Applied Databases

Lecture 13
KMP, Boyer-Moore, and Horspool Algorithms

Sebastian Maneth

University of Edinburgh - March 6th, 2017
Outline

1. Morris-Pratt Algorithm
2. KMP
3. Boyer-Moore
4. Horspool
Recap: Naive Method

Given Pattern $P$ (length $m$) Text $T$ (length $n$) find all occurrences of $P$ in $T$.

$P = \begin{array}{cccc}
a & b & a & b & c \\
\end{array}$

$T = \begin{array}{cccccccc}
a & b & a & b & a & a & b & a & b & c & a & b & a & b & a & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
a & b & a & b & c \\
\end{array}$

(1) **Naive method**

$\rightarrow$ shift always by **one**

$\rightarrow$ worst-case complexity $m(n - m + 1)$

i.e., in $O(mn)$
Recap: Naive Method

Given Pattern P, Text T, find all occurrences of P in T.

\[ P = \begin{array}{cccc} a & b & a & b & c \end{array} \]

\[ T = \begin{array}{cccccccccccc} a & b & a & b & a & a & b & a & b & c & a & b & a & b & a & b & a & a & b & c \end{array} \]

\[ \begin{array}{cccc} a & b & a & b & c \end{array} \]

**Best-Case**  \( O(n) \)

**Average-Case**  \( \rightarrow \) show that if P and T are randomly chosen from an alphabet with d letters, then \#character-comparisons of Naive Algorithm is:

\[
(n - m + 1) \frac{1 - d^{-m}}{1 - d^{-1}} \leq 2(n - m + 1)
\]

**Questions**

Best-Case Complexity?

Average-Case Complexity? (on random strings)
Recap: Automaton Method

Given Pattern $P$ (length $m$) Text $T$ (length $n$) find all occurrences of $P$ in $T$.

$P = \begin{array}{c}
\text{a} \\
\text{b} \\
\text{a} \\
\text{b} \\
\text{c}
\end{array}$

$T = \begin{array}{c}
\text{a} \\
\text{b} \\
\text{a} \\
\text{b} \\
\text{a} \\
\text{a} \\
\text{b} \\
\text{a} \\
\text{b} \\
\text{a} \\
\text{b} \\
\text{a} \\
\text{b} \\
\text{a} \\
\text{a} \\
\text{b} \\
\text{c}
\end{array}$

$l_{ospre}(u, v) = \text{length of longest proper suffix of } u \text{ that is prefix of } v$

to state $l_{ospre}(ababa, P) = |aba| = 3$
Recap: Automaton Method

Given Pattern P (length $m$) Text T (length $n$) find all occurrences of $P$ in $T$.

$P = \text{a b a b c}$

$T = \text{a b a b a a b a b c a b a b a b a a b c}$

**Automaton method**

$\text{lonspre}(u, v) = \text{length of longest proper suffix of } u \text{ that is prefix of } v$

$|S| = \text{size of alphabet of } P$
→ for automata wo mismatch-transitions, $|S| = \text{size of alphabet of } T$ [large!!]

$\rightarrow O(m|S|) \text{ size & time to build}$
$\rightarrow O(n) \text{ matching time}$
1. Morris-Pratt Algorithm

Brief History:

→ 1970: James H. Morris built a text editor for the CDC 6400 computer

→ with Vaughan Pratt, developed “A linear pattern matching algorithm”
   [Report 40, University of California, Berkely, 1970]

→ Matching time: $O(n + m)$

→ rigorous analysis (Knuth) revealed: delay at a character can be $O(m)$

→ Knuth added one check to Morris&Pratt’s conditions, and was then able to prove logarithmic delay (tight bound)

