CS 5114
Solutions to Homework Assignment 9

Meghendra Singh

April 21, 2017

[30] 1. Exercise 32.3-4. Given two patterns P and P', describe how to construct a finite automaton that determines all occurrences of *either* pattern. Try to minimize the number of states in your automaton.

For the given problem of constructing a finite automaton that determines all occurrences of the two patterns P and P', lets assume that P is m characters long and P' is n characters long. In order to minimize the number of states in the automaton, we also assume that both P and P' have a longest common prefix string till an arbitrary position k in the two patterns. Therefore,

$$P[i] = P'[i]\,,\ \forall i\ \ 1 \leq i \leq k$$

There can be four possible values of k:

1) $k < m$ and $k < n$, i.e. $P[1...k]$ is a longest common prefix of both P and P'.

2) $k = m < n$ (or equivalently $k = n < m$), i.e. one of the patterns is a prefix of the other pattern.

3) $k = m = n$, i.e. both the patterns are exactly the same.

4) $k = 0$, i.e. the two patterns P and P' have no common prefix.

Before discussing the construction of automata for the above cases, we first define a ***suffix function*** σ for an arbitrary pattern P'' of length q which we will be using in our construction. σ maps the alphabet Σ^* of P'', to the set $\{0, 1, ..., q\}$ such that $\sigma(x)$ is the length of the longest prefix of P'' that is also a suffix of a string x. The suffix function can be written as follows:

$$\sigma(x) = \max\{l : P'' \sqsupset x\}$$

Next, we describe the construction of automata for the fours cases:

1) $k < m$ and $k < n$: In this case, we begin by constructing an automaton for the longest common prefix of the two patterns, i.e. $P_k = P[1...k]$ as follows:

i. The state set Q is $\{0, 1, ..., k\}$. The start set q_0 is state 0. Let the set of unique alphabets in P be Σ_P, and those in P' be $\Sigma_{P'}$. The set of input alphabet for the automata will be $\Sigma = \Sigma_P \cup \Sigma_{P'}$.

ii. Let σ_k be the suffix function corresponding to the pattern P_k. The transition function δ can be defined for any state $q \in Q$ and input character a, as follows:

$$\delta(q, a) = \sigma_k(P_q a)\,,\ \forall q\ \ 0 \leq q \leq k \text{ and } \forall a \in \Sigma$$

iii. Next, we need to consider the remaining characters which are not in the longest common prefix of P. We add the states $\{k+1, k+2, ..., m\}$ to the set Q and add the transition $\delta(k, P[k+1]) = k+1$ from state k to state $(k+1)$.

iv. Let σ_p be the suffix function corresponding to the pattern $P[(k+1), ..., m]$. Now, we add the subsequent transitions for states $(k+1), ..., m$ similar to **step ii.** i.e.:

$$\delta(q, a) = \sigma_p(P_q a), \ \forall q \ \ (k+1) \leq q \leq m \text{ and } \forall a \in \Sigma$$

v. Add the state m to the set of final states, i.e. $A = \{m\}$.

vi. Next, we consider the remaining characters which are not in the longest common prefix of P'. We add the states $\{(k+1)', (k+2)', ..., n\}$ to the set Q and add the transition $\delta(k, P'[k+1]) = (k+1)'$ from state k to state $(k+1)'$.

vii. Let $\sigma_{p'}$ be the suffix function corresponding to the pattern $P[(k+1)', ..., n]$. Now, we add the subsequent transitions for states $(k+1)', ..., n$ similar to **step iv.** i.e.,

$$\delta(q, a) = \sigma_{p'}(P'_q a), \ \forall q \ \ (k+1)' \leq q \leq n \text{ and } \forall a \in \Sigma$$

viii. Add the state n to the set of final states, i.e. $A = \{m, n\}$.

The complete automaton constructed in the above steps can be written as follows:

$$\begin{aligned}
Q &= \{0, 1, ..., k, (k+1), (k+2), ..., m, (k+1)', (k+2)', ..., n\} \\
q_0 &= 0 \\
A &= \{m, n\} \\
\Sigma &= \Sigma_P \cup \Sigma_{P'} \\
\delta(q, a) &= \sigma_k(P_q a), \ \forall q \ \ 1 \leq q \leq k \text{ and } \forall a \in \Sigma \\
\delta(k, P[k+1]) &= (k+1) \\
\delta(k, P'[k+1]) &= (k+1)' \\
\delta(q, a) &= \sigma_P(P_q a), \ \forall q \ \ (k+1) \leq q \leq m \text{ and } \forall a \in \Sigma \\
\delta(q, a) &= \sigma_{P'}(P'_q a), \ \forall q \ \ (k+1)' \leq q \leq n \text{ and } \forall a \in \Sigma
\end{aligned}$$

This automaton will determine all occurrences of both the patterns P and P' and would have a minimum number of states.

