Strings

- A string is a sequence of characters.
- Examples of strings:
  - C++ 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 (used by C and C++)
    - Unicode (used by Java)
    - (0, 1)
    - (A, C, G, T)

Let $P$ be a string of size $m$.
- A substring $P[1..j]$ of $P$ is the subsequence of $P$ consisting of the characters with ranks between 1 and $j$.
- A prefix of $P$ is a substring of the type $P[i..j]$.
- A suffix of $P$ is a substring of the type $P[i..m-1]$.

Given strings $T$ (text) and $P$ (pattern), the pattern matching problem consists of finding a substring of $T$ equal to $P$.

Applications:
- Text editors
- Search engines
- Biological research

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:
- $T = aababah$
- $P = abacab$

The Brute-Force Match algorithm is defined as

```java
Algorithm BruteForceMatch(T, P):
Input text $T$ of size $n$ and pattern $P$ of size $m$
Output starting index of a substring of $T$ equal to $P$ or $-1$ if no such substring exists
for $i = 0$ to $m - 1$  
  { test shift $i$ of the pattern }
  if $j = n$
    while $j < m  
      if $j = m$
        return $j$ [match at $j$]
      else
        break [while loop (mismatch)]
    return -1 [no match anywhere]
else
  break [while loop (mismatch)]
```

Last-Occurrence Function

- Boyer-Moore’s algorithm preprocesses the pattern $P$ and the alphabet $\Sigma$ to build the last-occurrence function $L$ mapping $\Sigma$ to integers, where $L(i)$ is defined as
  - the largest index $i$ such that $P[i] = c$
  - $-1$ if no such index exists

Example:
- $\Sigma = \{a, b, c, d\}$
- $P = abaca$

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 $O(m + s)$, where $m$ is the size of $P$ and $s$ is the size of $\Sigma$. 
The Boyer-Moore Algorithm

The KMP Algorithm - Motivation

KMP Failure Function

The KMP Algorithm
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(P)**

```plaintext
F[0] ← 0
i ← 1
j ← 0
while i < m
if P[i] = P[j]
{we have matched $j+1$ chars}
F[i] ← j + 1
i ← i + 1
j ← j + 1
else if j > 0 then
{use failure function to shift $P$}
F[i] ← F[j-1]
else
F[i] ← 0 {no match}
i ← i + 1
```

**Example**

```
| P | a | b | c | a | b | c | a | b | c | a | b | a | a | b | a | c | a | b |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| F[j] | a | b | c | a | b | c | a | b | c | a | b | a | a | b |
```