→ #character comparisons is $\leq 2n - 1$
1. Morris-Pratt Algorithm

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5 \\
0 & 0 & 1 & 1 & 2 \\
\end{array}
\]

\[P = \begin{array}{cccc}
a & b & a & a & b \\
0 & 0 & 1 & 1 & 2 \\
\end{array}\]

\[
MP[k] = \text{lpropre}(P[1..k], P)
\]

\[MP[3] = \text{length of longest proper suffix of } aba, \text{ that is prefix of } abaab = |a| = 1\]

\[MP[5] = \text{length of longest proper suffix of } abaab, \text{ that is prefix of } abaab = |ab| = 2\]

Note: a suffix of a string \(w\) is \textit{proper}, if it is \textit{strictly shorter} than \(w\).
1. Morris-Pratt Algorithm

\[ P = \begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
a & b & a & a & b \\
0 & 0 & 1 & 1 & 2 \\
\end{array} \]

\[ T = \begin{array}{cccccccccccccccc}
a & b & b & a & b & a & a & a & b & a & a & b & a & b & a & b & a & a & b & c \\
a & b & a & a & b & a & a & a & b & a & a & b & a & b & a & a & b & a & b & a & a & b & c \\
\end{array} \]

\[ \text{MP}[k] = \text{lo spre}(P[1..k], P) \]

\[ \text{MP}[3] = |a| = 1 \]

\[ \text{MP}[5] = |ab| = 2 \]

Start matching with \textit{current lospree = 0}
1. Morris-Pratt Algorithm

\[ P = \begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
a & b & a & a & b \\
0 & 0 & 1 & 1 & 2 \\
\end{array} \]

\[ T = \begin{array}{cccccccccccccccccccc}
0 & 1 \\
a & b & b & a & b & a & a & a & b & a & a & b & a & b & a & b & a & b & a & b & a & b & c \\
\end{array} \]

\[ MP[ k ] = \text{lospre}( P[1..k], P ) \]

\[ MP[3] = |a| = 1 \]

\[ MP[5] = |ab| = 2 \]

Match!

\[ \rightarrow \text{increase lospre by 1} \]
1. Morris-Pratt Algorithm

\[ P = \begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
\begin{array}{ccccc}
a & b & a & a & b \\
0 & 0 & 1 & 1 & 2 \\
\end{array}
\end{array} \]

\[ MP[ k ] = \text{lospre}( P[1..k], P ) \]

MP[ 3 ] = \text{|}\text{a}| = 1
MP[ 5 ] = \text{|ab|} = 2

\[ T = \begin{array}{cccccccccccccccc}
0 & 1 & 2 \\
\begin{array}{cccccccccccccccc}
a & b & b & a & b & a & a & a & b & a & b & a & b & a & b & a & b & b & c \\
\end{array}
\end{array} \]

Match!

→ increase \text{lospre} by 1
1. Morris-Pratt Algorithm

\[ P = \begin{array}{cccc}
1 & 2 & 3 & 4 & 5 \\
abaaab \\
0 & 0 & 1 & 1 & 2
\end{array} \]

\[ T = \begin{array}{cccccccccccccccc}
a & b & b & a & b & a & a & a & b & a & a & b & a & b & a & b & a & b & a & b & c \\
abaaab
\end{array} \]

\[ MP[ k ] = \text{lospre}( P[1..k], P ) \]

\[ MP[ 3 ] = |a| = 1 \]

\[ MP[ 5 ] = |ab| = 2 \]

\[ \text{Mismatch!} \rightarrow \text{set current lospre to } MP[2] = 0 \]

\[ \rightarrow \text{continue matching at same position (do not advance)} \]
1. Morris-Pratt Algorithm

\[ T = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
\text{a b a a b} \\
\end{array} \]

\[ P = \begin{array}{cccc}
0 & 0 & 1 & 1 & 2 \\
\text{a b a a b} \\
\end{array} \]

\[ \text{MP}[k] = \text{lospre}(P[1..k], P) \]

\[ \text{MP}[3] = |a| = 1 \]

\[ \text{MP}[5] = |ab| = 2 \]

\[ T = \begin{array}{cccccccccccccc}
0 & 1 & 2 \\
\text{a b b a b a a a b a a b a b a b a b a b a b c} \\
\text{a b a a b} \\
\text{a b a a b} \\
\text{a b a a b} \\
\end{array} \]

Mismatch!

