Strings
- A string is a sequence of characters
- Examples of strings:
  - Java program
  - HTML document
  - DNA sequence
  - Digitized image
- An alphabet $\Sigma$ is the set of possible characters for a family of strings
- Example of alphabets:
  - ASCII
  - Unicode
  - $\{0, 1\}$
  - $\{A, C, G, T\}$

Boyer-Moore Heuristics
- The Boyer-Moore's pattern matching algorithm is based on two heuristics
  - Looking-glass heuristic: Compare $P$ with a subsequence of $T$ moving backwards
  - Character-Jump heuristic: When a mismatch occurs at $T[i] = c$
    - If $P$ contains $c$, shift $P$ to align the last occurrence of $c$ in $P$ with $T[i]$
    - Else, shift $P$ to align $P[i]$ with $T[i+1]$
- Example

Brute-Force Algorithm
- The brute-force pattern matching algorithm compares the pattern $P$ with the text $T$ for each possible shift of $P$ relative to $T$, until either
  - A match is found, or
  - All placements of the pattern have been tried
- Brute-force pattern matching runs in time $O(nm)$
- Example of worst case:
  - $T = aaah \ldots ah$
  - $P = aaah$
  - May occur in images and DNA sequences
  - Unlikely in English text

Last-Occurrence Function
- Boyer-Moore's algorithm preprocesse the pattern $P$ and the alphabet $\Sigma$ to build the last-occurrence function $L$ mapping $\Sigma$ to integers, where $L(c)$ is defined as
  - The largest index $i$ such that $P[i] = c$ or
  - $-1$ if no such index exists
- Example:
  - $\Sigma = \{a, b, c, d\}$
  - $L(a) = 4$
  - $L(b) = 5$
  - $L(c) = 3$
  - $L(d) = 1$
- The last-occurrence function can be represented by an array indexed by the numeric codes of the characters
- The last-occurrence function can be computed in time $O(m + \sigma)$, where $m$ is the size of $P$ and $\sigma$ is the size of $\Sigma$
The Boyer-Moore Algorithm

Algorithm BoyerMooreMatch(T, P, j)

1. L ← lastOccurrenceFunction(P, Σ)
2. j ← m - 1
3. repeat
   4. if (T[j] = P[1])
      5. return i (match at i)
   6. else
      7. j ← j - 1
8. else
   9. character-jump
 10. i ← L[T[j]]
11. j ← m
12. until j ≥ 0
13. return -1 (no match)

Example

Case 1: j ≤ 1 + 1

Case 2: 1 + l ≥ j

Analysis

- Boyer-Moore’s algorithm runs in time \(O(m + n)\)
- Example of worst case:
  - \(T = \text{aa...a} \)
  - \(P = \text{baaa} \)
- The worst case may occur in images and DNA sequences but is unlikely in English text
- Boyer-Moore’s algorithm is significantly faster than the brute-force algorithm on English text

The KMP Algorithm - Motivation

- Knuth-Morris-Pratt’s algorithm compares the pattern to the text in left-to-right, but shifts the pattern more intelligently than the brute-force algorithm.
- When a mismatch occurs, what is the most we can shift the pattern so as to avoid redundant comparisons?
- Answer: the largest prefix of \(P[0..j]\) that is a suffix of \(P[1..j]\)

KMP Failure Function

- Knuth-Morris-Pratt’s algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.
- The failure function \(F[j]\) is defined as the size of the largest prefix of \(P[0..j]\) that is also a suffix of \(P[1..j]\).
- Knuth-Morris-Pratt’s algorithm modifies the brute-force algorithm so that if a mismatch occurs at \(P[j] \neq T[i]\) we set \(j ← F[j - 1]\).

The KMP Algorithm

- The failure function can be represented by an array and can be computed in \(O(m)\) time.
- At each iteration of the while-loop, either
  - \(i\) increases by one, or
  - the shift amount \(i - j\) increases by at least one (observe that \(F[j - 1] < j\)).
- Hence, there are no more than \(2m\) iterations of the while-loop.
- Thus, KMP’s algorithm runs in optimal time \(O(m + n)\)

Algorithm KMPMatch(T, P)

- \(F \leftarrow \text{failureFunction}(P)\)
- \(j \leftarrow 0\)
- while \(i < n\)
  - if \(T[j] = P[i]\)
    - return \(i\) (match)
  - else
    - if \(j = m - 1\)
      - return \(-1\) (no match)
    - else
      - if \(j > 0\)
        - \(j \leftarrow F[j - 1]\)
      - else
        - \(j \leftarrow j + 1\)
  - \(i \leftarrow i + 1\)
- return \(-1\) (no match)
The failure function can be represented by an array and can be computed in $O(m)$ time.

The construction is similar to the KMP algorithm itself.

At each iteration of the while-loop, either

- $i$ increases by one, or
- the shift amount $i-j$ increases by at least one (observe that $F(j-1) < j$)

Hence, there are no more than $2m$ iterations of the while-loop.

**Algorithm failureFunction**

```
F[0] ← 0
i ← 1
j ← 0
while i < m
    if P[i] = P[j]
        F[i] ← j + 1
        i ← i + 1
        j ← j + 1
    else if j > 0 then
        j ← F[j-1]
    else
        F[i] ← 0
        i ← i + 1
```

**Example**

Pattern Matching 13

```
1 2 3
a b a c
F[10] = 0
```

Pattern Matching 14

```
1 2 3 4 5
a b c a a
F[10] = 1
```

```
13 14 15 16 17 18 19
a b c a a
F[19] = 2
```