**Boyer-Moore: Main ideas**

Longer shifts result from:
1. matching \( P \) against \( T \) right-to-left, in order \( P[n], P[n-1], \ldots \)
2. “bad character shift rule”
   - to avoid repeating unsuccessful comparisons against a mismatched target character
3. “good suffix shift rule”
   - to align only matching pattern characters against target characters already successfully matched

Either rule alone works, but they’re more effective together.

**Bad Character Rule Formally**

For each character \( x \in \Sigma \), let
\[
R(x) = \max \{ i < n \mid P[i] = x \} \cup \{0\}
\]
\( R(x) \): right-most occurrence of \( x \) in \( P[1 \ldots n-1] \), or 0

Easy to compute in time \( \Theta(|\Sigma| + |P|) \):
- for each \( x \in \Sigma \) do \( R(x) := 0 \);
- for \( i := 1 \) to \( n-1 \) do
  - \( R(P[i]) := i \);

**Bad Character Shift**

When \( P[i] \neq T[h] \) for \( i = x \), shift \( P \) to the right by \( \max\{1, i - R(x)\} \). This means:
- if the right-most occurrence of \( x \) in \( P[1 \ldots n-1] \) is at \( j < i \), chars \( P[j] \) and \( T[h] \) get aligned
- if the right-most occurrence of \( x \) in \( P[1 \ldots n-1] \) is at \( j > i \), the pattern is shifted to the right by one
- if \( x \) doesn’t occur in \( P[1 \ldots n-1] \), shift = 1, and the pattern is next aligned with \( T[h+1 \ldots h+n] \)

**Strong) Good Suffix Rule**

Bad character rule is effective, e.g., in searching natural language text (because mismatches are probable)
- Horspool’s version of BM applies the bad-char rule only

With a small alphabet, occurrences of \( x \) at \( P[j] \) for \( i < j < n \) are probable (\( \rightarrow i - R(x) < 0 \); doesn’t help)
- **Extended** bad-char rule helps by recording for each \( x \in \Sigma \) and each \( i \) the right-most occurrence of \( x \) in \( P[i \ldots i-1] \)

Additional benefit can be obtained from considering the **successfully matched suffix of \( P \)**

We concentrate to so called strong good suffix rule, which is more powerful than the original BM suffix rule.
**Good Suffix Rule Formally**

Suppose that \( P[i \ldots n] \) has been successfully matched against \( T \).

**Case 1:** If \( P[i-1] \) is a mismatch and \( P \) contains another copy of \( P[i \ldots n] \) which is not preceded by char \( P[i-1] \), shift \( P \) s.t. the closest such copy is aligned with the substring already matched by \( P[i \ldots n] \).

(See the previous slide for an example)

What if no preceding copy of \( P[i \ldots n] \) exists?

\(~\Rightarrow~\) Case 2

**Good Suffix Rule: Case 2**

Consider a mismatch at \( P[n-5] \):

\[
\begin{array}{c|c|c|c|c}
\hline
i & 1 & 2 & 3 & 4 \\
\hline
T & maitava talomaisena onalomainiuluun & maitava talomaisena onalomainiuluun & maitava talomaisena onalomainiuluun & maitava talomaisena onalomainiuluun \\
\hline
P & maisemaomalona & maisemaomalona & maisemaomalona & maisemaomalona \\
\hline
\end{array}
\]

No preceding occurrence of “aloma” in \( P \), but a potential occurrence of \( P \) begins at \( T[13] = “ma” \)

\[ i = 1 \]

\[ n = 3 \]

\[ T: maitava talomaisena onalomainiuluun \]

\[ P: maisemaomalona \]

**Case 2**

Assume that \( P[i \ldots n] \) has been successfully matched against substring \( r \) of the target.

\[ i = 2 \ldots n+1 \]

Define \( L'(i) \) as the largest position of \( P \) that satisfies the following:

\[ P[i',L'(i)] = P[i,n] \quad \text{thus } i' = L'(i) - n + i \]

\[ i' > 0 \]

\text{if no such copy of suffix } P[i \ldots n] \text{ occurs in } P, \text{ let } L'(i) = 0

**NB 1:** If \( L'(i) > 0 \), then \( P[i',L'(i)] \) is the closest copy of “good suffix” \( P[i \ldots n] \), and gives the shift \( n - L'(i) \)

**NB 2:** Since \( P[n+1 \ldots n+1] = \epsilon \), \( L(n+1) \) is the right-most position \( j \) s.t. \( P[j] \neq P[n] \) (or \( 0 \) if all chars are equal).