Since \text{lospre}=0, advance one letter to the right (and leave \text{lospre}=0)
Given Pattern $P$, Text $T$, find all occurrences of $P$ in $T$.

$Lopopre(P[1..j], P)$ will never be used.

Blue arrow = read current symbol again

Given Pattern $P$, Text $T$, find all occurrences of $P$ in $T$.

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
\hline
P = & a & b & a & a & b \\
& 0 & 0 & 1 & 1 & 2
\end{array}
\]

Lopopre's

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 4 & 5 \\
T = & a & b & b & a & b & a & a & a & b & a & b & a & b & a & b & a & b & a & b & a & ab & c
\end{array}
\]

Question

Time Complexity (for matching)?

Blue arrow = read current symbol again
Delay

\[ T = \begin{array}{ccccccc}
0 & 1 & 2 & a & b & b & a & b & a & a & a & b & a & b & a & b & a & b & a & b & a & b & a & b & c \\
\end{array} \]

MP[ k ] = _lospre_( P[1..k], P )

MP[ 3 ] = |a| = 1
MP[ 5 ] = |ab| = 2

delay = 2  (number of checks at this position)
Delay

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
\begin{array}{ccccc}
a & b & a & a & b \\
0 & 0 & 1 & 1 & 2
\end{array}
\end{array}
\]

\[MP[i] = \text{iospre}(P[1..i], P)\]

\[MP[3] = |a| = 1\]
\[MP[5] = |ab| = 2\]

\[
\begin{array}{cccccccccccccccc}
0 & 1 & 2 & a & b & b & a & b & a & a & a & b & a & b & a & b & a & b & a & b & c
\end{array}
\]

\[
\begin{array}{cccccccccccccc}
a & b & a & a & b & a & a & b & a & a & b & a & b & a & a & b & a & b & a & b & a & b & a & b & c
\end{array}
\]

\[\text{delay} = 2 \quad \text{(number of checks at this position)}\]

→ what is the maximum delay?
→ for which pattern \( P \)?
Maximum Delay

\[ P = \begin{array}{cccc} 1 & 2 & 3 & 4 \\ a & a & a & a \\ 0 & 1 & 2 & 3 \end{array} \]

\[ T = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ a & a & a & b \\ a & a & a & a \\ 2 \end{array} \]

mismatch: lospre = MP[3] = 2
Maximum Delay

\[ P = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
0 & 1 & 2 & 3 \\
\end{array} \]

\[ T = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\text{a} & \text{a} & \text{b} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
2 & 2 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
1 & 1 \\
\end{array} \]

mismatch: lospre = MP[3] = 2

mismatch: lospre = MP[2] = 1
Maximum Delay

P =

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

T =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

mismatch: lospre = MP[3] = 2

mismatch: lospre = MP[2] = 1

mismatch: lospre = MP[1] = 0
Maximum Delay

$$P = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
0 & 1 & 2 & 3 \\
\end{array}$$

$$T = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\text{a} & \text{a} & \text{a} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
2 & \text{a} & \text{a} & \text{a} \\
1 & \text{a} & \text{a} & \text{a} \\
0 & \text{a} & \text{a} & \text{a} \\
\end{array}$$

mismatch: \(\text{lospre} = MP[3] = 2\)

mismatch: \(\text{lospre} = MP[2] = 1\)

mismatch: \(\text{lospre} = MP[1] = 0\)

mismatch at \(\text{lospre} = 0\) → advance
### Delay in MP Algorithm

#### Example

Let's consider the following sequences:

**P** = `a a a a`  
**T** = `a a a a b`  

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P</strong></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T</strong></td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

