You are free to use these slides. If you do, please sign the guestbook (www.langmead-lab.org/teaching-materials), or email me (ben.langmead@gmail.com) and tell me briefly how you’re using them. For original Keynote files, email me.
Exact matching: slightly less naïve algorithm

\[ P: \text{word} \]

\[ T: \text{There would have been a time for such a word} \]

We match \( w \) and \( o \), then mismatch \((r \neq u)\)

Mismatched text character \((u)\) doesn’t occur in \( P \)

... since \( u \) doesn’t occur in \( P \), we can skip the next two alignments
Boyer-Moore

Use knowledge gained from character comparisons to skip future alignments that definitely won’t match:

1. If we mismatch, use knowledge of the mismatched text character to skip alignments
   “Bad character rule”

2. If we match some characters, use knowledge of the matched characters to skip alignments
   “Good suffix rule”

3. Try alignments in one direction, then try character comparisons in opposite direction
   For longer skips

Boyé-Moore: Bad character rule

Upon mismatch, let $b$ be the mismatched character in $T$. Skip alignments until (a) $b$ matches its opposite in $P$, or (b) $P$ moves past $b$.

Step 1:

T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

Step 2:

T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

Step 3:

T: G C T T C T G C T A C C T T T T G C G C G C G C G C G G A A
P: C C T T T T G C

(etc)
Boyer-Moore: Bad character rule

Step 1: $T: GCTTCTGC\textcolor{red}{T}ACCTTTTGCGCGCGCGCGGAA$

$P: C\textcolor{red}{T}TTTTC\textcolor{red}{G}C$

Step 2: $T: GCTTCTGCT\textcolor{red}{A}CCTTTTGCGCGCGCGCGGAA$

$P: C\textcolor{red}{T}TTTTTC\textcolor{red}{G}C$

Step 3: $T: GCTTCTGCT\textcolor{red}{A}CCTTTTTGCGCGCGCGCGGAA$

$P: C\textcolor{red}{T}TTTTT\textcolor{red}{G}C$

We skipped 8 alignments

In fact, there are 5 characters in $T$ we never looked at
As soon as $P$ is known, build a $|\Sigma|$-by-$n$ table. Say $b$ is the character in $T$ that mismatched and $i$ is the mismatch’s offset into $P$. The number of skips is given by element in $b$th row and $i$th column.

Gusfield 2.2.2 gives space-efficient alternative.
Boyer-Moore: Good suffix rule

Let $t$ be the substring of $T$ that matched a suffix of $P$. Skip alignments until (a) $t$ matches opposite characters in $P$, or (b) a prefix of $P$ matches a suffix of $t$, or (c) $P$ moves past $t$, whichever happens first.

**Step 1:**

$T: \text{CGTGCCTACTTTACCTTTACCTTTACGCGAA}$  
$P: \text{CTTACTTAC}$  
*Case (a)*

**Step 2:**

$T: \text{CGTGCCTACTTTACCTTTACCTTTACGCGAA}$  
$P: \text{CTTACTTAC}$  
*Case (b)*

**Step 3:**

$T: \text{CGTGCCTACTTTACCTTTACCTTTACGCGAA}$  
$P: \text{CTTACTTAC}$
Boyer-Moore: Good suffix rule

Like with the bad character rule, the number of skips possible using the good suffix rule can be precalculated into a few tables (Gusfield 2.2.4 and 2.2.5).

Rule on previous slide is the weak good suffix rule; there is also a strong good suffix rule (Gusfield 2.2.3).

With the strong good suffix rule (and other minor modifications), Boyer-Moore is $O(m)$ worst-case time. Gusfield discusses proof.
Boyer-Moore: Putting it together

After each alignment, use bad character or good suffix rule, whichever skips more

**Bad character rule:**
Upon mismatch, let \( b \) be the mismatched character in \( T \). Skip alignments until (a) \( b \) matches its opposite in \( P \), or (b) \( P \) moves past \( b \).

**Good suffix rule:**
Let \( t \) be the substring of \( T \) that matched a suffix of \( P \). Skip alignments until (a) \( t \) matches opposite characters in \( P \), or (b) a prefix of \( P \) matches a suffix of \( t \), or (c) \( P \) moves past \( t \), whichever happens first.

**Step 1:**
\[
T: \quad \text{GTTATAGC} \quad \text{T} \quad \text{GATCGCGGCGTAGCGGCGAA} \\
P: \quad \text{GTAGCGGCG} \\
bc: 6, gs: 0
\]

**Step 2:**
\[
T: \quad \text{GTTATAGCTG} \quad \text{ATCGCGGCGTAGCGGCGAA} \\
P: \quad \text{GTAGCGGCG} \\
bc: 0, gs: 2
\]

**Step 3:**
\[
T: \quad \text{GTTATAGCTGATCGCGGCGTAGCGGCGAA} \\
P: \quad \text{GTAGCGGCG} \\
bc: 2, gs: 7
\]

**Step 4:**
\[
T: \quad \text{GTTATAGCTGATCGCGGCGTAGCGGCGAA} \\
P: \quad \text{GTAGCGGCG} \\
\]

Part (a) of bad character rule

Part (a) of good suffix rule

Part (b) of good suffix rule
Boyer-Moore: Putting it together

Step 1:

T: GTTATAGCTGATCGCGGCGGTAGCGGCGAA
P: GTAGCGGGCG

Step 2:

T: GTTATAGCTGATCGCGGCGGTAGCGGCGAA
P: GTAGCGGGCG

Step 3:

T: GTTATAGCTGATCGCGGCGGTAGCGGCGAA
P: GTAGCGGGCG

Step 4:

T: GTTATAGCTGATCGCGGCGGTAGCGGCGAA
P: GTAGCGGGCG

Up to now: 15 alignments skipped, 11 text characters never examined
Boyer-Moore: Worst and best cases

Boyer-Moore (or a slight variant) is $O(m)$ worst-case time

What’s the best case?

Every character comparison is a mismatch, and bad character rule always slides $P$ fully past the mismatch

How many character comparisons? $\text{floor}(m / n)$

Contrast with naive algorithm
## Performance comparison

Comparing simple Python implementations of naïve exact matching and Boyer-Moore exact matching:

<table>
<thead>
<tr>
<th></th>
<th>Naïve matching</th>
<th>Boyer-Moore</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># character</td>
<td>wall clock</td>
</tr>
<tr>
<td>comparisons</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td><strong>P</strong>: “tomorrow”</td>
<td>5,906,125</td>
<td>2.90 s</td>
</tr>
<tr>
<td><strong>T</strong>: Shakespeare’s complete works</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P</strong>: 50 nt string from Alu repeat*</td>
<td>307,013,905</td>
<td>137 s</td>
</tr>
<tr>
<td><strong>T</strong>: Human reference (hg19) chromosome 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* GCGCGGGTGCTCACGCTGGAATCCCAAGCAGTCCAAGTGGAGGAGGGCGAGGCAGGCGGG

---

**Johns Hopkins University**

**Whiting School of Engineering**