**Computing the \( L' \) Values (1)**

Define \( N_i(P) \) to be the length of the longest common suffix of \( P[1 \ldots j] \) and \( P \) (\( \Rightarrow 0 \leq N_i(P) \leq j \))

**Example:**

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
i & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\hline
P & 12345678901234 & 12345678901234 & 12345678901234 & 12345678901234 & 12345678901234 & 12345678901234 & 12345678901234 & 12345678901234 \\
N_i(P) & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 & N_i(P) = 0 \\
\hline
\end{array}
\]

**Computing the \( L' \) Values (2)**

Remember: \( Z \sim \text{longest repeat of prefix} \).

Now \( N_j \sim \text{longest common suffix} \) are reverses of \( Z \):

\[ N_j(P) = Z_{n-j+1}(P') \]

where \( P' \) is the reverse of \( P \)

**Example:**

\[
\begin{array}{c|c|c}
\hline
j & 1 & 2 & 3 & 4 \\
\hline
P & 1234567890123456 & 1234567890123456 & 1234567890123456 & 1234567890123456 \\
N_i(P) & N_1(1234567890123456) = 1234567890123456 & N_2(1234567890123456) = 1234567890123456 & N_3(1234567890123456) = 1234567890123456 & N_4(1234567890123456) = 1234567890123456 \\
\hline
\end{array}
\]

\(~\Rightarrow~\) the \( N_j \) values can be computed in time \( \Theta(|P|) \) by applying the \( Z \) algorithm to the reversal of \( P \).

**Computing the \( L' \) Values (3)**

**Theorem 2.2.2:** If \( L(i) > 0 \), then \( L'(i) = \max \{ j < n \mid N_j(P) = |P[i \ldots n]| \} \).

**Proof.** Such \( j \) is the right endpoint of the closest copy of \( P[i \ldots n] \) which is not preceded by \( P[i-1] \).

\(~\Rightarrow~\) the \( L'(i) \) values can be computed in \( \Theta(|P|) \) time by locating the largest \( j \) s.t. \( N_j(P) = |P[i \ldots n]| = n - i + 1 \)

\[ \Rightarrow \text{ such } j \text{ is } L'(i) \text{ for } i = n - N_j(P) + 1; \]

\[ i = n + 1 \Rightarrow L'(i) = 0 \]

\[ i = 1 \Rightarrow L'(n - N_j(P) + 1) = j; \]
How to compute the smallest shift that aligns a matching prefix of $P$ with a suffix of the successfully matched substring $t$ of $T (= P[1 \ldots n])$?

For $i \geq 2$, let $l(i)$ be the maximum length such that $P[1 \ldots l(i)]$ is equal to a suffix of $P[i \ldots n]$

**Example:** For $P = P[1 .. 5] = \text{“ababa”}$,
- $l(6) = 0$ ($\leftarrow P[6 .. 5] = \epsilon$),
- $l(5) = l(4) = 1$ (\text{“a”}), and
- $l(3) = l(2) = 3$ (\text{“aba”})

**Critical Points:**
- For $P[6 - 1]$ is a mismatch (after matching $P[1 \ldots n]$ successfully)
  - (Case 1) if $L(i) > 0$, shift the pattern to the right by $n - L(i)$ positions
  - (Case 2) if $L(i) = 0$, shift the pattern to the right by $n - l(i)$ positions

**Note:** If already $P[i]$ fails to match, $i = n + 1$, which also gives correct shifts.

When an occurrence of $P$ has been found, shift $P$ to the right by $n - l(2)$ positions. Why? To align a prefix of $P$ with the longest matching proper suffix of the occurrence.

**Final Remarks**

The presented rules avoid performing unnecessary comparisons that would fail.

They can be shown to lead to linear-time behavior, but only if $P$ does not occur in $T$. Otherwise the worst-case complexity is still $\Theta(nm)$.

A simple modification ("Galil rule"; Gusfield, Sect. 3.2.2) leads to a provably linear worst-case time.

On natural language texts the running time is typically sub-linear, and normally BM searches for longer patterns faster.