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

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

Suppose the initial part of $p = \text{statistics}$ matches $t_{i...i+6}$, 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 5 positions.

\[
\begin{array}{ccccccc}
  t = & \cdots & i & \cdots & i+s & \cdots & i+k & \cdots \\
  p = & 0 & \cdots & s & \cdots & k & \cdots \\
end{array}
\]

$\Rightarrow$ if $p$ matched up to index $k \geq -1$ we can safely shift $p$ by

\[
\text{shift}[k] = \min\{s \geq 1 \mid p_{s...k} = p_{0...k-s}\}.
\]
The Knuth-Morris-Pratt Algorithm

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

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

\[ i \leftarrow k \leftarrow 0 \]

\[ \text{while } i \leq n - m \text{ do} \]

\[ \text{if } t[i + k] = p[k] \text{ then} \]

\[ k \leftarrow k + 1 \]

\[ \text{else} \]

\[ i \leftarrow i + \text{shift}[k - 1] \]

\[ k \leftarrow \max(k - \text{shift}[k - 1], 0) \]

\[ \text{return } -1 \]

How efficient is this? \( \Rightarrow \) at most \( 2n \) loop iterations

And how do we compute the shift table?
Running Time of *KMP-Search*

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 no such position exists) in time $O(m+n)$.

Note that this is a **uniform** algorithm: not only the text, but even the pattern is part of the input.
Computing the Shift Table

How to compute the shift table?

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

\[ \Rightarrow \text{run } KMP-Search(p, p) \text{ and record the matches in the shift table!} \]
Computing the Shift Table

\[ \text{KMP-Shift}(p) \text{ where } p = p[0, \ldots, m] \]

1. \[ \text{shift}[-1] \leftarrow \text{shift}[0] \leftarrow 1 \]
2. \[ i \leftarrow 1 \]
3. \[ k \leftarrow 0 \]
4. \[ \textbf{while } i + k \leq m \textbf{ do} \]
5. \[ \text{if } p[i+k] = p[k] \text{ then} \leftarrow \text{found } p[0 \cdots k] = p[i \cdots i+k] \]
6. \[ \text{shift}[i+k] \leftarrow i \]
7. \[ k \leftarrow k + 1 \]
8. \[ \text{else} \]
9. \[ \text{if } k = 0 \text{ then} \leftarrow \text{no match here at all} \]
10. \[ \text{shift}[i] \leftarrow i + 1 \]
11. \[ i \leftarrow i + \text{shift}[k-1] \leftarrow \text{is this OK???} \]
12. \[ k \leftarrow \text{max}(k - \text{shift}[k-1], 0) \]
Regular expressions over an alphabet \( \Sigma \) consist of

- the symbols 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 \cdots u_n \mid n \geq 0, \ u_1, \ldots, u_n \in L(E)} )</td>
</tr>
<tr>
<td>( E \mid E' )</td>
<td>( L(E) \cup L(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: \( E = a \cdot (((a \cdot b) \mid d^*) \cdot c) \) 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 \( \epsilon \)?
  - single symbols never do
  - \( E^* \) always does
  - \( E_1 \mid E_2 \) does if \( E_1 \) does or \( E_2 \) does
  - \( E_1 \cdot E_2 \) does if both \( E_1 \) and \( E_2 \) do
  \( \Rightarrow \) obvious recursive algorithm.
Example: \( E = a \cdot ((a \cdot b) \mid d^*) \cdot c \) as a tree

- Now we match \( t[1], t[2], \ldots \), one at a time.
- After iteration \( i \), the nodes we can end up in by reading \( t[1 \cdots i] \) will be marked.
- \( t[i + 1] \) can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: \( \text{next}(E, a, \text{boolean}) \).
- May this \( a \) start a new match?