2) $k = m < n$ (or equivalently $k = n < m$): In this case the shorter patterns is a prefix of the longer pattern. Assuming P' to be the shorter pattern of length $n = k$, we can follow the construction steps until **step v.** above and add the state k to the set of final states, i.e $A = \{m, k\}$. The obtained automata will determine all occurrences of both the patterns P and P' and would have a minimum number of states.

3) $k = m = n$: In this case as both the patterns are exactly the same, we can follow the construction steps until **step ii.** in section **1)** above and add the state k to the set of final states, i.e $A = \{k\}$ to obtain the required finite automaton.

4) $\underline{k = 0}$: In this case the two patterns have no common prefix and substituting k with 0 in the construction steps in section **1)** above will result in the required automaton.

Consider the simple example, where we build the automaton for two patterns P = `abcd` and P' = `abad`. Here $m = 4$, $n = 4$, $k = 2$ and the longest common prefix P_k = `ab`. The automaton constructed for these two patterns using the above steps is as follows:

$$Q = \{0, 1, 2, 3, 4, 3', 4'\}$$
$$q_0 = 0$$
$$A = \{4, 4'\}$$
$$\Sigma = \{a, b, c, d\}$$

The transitions (δ) are shown in the last automaton of figure 1 below, which represents the automation construction process.

