Pre-processing for good suffix

• For a given $i$, let $k = |P| - i + 1$, i.e., the length of $P[i, |P|]$
  – Let $L(i) = j < |P|$ be the largest position such that $P[i, |P|]$ matches $P[j-k+1, j]$ and $P(j-k) \neq P(i-1)$
  – If no such $j$ exists, $L(i) = 0$

• Let $l(i)$ be the length of the largest suffix of $P(i, |P|)$ that is also a prefix

• These can be calculated in linear time (see Gusfield)
Position: 1  2  3  4  5  6  
String:  x  t  p  x  t  d  
$Z_i$:  0  0  2  0  0  0  
$L(i)$:  0  0  0  0  0  0  
$l(i)$:  0  0  0  0  0  0  

$R(x) = 4$
$R(t) = 5$
$R(p) = 3$
$R(d) = 6$
Using good suffix and bad character

• Using simple bad character and strong good suffix

• Bad character: shift $P$ by $\max(1, i - R(T(k)))$

• Good suffix:
  – If an occurrence of $P$ is found then shift $P$ by $|P| - l(2)$
  – Else if $i = |P|$ then advance $P$ by 1
  – Else if mismatch is at $i-1$ of $P$ and $L(i) > 0$ then shift $P$ by $|P| - L(i)$
  – Else shift $P$ by $|P| - l(i)$

• Shift by the max of these two rules
For current example

• All $l(i)$ and $L(i)$ are 0, hence Good Suffix rule becomes:
  
  – If $i = |P|$ and no match, advance by 1
  
  – otherwise if no match, advance by $|P|$
Align both strings at their beginning position and begin comparing from the last character of $P$

Comparisons: 1

$R(x) = 4; R(t) = 5; R(p) = 3; R(d) = 6$
If symbols don’t match, shift \( P \) by the max of the good suffix and bad character rules.

**Comparisons: 2**

\( R(p) = 3 \), hence bad character: shift \( \max(1,6-3)=3 \)

\( i = |P| \), hence good suffix: shift 1
Boyer-Moore

If symbols match, compare previous symbols

Comparisons: 3
Boyer-Moore

If symbols match, compare previous symbols

Comparisons: 4
Boyer-Moore

If symbols match, compare previous symbols

Comparisons: 5
If symbols match, compare previous symbols

Comparisons: 6
Boyer-Moore

If symbols match, compare previous symbols

Comparisons: 7
If $P$ is found, shift $P$ by $|P| - l(2)$, begin at end

Comparisons: 8

$R(x) = 4; R(t) = 5; R(p) = 3; R(d) = 6$
If symbols don’t match, shift $P$ by the max of the good suffix and bad character rules

Comparisons: 9

$R(t) = 5$, hence bad character: shift max(1,6-5)=1

$i = |P|$, hence good suffix: shift 1
Boyer-Moore

If symbols don’t match, shift $P$ by the max of the good suffix and bad character rules

Comparisons: 10

$R(p) = 3$, hence bad character: shift $\max(1,6-3)=3$

$i = |P|$, hence good suffix: shift 1
If symbols don’t match, shift $P$ by the max of the good suffix and bad character rules

**Comparisons: 11**

$R(s) = 0$, hence bad character: shift $\max(1, 6-0) = 6$

$i = |P|$, hence good suffix: shift 1
If symbols don’t match, shift $P$ by the max of the good suffix and bad character rules

Comparisons: 12

$R(t) = 5$, hence bad character: shift $\max(1, 6-5) = 1$  

$i = |P|$, hence good suffix: shift 1
Boyer-Moore

If symbols match, compare previous symbols

Comparisons: 13
If symbols match, compare previous symbols

Comparisons: 14
If symbols match, compare previous symbols

Comparisons: 15
If symbols match, compare previous symbols

Comparisons: 16
If symbols match, compare previous symbols

Comparisons: 17
Boyer-Moore

If $P$ is found, shift $P$ by $|P| - l(2)$, begin at end (past end... finished)

Comparisons: 17

vs. 42 for naive algorithm and 30 for Knuth-Morris-Pratt
Boyer Moore algorithm

- Current example does not show some parts of Boyer Moore that are de-emphasized in the text
- When using the good suffix for shifting with $L(i)$ or $l(i)$, some of string is already matched
- Without skipping already matched material, lots of duplicate effort
Good suffix rule (strong)

Illustration from Gusfield

<table>
<thead>
<tr>
<th>$T$</th>
<th>$x$</th>
<th>$t$</th>
</tr>
</thead>
</table>

$P$ before shift

| $P$ before shift | $z$ | $t'$ | $y$ | $t$ |

$P$ after shift

| $P$ after shift | $z$ | $t'$ | $y$ | $t$ |
Good suffix rule (strong)

Illustration from Gusfield

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>x</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P before shift</th>
<th></th>
<th>z</th>
<th>t'</th>
<th>y</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P after shift</th>
<th></th>
<th>z</th>
<th>t'</th>
<th>y</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>← skip →</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21
Algorithm in Gusfield Text

\[ k \leftarrow |P| \]

while \( k \leq |T| \)

\[ i \leftarrow |P| \]

\[ h \leftarrow k \]

while \( i > 0 \) and \( P(i) = T(h) \)

\[ i \leftarrow i - 1 \]

\[ h \leftarrow h - 1 \]

if \( i = 0 \)

found \( P \) ending at \( k \)

\[ k \leftarrow k + |P| - l'(2) \]

else

\[ k \leftarrow k + \max(\text{bad-char, good-suffix}) \]

22
More to keep track of

\[ k \leftarrow |P| \]

while \( k \leq |T| \)

\[ i \leftarrow |P| \]

\[ h \leftarrow k \]

while \( i > 0 \) and \( P(i) = T(h) \)

\[ i \leftarrow i - 1 \]

\[ h \leftarrow h - 1 \] ← may need to skip over some

if \( i = 0 \)

found \( P \) ending at \( k \)

\[ k \leftarrow k + |P| - l'(2) \] ← anything to skip over?

else

\[ k \leftarrow k + \max(\text{bad-char, good-suffix}) \] ← anything to skip over?