of characters.

Algorithms for string matching:

In this project you will implement two algorithms for string matching. The first one is called Naive-String-Matching and the pseudocode is given below:

```
NAIVE-STRING-MATCHER(T, P)
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4      if P[1..m] == T[s + 1..s + m]
5          print "Pattern occurs with shift" s
```

The naive algorithm finds all valid shifts using a loop that checks the condition $P[1 . . m] == T[s+1 . . s+m]$ for each of the $n - m + 1$ possible values of s. Figure below portrays the naive string-matching procedure as sliding a "template" containing the pattern over the text, noting for which shifts all of the characters on the template equal the corresponding characters in the text. The for loop of lines 3–5 considers each possible shift explicitly. The test in line 4 determines whether the current shift is valid; this test implicitly loops to check corresponding character positions until all positions match successfully or a mismatch is found.
Line 5 prints out each valid shift s.



The second algorithm is called Finite-Automata-Matcher and the pseudocode is given below:

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1  n = T.length
2  q = 0
3  for i = 1 to n
4       q = δ(q, T[i])
5       if q == m
6            print "Pattern occurs with shift" i − m
```

The pattern to be matched is represented by the transition function δ. The text is given to the automaton as input and the automaton reads the text character by character and changes states according to its transition function. When the automaton enters into the accepting state in prints a message. The length of the pattern is *m*, the set of states *Q* is {0, 1, . . ., *m*}, the start state is 0, and the only accepting state is state *m*.

For the Finite-Automata-Matcher algorithm to work the transition function δ has to be computed. The following algorithm Compute-Transition-Function(*P*, computes given a the pattern *P* and the alphabet Σ.

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
1  m = P.length
2  for q = 0 to m
3       for each character a ∈ Σ
4            k = min(m + 1, q + 2)
5            repeat
6                 k = k − 1
7            until P_k ⊐ P_q a
8            δ(q, a) = k
9  return δ
```

The nested loops beginning on lines 2 and 3 consider all states *q* and all characters *a*, and lines 4–8 set $\delta(q, a)$ to be the largest *k* such that $P_k \sqsupset P_q a$. The code starts with the largest conceivable value of *k* which is min(*m*, *q* + 1). It then decreases *k* until $P_k \sqsupset P_q a$, which must eventually occur, since $P_0 = \varepsilon$ is a suffix of every string.

We say that a string *w* is a suffix of a string *x*, denoted $w \sqsupset x$, if $x = yw$ for some $y \in \Sigma^*$.

In this project you can assume that the alphabet contains only digits and lower case letters, that is, {0, . ., 9, *a*, . ., *z*}.

In the end of a single run you are required to print the number of occurrences of the pattern in each line and the time it takes for both string matching algorithms algorithms. Note that for the Finite-Automata-Matcher algorithms you should include the time it takes to compute the transition function.

Example Run

You will be given a pattern and a text file and output the number of occurrences of the pattern in the text file. You can assume that the array $T[1 . . n]$ corresponds to a line of the text file. For example given the following pattern and text file:

Pattern: automata

Text File:

> Automata Theory is a theoretical branch of computer science. It established its roots during the 20th Century, as mathematicians began developing - both theoretically and literally - machines which imitated certain features of man, completing calculations more quickly and reliably. The word automaton itself, closely related to the word "automation", denotes automatic processes carrying out the production of specific processes. Simply stated, automata theory deals with the logic of computation with respect to simple machines, referred to as automata. Through automata, computer scientists are able to understand how machines compute functions and solve problems and more importantly, what it means for a function to be defined as computable or for a question to be described as decidable .

The output should be:

Line 1: 1 occurrence
Line 6: 1 occurrence
Line 7: 2 occurrences

Time for Naive-String-Matching: 21.14 ms.
Time for Finite-Automata-Matcher: 2.46 ms.