Efficient Algorithms and Problem Complexity

– Searching in Text –

Frank Drewes
Department of Computing Science
Umeå University
Today’s Menu

1. Knuth-Morris-Pratt Substring Search

2. Regular Expression Matching
Searching for a String in a Text

The problem

Input: A text \( t = t_0 \ldots n \) and a pattern \( p = p_0 \ldots m \), both in \( \Sigma^* \).
Output: The least \( i \) such that \( t_{i \ldots i+m} = p \) (and \(-1\) if not such \( i \) exists).

- Naive algorithm: Check for \( i = 0, \ldots, n - m \) whether \( t_{i \ldots i+m} = p \).
- Running time: \( O(mn) \) (\( \Rightarrow \) quadratic if the pattern is large).
- Nowadays, texts and patterns are very large in some applications. \( \Rightarrow O(m + n) \) would be desirable.
The Observation by Knuth, Morris, and Pratt

Suppose the initial part of $p = \text{statistics}$ matches $t_{i\ldots i+6}$, but $t_{i+7} \neq i$.

- The naive algorithm increases $i$ by 1 and tries to match $p$ again.
- But we could immediately shift $i$ by 5 positions!
- If the failure occurs after $\text{statistics}$, we can shift by 8 positions.

We can safely shift $p$ by

$$\text{shift}(k) = \min\{s \geq 1 \mid p_{0\ldots k-s} = p_{s\ldots k}\}.$$
The Knuth-Morris-Pratt Algorithm

\[ KMP-Search(p, t) \text{ where } p = p[0, \ldots, m] \text{ and } t = t[0, \ldots, n] \]

Compute shift table \( \text{shift}[-1, \ldots, m] \)

\[ i = k = 0 \]

\begin{align*}
\text{while } i + m &\leq n \text{ do } \quad \text{Trace the value of } 2i + k! \\
\text{if } t[i + k] &= p[k] \text{ then } \\
&k = k + 1 \quad \left\{ \text{Value of } 2i + k \text{ increases!} \right. \\
\text{if } k &> m \text{ then return } i \\
\text{else} \\
&i = i + \text{shift}[k - 1] \\
&k = \max(k - \text{shift}[k - 1], 0) \quad \left\{ \text{Value of } 2i + k \text{ increases even here!} \right. \\
\text{return } -1
\end{align*}

How efficient is this? And how do we compute the shift table?
As we shall see, the shift table can be computed in time $O(m)$. This yields:

**Theorem**

*KMP-Search* returns the position of the first occurrence of $p$ in $t$ (and $-1$ if not such position exists) in time $O(m+n)$. 
Computing the Shift Table

How to compute the shift table?

We have to match (prefixes of) $p$ against $p$

$\Rightarrow$ run $\text{KMP-Search}(p, p)$ and record the matches in the shift table!
Knuth-Morris-Pratt Substring Search

Computing the Shift Table

\[ KMP-Shift(p) \text{ where } p = p[0, \ldots, m] \]
\[
\text{shift}[-1] = \text{shift}[0] = 1 \\
i = 1 \\
k = 0 \\
\text{while } i + k \leq m \text{ do} \\
\quad \text{if } p[i + k] = p[k] \text{ then } \leftarrow \text{found } p[0 \cdots k] = p[i \cdots i + k] \\
\quad \quad \text{shift}[i + k] = i \\
\quad \quad k = k + 1 \\
\quad \text{else} \\
\quad \quad \text{if } k = 0 \text{ then } \leftarrow \text{no match here at all} \\
\quad \quad \quad \text{shift}[i] = i + 1 \\
\quad \quad i = i + \text{shift}[k - 1] \leftarrow \text{is this OK???} \\
\quad \quad k = \max(k - \text{shift}[k - 1], 0) 
\]
Recalling Regular Expressions

Regular expressions over an alphabet $\Sigma$ consist of

- the letters in $\Sigma$ (we omit $\emptyset$ and $\epsilon$),
- the unary operation $^*$ (“Kleene star”), and
- the binary operations $|$ and $\cdot$.

A regular expression $E$ denotes a language $L(E) \subseteq \Sigma^*$ defined as follows:

<table>
<thead>
<tr>
<th>Expression $E$</th>
<th>Semantics $L(E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>${a}$</td>
</tr>
<tr>
<td>$E^*$</td>
<td>${u_1 \ldots u_n \mid n \geq 0, u_1, \ldots, u_n \in L(E)}$</td>
</tr>
<tr>
<td>$E</td>
<td>E'$</td>
</tr>
<tr>
<td>$E \cdot E'$</td>
<td>${uu' \mid u \in L(E), u' \in L(E')}$</td>
</tr>
</tbody>
</table>
Regular Expression Matching by Repeated Tree Traversal

Example: $E = a \cdot ((a \mid b^*) \cdot c)$ as a tree

- Matching $E$ to a text $t$ means to check whether $t = uvw$ with $v \in L(E)$.
- We do this by reading $t$ from left to right.
- Preprocessing: which subtrees match $\epsilon$?
- Match next symbol of $t$:
  $$match\_next(E, t[i], true)$$

May this be the first symbol matched by $E$?

Example: $adaabcc$
Regular Expression Matching by Repeated Tree Traversal

\[ \text{match\_next}(E, a, \text{first}) \] where \( E_1, E_2 \) are the children of \( E \) (if present)

- if \( E.\text{type} == \text{letter} \) then
  \[ E.\text{matched} = \text{first} \land E.\text{val} == a \]

- else if \( E.\text{type} == \cdot \) then
  \[ b = E_1.\text{matched} \lor (E_1.\text{epsilon} \land \text{first}) \]
  \[ \text{match\_next}(E_1, a, \text{first}) \]
  \[ E.\text{matched} = \text{match\_next}(E_2, a, b) \lor (E_1.\text{matched} \land E_2.\text{epsilon}) \]

- else if \( E.\text{type} == | \) then
  \[ E.\text{matched} = \text{match\_next}(E_1, a, \text{first}) \lor \text{match\_next}(E_2, a, \text{first}) \]

- else if \( E.\text{type} == * \) then
  \[ E.\text{matched} = \text{match\_next}(E_1, a, \text{first} \lor E_1.\text{matched}) \]

return \( E.\text{matched} \)
The matching procedure:

\[
\text{match}(E, t) \text{ where } t = T[0, \ldots, n]\\
\text{mark}_\text{epsilon}(E)\\
\text{if } E.\text{epsilon} \text{ then return true}\\
\text{for } i = 0, \ldots, n \text{ do}\\
\quad \text{if } \text{match}_\text{next}(E, t[i], true) \text{ then}\\
\quad \quad \text{return } true\\
\text{return } false
\]

Procedure \textit{match_next} traverses \( E \) once
⇒ running time \( O(mn) \), where \( m \) is the size of \( E \).
Please read Chapter 9 in the textbook.