- **Mismatch at lospre = 0**: MP[1] = 0  
  → advance

- **Mismatch at lospre = MP[1] = 0**: MP[2] = 1  
  → advance

  → advance


**Question:** How to decrease the delay?
2. Knuth-Morris-Pratt (KMP)

\[ T = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\text{a} & \text{a} & \text{a} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
\end{array} \]

\[ P = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
0 & 1 & 2 & 3 \\
\end{array} \]

\[ \rightarrow \text{ how to decrease the delay??} \]

from this check, we know \( T[3] \neq \text{“a”} \)

Thus: if next check-letter in \( P \) is “a”,
then mismatch
2. KMP

\[ P = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
0 & 1 & 2 & 3 \\
\end{array} \]

\[ T = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\text{a} & \text{a} & \text{a} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
2 & \text{a} & \text{a} & \text{a} \\
\end{array} \]

→ how to decrease the delay??

from this check, we know \( T[3] \neq \text{“a”} \)

Thus: if next check-letter in \( P \) is “a”, then mismatch

→ does next check-letter in \( P \) equal fail-letter in \( P \)?

If so, then KMP-entry should be smaller!!

(= bigger shift = smaller delay)
### 2. KMP

Mismatch at \textit{lospre} = 3

- \text{fail-letter} = 4
- \text{next check-letter} = \text{MP}[3] + 1 = 2 + 1 = 3


Thus: \textit{KMP}[3] should not equal “2”!
2. KMP

\[ P = \begin{array}{cccc}
1 & 2 & 3 & 4 \\
\text{a} & \text{a} & \text{a} & \text{a} \\
0 & 1 & ? & 3 \\
\end{array} \]

\[ T = \begin{array}{cccc}
0 & 1 & 2 & 3 \\
\text{a} & \text{a} & \text{a} & \text{b} \\
\text{a} & \text{a} & \text{a} & \text{a} \\
2 & & & \\
\text{a} & \text{a} & \text{a} & \text{a} \\
\end{array} \]

Mismatch at \text{lospre} = 3

\text{fail-letter} = 4
\text{next check-letter} = MP[3] + 1 = 2 + 1 = 3


Thus: \text{KMP}[3] should \textbf{not} equal “2”!

→ find longest \text{lospre}, such that \text{next-letter} \neq \text{fail-letter}

→ if does not exist, then mark “–1” = ADVANCE (and match with P[1])
   = no further check (delay)
2. KMP

\[
P = \begin{array}{ccccc}
  1 & 2 & 3 & 4 & 5 \\
  a & b & a & a & b \\
  0 & 0 & 1 & 1 & 2
\end{array}
\]

Previous table (Morris-Pratt)

\[
-1 & 1 & 2 & 3 & 4 & 5 \\
  a & b & a & a & b \\
  0 & -1 & 1 & 0 & 2
\]

Knuth-Morris-Pratt table

**KMP[ j ]** largest \( k \geq 0 \) such that \text{strong\_cond}(j,k) holds, or -1 if such \( k \) does not exist

**\text{strong\_cond}(j,k):** \( P[1..k] \) is a proper suffix of \( P[1..j] \) and \( P[k+1] \neq P[j+1] \)
2. KMP

Knuth-Morris-Pratt table

MP table

T = abbaaabaaaababaabaababc

no 2\textsuperscript{nd} check!

KMP[j] largest \( k \geq 0 \) such that strong\_cond(j,k) holds:

strong\_cond(j,k): \( P[1..k] \) is a proper suffix of pat[1..j] and \( P[k+1] \neq P[j+1] \)
2. KMP

Matching complexity of KMP: $O(m + n)$

→ what is the maximum delay for KMP?

→ in $O(\log m)$ [Knuth]

→ can actually occur (e.g., for Fibonacci strings)
2. KMP

Lemma
For KMP, \( \text{delay}(m) = O(\log m) \), and the bound is tight.