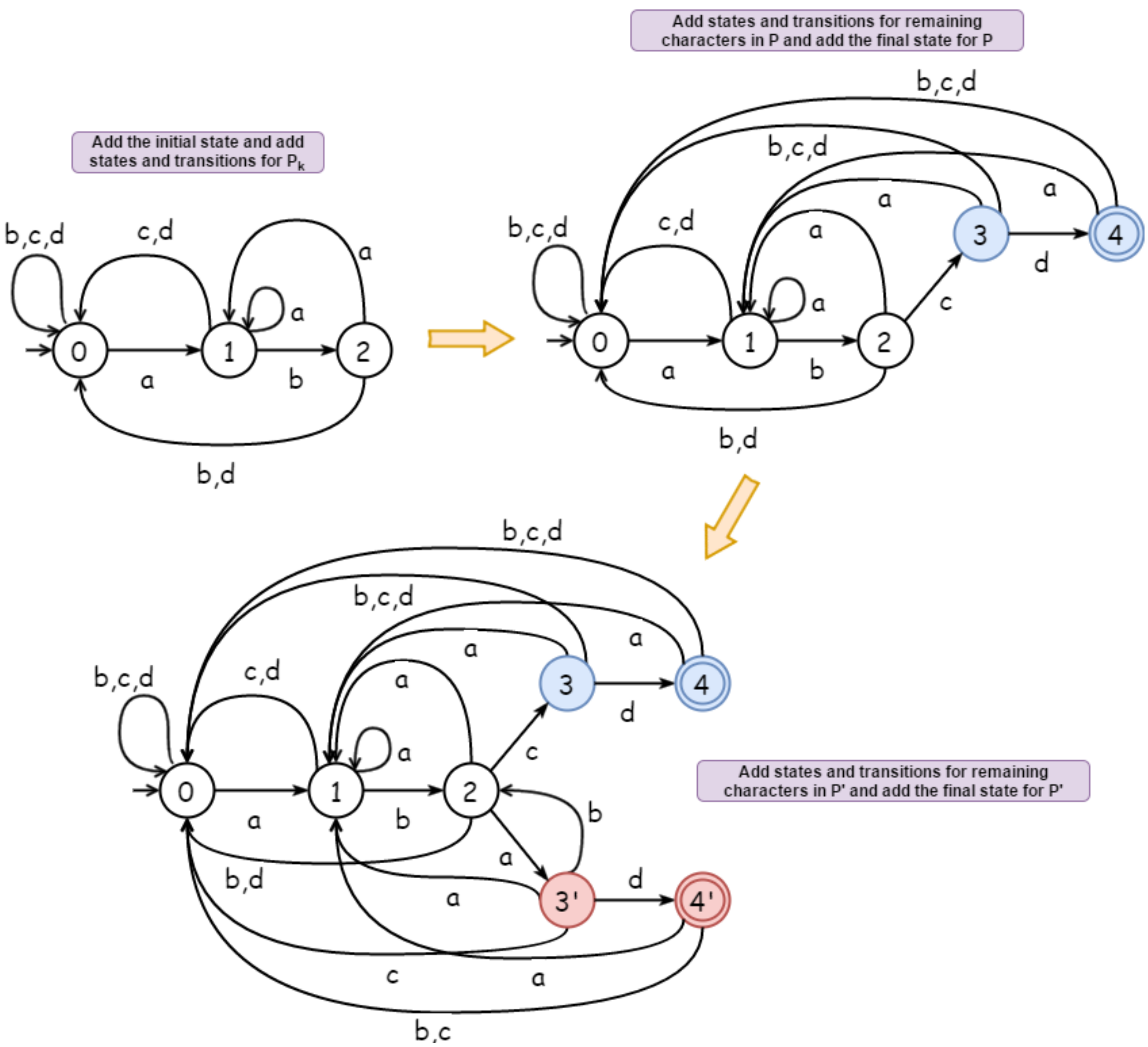

Figure 1: Construction of automaton which determines all occurrences of patterns `abcd` and `abad` in any text.

---

[30] 2. Exercise 32.4-7. Give a linear-time algorithm to determine whether a text T is a cyclic rotation of another string T'. For example, `arc` and `car` are cyclic rotations of each other. Give pseudocode for your algorithm.

For any two strings to be cyclic rotations of each other first they need to be of the same length. Consider the string in the question i.e. let $T = \texttt{arc}$, the set of possible cyclic rotations of T are $C = \{\texttt{rca,car}\}$. If we concatenate T to itself i.e. $T \cdot T$, we get the string: `arcarc`, for which the set of sub-strings of length 3 are: $S = \{\texttt{arc,rca,car}\}$. We can clearly see that the set of cyclic rotations is a subset of the set of length 3 sub-strings of the concatenated string, i.e. $C \subset S$. Hence, if we consider the concatenated string (i.e. $T \cdot T$)as text and the other string T' as a pattern, we can be sure that T is a cyclic rotation of T' if the pattern T' is found at-least once in the concatenated text $T \cdot T$.

We use this idea in the algorithm Is-Cyclic-Rotation below. We make use of a matcher subroutine Kmp-Matcher(T'', T'), which uses Knuth-Morris-Pratt algorithm to find if the pattern T' exists in text T'' (i.e. returns $True$, if it finds T' in T'') in linear time on the length of text T''. The pseudocode for Is-Cyclic-Rotation algorithm is as follows:

Is-Cyclic-Rotation(T, T')

```
1   Let m = T.length
2   Let n = T'.length
3   // Proceed only if the lengths of the text strings match
4   if m == n
5       // Concatenate T with itself
6       T'' = T · T
7       // Kmp-Matcher returns True if T' ∈ T'', False otherwise.
8       return Kmp-Matcher(T'', T')
9   // Return False if the string length don't match
10  return False
```

In the pseudocode above, we simply concatenate T with itself (line 6) and execute Kmp-Matcher with the concatenated string T'' as the text and T' as the pattern (line 8). The concatenation step in line 6 will execute in linear time for the length of T, i.e. $\Theta(m)$. The subroutine Kmp-Matcher will execute in linear time with respect to the length of T'', i.e. $\Theta(2m) \approx \Theta(m)$. Hence the time complexity of Is-Cyclic-Rotation is $\Theta(m)$, (or equivalently $\Theta(n)$, as $m = n$) where m is the length of the text string T. Therefore, we have a linear-time algorithm to determine if a text string T is a cyclic rotation of another text string T'.

Consider the example given in the question, i.e. $T = \texttt{arc}$ and $T' = \texttt{car}$. The algorithm Is-Cyclic-Rotation(T, T') will first compute $T'' = \texttt{arcarc}$, and then call Kmp-Matcher(T'', T'). Kmp-Matcher will find $\texttt{car} \in \texttt{ar}\underline{\texttt{car}}\texttt{c}$ and return $True$, which will be returned by Is-Cyclic-Rotation as is. Consider the case if $T' = \texttt{cer}$, and since $\texttt{cer} \notin \texttt{arcarc}$ Kmp-Matcher will return $False$, which will be returned as is.
