Horspool Algorithm

- The pattern is check with the text from right to left and progresses left to right through the text.
- Horspool's algorithm shifts the pattern by looking up shift value in the character of the text aligned with the last character of the pattern in table made during the initialization of the algorithm.
-
- Let *c* be the character in the text that aligns with the last character of the pattern. If the pattern does not match there are 4 cases to consider.

Horspool's Algorithm

The **mismatch occurs at the last character** of the pattern:
Case 1: ***c* does not exist** in the pattern (Not the mismatch occurred here) then shift pattern right the size of the pattern.

```
T[0] ...      S      ... T[n-1]
              |
         LEADER
               LEADER
```

Case 2: The mismatch happens at the last character of the pattern and ***c* does exist** in the pattern then the shift should be to the **right most *c*** in the *m*-1 remaining characters of the pattern.

```
T[0] ...      A      ... T[n-1]
              |
         LEADER
            LEADER
```

Horspool's Algorithm

The **mismatch happens in the middle** of the pattern:
Case 3: The mismatch happens in the middle (therefore *c* is in pattern) and there are **no other *c* in the pattern** then the shift should be the pattern length.

```
T[0] ...     MER       ... T[n-1]
             |
          LEADER
                LEADER
```

Case 4: The mismatch happens in the middle of the pattern but **there is other *c* in pattern** then the shift should be the **right most *c*** in the *m*-1 remaining characters of the pattern.

```
T[0] ...    EDER       ... T[n-1]
            |
          LEADER           .      .
        LEADER
```

Horspool's Algorithm

Algorithm 2.11: Horspool
Input: text $T = T[0 \ldots n)$, pattern $P = P[0 \ldots m)$
Output: position of the first occurrence of P in T
Preprocess:
(1) for $c \in \Sigma$ do $shift[c] \leftarrow m$
(2) for $i \leftarrow 0$ to $m-2$ do $shift[P[i]] \leftarrow m-1-i$

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       | |       |       ||   |
LEADER        | |       |       ||   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       | |       |       ||   |
LEADER        | |       |       ||   |
       LEADER |       |       ||   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       |  |       |       ||   |
LEADER        |  |       |       ||   |
       LEADER  |       |       ||   |
         LEADER       |       ||   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       |  |       |       ||   |
LEADER        |  |       |       ||   |
       LEADER  |       |       ||   |
         LEADER       |       ||   |
                LEADER       ||   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       |  |       |       ||   |
LEADER        |  |       |       ||   |
       LEADER  |       |       ||   |
         LEADER       |       ||   |
                LEADER       ||   |
                       LEADER|   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||       |  |       |       ||   |
LEADER        |  |       |       ||   |
       LEADER  |       |       ||   |
         LEADER       |       ||   |
                LEADER       ||   |
                       LEADER|   |
                        LEADER   |
```

Horspool's Algorithm

Text: JIMY_HAILED_THE_LEADER_TO_STOP
Pattern: LEADER

```
JIMY_RAN_AND_HAILED_THE_LEADER_TO_STOP
     ||      |  |      |      ||   |
LEADER       |  |      |      ||   |
       LEADER |  |      |      ||   |
         LEADER      |      ||   |
                LEADER      ||   |
                       LEADER|   |
                        LEADER   |
                           LEADER
```

The worst case cost is $\Theta(nm)$, but for random text is $\Theta(n)$.

Try Yourself

Text: **JIM_SAW_ME_IN_A_BARBER_SHOP**

Pattern: **BARBER**

Advanced Algorithm

String Matching

Horspool's Algorithm

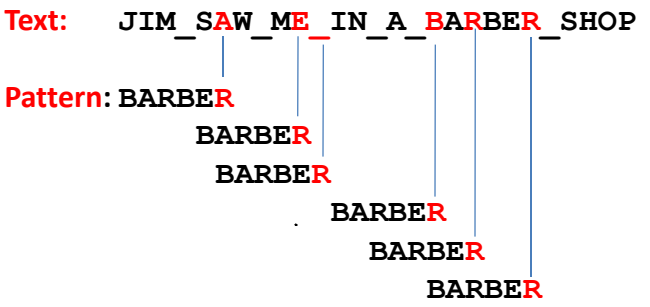
Consider, as an example, searching for the pattern BARBER in some text:

s_0 ... c ... s_{n-1}

B A R B E R

Starting with the last R of the pattern and moving right to left, we compare the corresponding pairs of characters in the pattern and the text. If all the pattern's characters match successfully, a matching substring is found. (Then the search can be either stopped altogether or continued if another occurrence of the same pattern is desired.) If, however, we encounter a mismatch, we need to shift the pattern to the right. Clearly, we would like to make as large a shift as possible without risking the possibility of missing a matching substring in the text. Horspool's algorithm determines the size of such a shift by looking at the character c of the text that was aligned against the last character of the pattern.

Try Yourself (Solution)



Horspool's Algorithm

Algorithm 2.11: Horspool
Input: text $T = T[0 \ldots n)$, pattern $P = P[0 \ldots m)$
Output: position of the first occurrence of P in T
Preprocess:
(1) for $c \in \Sigma$ do $shift[c] \leftarrow m$
(2) for $i \leftarrow 0$ to $m-2$ do $shift[P[i]] \leftarrow m-1-i$
Search:
(3) $j \leftarrow 0$
(4) while $j + m \leq n$ do
(5) if $P[m-1] = T[j+m-1]$ then
(6) $i \leftarrow m-2$
(7) while $i \geq 0$ and $P[i] = T[j+i]$ do $i \leftarrow i-1$
(8) if $i = -1$ then return j
(9) $j \leftarrow j + shift[T[j+m-1]]$
(10) return n