KMP1977 (p. 333)
Delay per scanned character is at most \( 1 + \log \phi m \), where \( \phi = (1 + \sqrt{5})/2 = 1.618... \) is the golden ratio.

1) Define Fibonacci strings:  \( f(1) = b, f(2) = a, f(n) = f(n-1)f(n-2) \)
   \( b, a, ab, aba, abab, abaababa, ... \)

<table>
<thead>
<tr>
<th>j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMP[j]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\[
\text{KMP}[F_k - 2] = F_{k-1} - 2
\]
Mismatch at position 19 = \( F_8 - 2 \), check at: 19, 11, 6, 3, 1, 0.
\( F_k = \text{round}(\phi^k / \sqrt{5}) \)  \( \rightarrow \text{delay}(m) = \log \phi(m) - 2. \)
3. Boyer-Moore

Robert S. Boyer and J. Strother Moore in 1977
3. Boyer-Moore

Idea 1  Match RIGHT-TO-LEFT in window

\[
T = \begin{array}{cccccccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c}
\end{array}
\]

first check
3. Boyer-Moore

Idea 1  Match RIGHT-TO-LEFT in window

\[ T = \begin{array}{ccccccccccccccc}
  a & b & a & b & a & a & b & a & b & c & a & b & a & b & a & b & a & a & b & c \\
  a & b & a & b & c \\
\end{array} \]

\[ R(a) = 2 \]

For each letter \( z \), let
\[ R(z) = \text{distance from right-most occurrence of } z \text{ in } P, \text{ to the end of } P \]
(and \(|P| \) if there is no occurrence)
3. Boyer-Moore

Idea 1: Match **RIGHT-TO-LEFT** in window

\[ T = \begin{array}{cccccccccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\end{array} \]

For each letter \( z \), let
\[ R(z) = \text{distance from right-most occurrence of } z \text{ in } P, \text{ to the end of } P \]
(and \( |P| \) if there is no occurrence)

\[ \rightarrow \text{ shift } R(a) \text{ to the right at mismatch with } "a" \]
(for all smaller shifts, we get a mismatch)
3. Boyer-Moore

**Idea 1**  Match **RIGHT-TO-LEFT** in window

$$T = \begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\end{array}$$

For each letter $z$, let

$R(z) = \text{distance from right-most occurrence of } z \text{ in } P, \text{ to the end of } P$

(and $|P|$ if there is no occurrence)

$R(a) = 2, \; R(b) = 1$

→ may shift $R(a)$ to the right at mismatch with “a”

(for all smaller shifts, we get a mismatch)
3. Boyer-Moore

Idea 1  Match **RIGHT-TO-LEFT** in window

\[ T = \begin{array}{cccccccccccccc}
  a & b & a & b & a & a & b & a & b & c & a & b & a & b & a & b & a & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
\end{array} \]
3. Boyer-Moore

Idea 1  Match **RIGHT-TO-LEFT** in window

\[
T = \begin{array}{cccccccccccc}
  a & b & a & b & a & b & a & b & c & a & b & a & b & a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
\end{array}
\]
3. Boyer-Moore

Idea 1  Match **RIGHT-TO-LEFT** in window

\[ T = \begin{array}{cccccccccccc}
  a & b & a & b & a & a & b & a & b & c & a & b & a & b & a & b & a & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
\end{array} \]

→ after **only 8 comparisons**, detects the first match!

→ compare this with the previous methods!
3. Boyer-Moore

Idea 1 Match **RIGHT-TO-LEFT** in window

$$T = \begin{array}{cccccccccccc}
  a & b & a & b & a & b & a & b & c & a & b & a & b & a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
  a & b & a & b & c \\
\end{array}$$

→ after **only 8 comparisons**, detects the first match!

→ compare this with the previous methods!


→ allows for sub-linear matching time wrt $n = |T|$
3. Boyer-Moore

Idea 1  
Match **RIGHT-TO-LEFT** in window

\[ T = \begin{array}{cccccccccccccccc}
  \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{c} \\
  \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
  \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
  \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
  \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\end{array} \]

→ after **only 8 comparisons**, detects the first match!