Example: \( \text{adaabcc} \)
Regular Expression Matching by Repeated Tree Traversal

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

- Now we match $t[1], t[2], \ldots$, one at a time.
- After iteration $i$, the nodes we can end up in by reading $t[1 \cdots i]$ will be marked.
- $t[i + 1]$ can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: 
  $$\text{next}(E, a, \text{boolean}).$$
  May this $a$ start a new match?

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

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

- Now we match $t[1], t[2], \ldots$, one at a time.
- After iteration $i$, the nodes we can end up in by reading $t[1 \cdots i]$ will be marked.
- $t[i + 1]$ can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: $\text{next}(E, a, \text{boolean})$.

May this $a$ start a new match?

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

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

- Now we match \( t[1], t[2], \ldots \), one at a time.
- After iteration \( i \), the nodes we can end up in by reading \( t[1 \cdots i] \) will be marked.
- \( t[i + 1] \) can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: \( \text{next}(E, a, \text{boolean}) \).

Example: \( adaabcc \)
Regular Expression Matching by Repeated Tree Traversal

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

- Now we match $t[1], t[2], \ldots$, one at a time.
- After iteration $i$, the nodes we can end up in by reading $t[1 \cdots i]$ will be marked.
- $t[i + 1]$ can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: $\text{next}(E, a, \text{boolean})$.
  May this $a$ start a new match?

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

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

Now we match \( t[1], t[2], \ldots \), one at a time.

After iteration \( i \), the nodes we can end up in by reading \( t[1 \cdots i] \) will be marked.

\( t[i + 1] \) can be the first symbol of a new match or the continuation of an old one.

Procedure for matching the next symbol:
\[
\text{next}(E, a, \text{boolean}).
\]

May this \( a \) start a new match?

Example: \( adaabcc \)
Regular Expression Matching by Repeated Tree Traversal

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

- Now we match $t[1], t[2], \ldots$, one at a time.
- After iteration $i$, the nodes we can end up in by reading $t[1 \cdots i]$ will be marked.
- $t[i + 1]$ can be the first symbol of a new match or the continuation of an old one.
- Procedure for matching the next symbol: $\text{next}(E, a, \text{boolean})$.

May this $a$ start a new match?

Example: $adaabcc$
boolean next\( (E, a, \text{restartOK}) \) where \( E_1, E_2 \) are the children of \( E \) (if present)

\[
\begin{align*}
\text{if } & \ E.\text{type} \in \Sigma \text{ then} \\
& \ E.\text{mark} \leftarrow E.\text{type} = a \land \text{restartOK} \\
\text{else if } & \ E.\text{type} = | \text{ then} \\
& \ E.\text{mark} \leftarrow \text{next}(E_1, a, \text{restartOK}) \lor \text{next}(E_2, a, \text{restartOK}) \\
\text{else if } & \ E.\text{type} = * \text{ then} \\
& \ E.\text{mark} \leftarrow \text{next}(E_1, a, \text{restartOK} \lor E.\text{mark}) \\
\text{else if } & \ E.\text{type} = \cdot \text{ then} \\
& \ \text{boolean } b = E_1.\text{mark} \lor (E_1.\text{epsilon} \land \text{restartOK}) \\
& \ \text{next}(E_1, a, \text{restartOK}) \\
& \ E.\text{mark} \leftarrow \text{next}(E_2, a, b) \lor (E_1.\text{mark} \land E_2.\text{epsilon}) \\
\text{return } & \ E.\text{mark}
\end{align*}
\]

\textbf{Note:} Strict evaluation of \( \lor \) in case \( E.\text{type} = | \) is important!
Regular Expression Matching by Repeated Tree Traversal

The matching procedure:

```plaintext
match(E, t) where t = T[1, ..., n]

mark_\text{\textbackslash{epsilon}}(E)
if E.\text{\textbackslash{epsilon}} then return true
for i = 1, ..., n do
    if next(E, t[i], true) then
        return true
return false
```

Procedure next traverses $E$ once
$\Rightarrow$ running time $O(mn)$, where $m$ is the size of $E$. 