Pattern Matching

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

Let $P$ be a string of size $m$
- A substring $P[i..j]$ of $P$ is the subsequence of $P$ consisting of the characters with ranks between $i$ and $j$
- A prefix of $P$ is a substring of the type $P[0..j]$
- 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:
  - $T = \text{aaa ... ah}$
  - $P = \text{aaah}$
  - may occur in images and DNA sequences
  - unlikely in English text

Brute Force-Complexity
- Given a pattern $M$ characters in length, and a text $N$ characters in length...
- Worst case: compares pattern to each substring of text of length $M$. For example, $M=5$.
  - Total number of comparisons: $M(N-M+1)$
  - Total number of comparisons: $M(N-M+1)$
  - Best case time complexity: $O(M)$
Brute Force-Complexity (cont.)

- Given a pattern M characters in length, and a text N characters in length...
- Best case if pattern not found: Always mismatch on first character. For example, M=5.
  1) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  2) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  3) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  4) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  5) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  6) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  7) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  8) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  9) AAAAAAAAAAAAAAAAAAAAAAAAAAH
  Total number of comparisons: N
  Best case time complexity: O(N)

Best case if pattern not found: Always mismatch on first character. For example, M=5.

Boyer-Moore's Algorithm (1)

- 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[1] \neq e
- If P contains e, shift P to align the last occurrence of e in P with T[i]
- Else, shift P to align P[0] with T[i+1]

Example

```
Case 1: 7 6 5 4 3 2 1
       | | | | | | |
T: a a a a a a a a

P: a a a a a
```

Case 2: 1 3 4 5 6

```
Case 2: 1 3 4 5 6
       | | | | | |
T: a a a a a a a a

P: a a a a a
```

Last-Occurrence Function

- Boyer-Moore's algorithm preprocesses the pattern P and the alphabet \( \Sigma \) to build the last-occurrence function \( \text{L} \) mapping \( \Sigma \) to integers, where \( \text{L}(c) \) is defined as:
  - the largest index \( i \) such that \( P[i] = c \) or
  - \(-1\) if no such index exists.
- Example:
  \( \Sigma = \{a, b, c, d\} \)
  \( P = \text{abcd} \)

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 \( O(m + \nu) \), where \( m \) is the size of \( P \) and \( \nu \) is the size of \( \Sigma \).

Boyer-Moore's Algorithm (2)

```
function BoyerMooreMatch(T, P, \Sigma)
L = lastOccurrence(F, \Sigma)
i = m - 1;
repeat
  if (T[i] = P[j])
    i++; j++;
  else
    case
    if (j == 0)
      r = \text{L}(P[0]);
      i = \text{L}(P[i]);
      j = m - 1;
    else
      r = \text{L}(P[j - 1]);
      i = i - r;
      j = j + \text{L}(P[0]);
    end
  until (i < 0);
return i < 0;
```

Example of worst case:

```
Case 1: 10 9 8 7 6 5 4 3 2 1
       | | | | | | |
T: a a a a a a a a

P: a a a a a
```

Analysis

- Boyer-Moore's algorithm runs in time \( O(m + \nu) \).
- Example of worst case:
  - \( T = \text{one...a} \)
  - \( P = \text{name} \)

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.
Knuth-Morris-Pratt’s algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself.

The failure function \( F(i) \) 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 \( j \leftarrow F(j - 1) \).

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 \( O(m + n) \).

Example

Function FailureFunction(P)

Function KMPMatch(T, P)

Pattern Matching 13

Pattern Matching 14

Pattern Matching 15

Pattern Matching 16