→ compare this with the previous methods!

→ letters \( T[1], T[2], T[3], \) and \( T[4] \) are never checked!

→ allows for **sub-linear matching time** wrt \( n = |T| \)

→ for natural language, almost always sublinear time!
3. Boyer-Moore

Idea 1  →  match RIGHT-TO-LEFT
        →  shift $R(z)$ to the right at mismatch

"Horspool algorithm"

→ How does Google Chrome search text (Ctrl^ F) on a page so quickly?

It uses a search algorithm inspired by Boyer-Moore and Boyer-Moore-Horspool

→ V8 string search package (chromium project)
```c
// Boyer-Moore-Horspool string search.

template <typename PatternChar, typename SubjectChar>
template <typename PatternChar, typename SubjectChar> StringSearch::BoyermooreHorspoolSearch(
  StringSearch<PatternChar, SubjectChar> * search,
  Vector<const SubjectChar> subject,
  int start_index)
```
3. Boyer-Moore

Idea 2

\[ T = \begin{array}{cccccccccccccccc}
\text{a} & \text{b} & \text{c} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\end{array} \]

→ how far can we shift?
3. Boyer-Moore

Idea 2

\[ T = \begin{array}{cccccc}
     a & b & c & b & c & a \\
     a & b & a & b & c & a \\
     a & b & a & b & c & a \\
     a & b & a & b & c & a \\
     a & b & a & b & c & a \\
     a & b & a & b & c & a \\
\end{array} \]

→ how far can we shift?

→ all of \(|P|\).
  because "bc" does not occur to the left in \(P\)

→ for every suffix \(u\) of \(P\), let \(D(u)\) be the distance to the next occurrence of \(u\) to the left (if none exists, then \(D(u) = |P|\))
3. Boyer-Moore

Idea 2

\[ T = \begin{array}{cccccccccccccc}
  a & b & c & b & c & a & b & a & b & c & a & b & a & b & a & b & a & b & c \\
\end{array} \]

\[ c & b & a & b & c \]

\[ c & b & a & b & c \]

→ \( D(bc) = 5 \)

→ now **not OK**, to shift by 5! Why??
3. Boyer-Moore

Idea 2

$T = \text{a b c b c a b a b c a b a b a b a b c}
\text{c b a b c}
\text{c b a b c}$

$\rightarrow D(bc) = 5$

$\rightarrow$ now **not OK**, to shift by 5! **Why??**

$\rightarrow$ a suffix of $u$ is a prefix of $P$!
3. Boyer-Moore

Idea 2

$$T = \begin{array}{cccccccccc}
\text{a} & \text{b} & \text{c} & \text{b} & \text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{c} & \text{b} & \text{a} & \text{b} & \text{c} \\
\text{c} & \text{b} & \text{a} & \text{b} & \text{c}
\end{array}$$

$$\rightarrow D(bc) = 5$$

$$\rightarrow \text{now not OK, to shift by 5! Why??}$$

$$\rightarrow \text{a suffix of } u \text{ is a prefix of } P!$$

$$\rightarrow \text{for every suffix } u \text{ of } P, \text{ let}$$

$$L(u) = \text{lospre}(u, P)$$
3. Boyer-Moore

Idea 2

if mismatch after $cu$ on symbol $z$, then
$k = \max( R(z), D(u) )$
$k = \min( k, |P|-L(u) )$
else
report occurrence of $P$
$k := |P|-L(u)$
shift by $k$

maximum shift
restrict by $l_{osep}$
3. Boyer-Moore

T = ababababc ababc ababc ababc ababc

if mismatch after \texttt{cu} on symbol \texttt{z}, then

\[ k = \max( R(z), D(u) ) \]
\[ k = \min( k, |P|-L(u) ) \]

else

report occurrence of \texttt{P}

\[ k := |P|-L(P) \]

shift by \texttt{k}

L(P) = 0, hence we shift by |P| = 5
if mismatch after \texttt{cu} on symbol \texttt{z}, then
\begin{align*}
k &= \max( R(\texttt{z}), D(\texttt{u}) ) \\
k &= \min( k, |P|-L(\texttt{u}) )
\end{align*}
else
\begin{align*}
&\text{report occurrence of } P \\
k &= |P|-L(P) \\
&\text{shift by } k
\end{align*}

L(P) = 0, hence we shift by \(|P| = 5\)
3. Boyer-Moore

if mismatch after \texttt{cu} on symbol \texttt{z}, then
\[
\begin{align*}
k &= \max( R(z), D(u) ) \\
k &= \min( k, |P|-L(u) )
\end{align*}
\]
else
report occurrence of \texttt{P}
\[
k := |P|-L(P)
\]
shift by \texttt{k}
if mismatch after \text{cu} on symbol \( z \), then
\[
k = \max( R(z), D(u) )
\]
\[
k = \min( k, |P| - L(u) )
\]
else
report occurrence of \( P \)
\[
k := |P| - L(P)
\]
shift by \( k \)
3. Boyer-Moore

if mismatch after \texttt{cu} on symbol \texttt{z}, then
\[ k = \max( R(z), D(u) ) \]
\[ k = \min( k, |P|-L(u) ) \]
else
\begin{verbatim}
report occurrence of P
k := |P|-L(P)
shift by k
\end{verbatim}
3. Boyer-Moore

if mismatch after cu on symbol z, then

k = max( \( R(z) \), \( D(u) \) )

k = min( k, \( |P|-L(u) \) )

else

report occurrence of P

k := \( |P|-L(P) \)

shift by k
if mismatch after \texttt{cu} on symbol \texttt{z}, then
\[ k = \max( R(z), D(u) ) \]
\[ k = \min( k, |P|-L(u) ) \]
else
\begin{itemize}
  \item report occurrence of \texttt{P}
  \item \( k := |P|-L(P) \)
\end{itemize}
shift by \( k \)
if mismatch after $cu$ on symbol $z$, then

$$k = \max( R(z), D(u) )$$

$$k = \min( k, |P|-L(u) )$$

else

report occurrence of $P$

$$k := |P|-L(P)$$

shift by $k$
3. Boyer-Moore

if mismatch after cu on symbol z, then
  
k = max( R(z), D(u) )
  
k = min( k, |P|-L(u) )
else
  report occurrence of P
  
k := |P|-L(P)
shift by k
3. Boyer-Moore

if mismatch after cu on symbol z, then
k = max( R(z), D(u) )
k = min( k, |P|-L(u) )
else
report occurrence of P
k := |P|-L(P)
shift by k

→ finished!
→ would shift k=2, but end of text reached
4. Horspool

Only Idea 1

→ match from right-to-left
→ at mismatch (with $z$): shift to $R(z)$

$R(z) =$ distance from right-most occurrence of $z$ in $P[1..m-1]$, to the end of $P$
($|P|$ if there is no occurrence)
For each letter \( z \), let

\[
R(z) = \text{distance from right-most occurrence of } z \text{ in } P[1..,m-1], \text{ to the end of } P
\]

(and \(|P|\) if there is no occurrence)

\[
R(c) = 5
\]
BM – Average Case

To find first occurrence $i$ of an arbitrary 5-letter word in an English text, it inspects on average

$(0.25 \times i)$

text symbols.
In Practise
Experimental Map

→ Random text (10MB) and random patterns
→ Tested all algorithms (KMP, BM, BDM, etc)
→ 32-bit machine (UltraSPARC)
→ Only 4 algorithms have a zone on the map.

Surprising:

→ Results on DNA are same as random for $|\Sigma|=4$
→ Results on English text are same as random for $|\Sigma|=16$ (!)

alphabet sizes
END
Lecture 13