Knuth-Morris-Pratt & Boyer-Moore Algorithms

by Robert C. St.Pierre

Overview

- Notation review
- Knuth-Morris-Pratt algorithm
  - Discussion of the Algorithm
  - Example
- Boyer-Moore algorithm
  - Discussion of the Algorithm
  - Example
- Applications in automatic proving

Notation Review (1)

- T[1..n] – text to search, length n
  - Characters from Σ
- P[1..m] – pattern text, length m
  - Characters from Σ
  - m <= n
- Σ – finite alphabet
- δ – transition function in a FA

Notation Review (2)

- P_k – k-character prefix P[1..k] of P[1..m]
- w is a *prefix* of x, denoted $w \subset x$, if x = wy for some string $y \in \Sigma^*$
- w is a *suffix* of x, denoted $w \supset x$, if x = yw for some string $y \in \Sigma^*$
- ε - empty string

Notation Review (3)

- s – shift
  - $0 \leq s \leq n - m$
- If T[s+1..s+m] = P[1..m] then s is called a *valid shift* else s is called an *invalid shift*.
- The string-matching problem is to find all valid shifts.

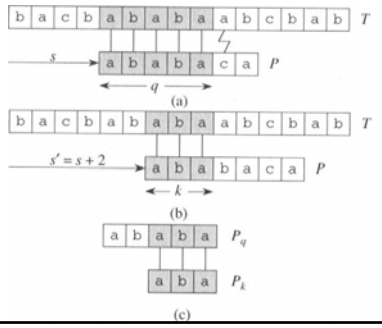
The Kunth-Morris-Pratt (KMP) Algorithm

- $\Theta(m)$ preprocessing time
  - Saves a factor of $|\Sigma|$ over the preprocessing time for the FA
    - Done by using an auxiliary function π, instead of the transition function δ.
- $\Theta(n)$ matching time
- Section 32.4 in text.

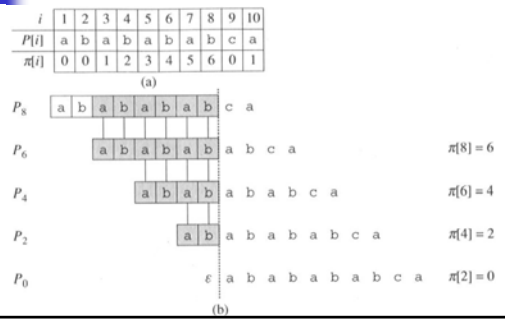
KMP Algorithm Discussion – Prefix Function (1)

"The prefix function π for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid testing useless shifts in the naive pattern matching algorithm or to avoid the precomputation of δ for a string-matching automation."

KMP Algorithm Discussion – Prefix Function (2)



KMP Algorithm Discussion – Prefix Function (3)



KMP Algorithm Discussion – Prefix Function (4)

Formally:

- Given a pattern P[1..m], the *prefix function* for the pattern P is the function π : {1, 2, ..., m} -> {0, 1, ..., m-1} such that $\pi[q] = \max\{k: k < q \text{ and } P_k \supset P_q\}$.
- π[q] is the length of the longest prefix of P that is a proper suffix of P_q.

KMP Algorithm Discussion – Pseudocode (1)

Compute-Prefix-Function(P)

```
1.  m <- length[T]
2.  π[1] <- 0
3.  k <- 0
4.  for q <- 2 to m
5.      do while k > 0 and P[k+1] != P[q]
6.            do k <- π[k]
7.         if P[k+1] = P[q]
8.            then k <- k + 1
9.         π[q] <- k
10. return π
```

KMP Algorithm Discussion – Pseudocode (2)

KMP-Matcher(T, P)

```
1.  n <- length[T]
2.  m <- length[P]
3.  π <- Compute-Prefix-Function(P)
4.  q <- 0                          ; Number of characters matched.
5.  for i <- 1 to n                 ; Scan the text from left to right
6.      do while q > 0 and P[q+1] != T[i]
7.            do q <- π[q]          ;Next character does not match.
8.         if P[q+1] = T[I]
9.            then q <- q + 1       ;Next character matches
10.        if q = m                 ;Is all of P matched?
11.           then print "Pattern occurs with shift" i – m
12.                q <- π[q]        ;Look for the next match
```

KMP Algorithm Discussion – Correctness (1)

- Lemma 32.5 (Prefix-function iteration lemma)
  - Let P be a pattern of length m with prefix function π. Then, for q = 1, 2, ..., m, we have $\pi^*[q] = \{k: k < q \text{ and } P_k \supset P_q\}$.
- Lemma 32.6
  - Let P be a pattern of length m, and let π be the prefix function for P. For q = 1, 2, ..., m, if $\pi[q] > 0$, then $\pi[q] - 1 \in \pi^*[q - 1]$.

KMP Algorithm Discussion – Correctness (2)

For $q = 2, 3, \ldots, m$, define the subset $E_{q-1} \subseteq \pi^*[q-1]$ by

$$\begin{aligned} E_{q-1} &= \{k \in \pi^*[q-1] : P[k+1] = P[q]\} \\ &= \{k : k < q-1 \text{ and } P_k \supset P_{q-1} \text{ and } P[k+1] = P[q]\} \text{ (by Lemma 32.5)} \\ &= \{k : k < q-1 \text{ and } P_{k+1} \supset P_q\} . \end{aligned}$$

Corollary 32.7

Let P be a pattern of length m, and let π be the prefix function for P. For $q = 2, 3, \ldots, m$,

$$\pi[q] = \begin{cases} 0 & \text{if } E_{q-1} = \emptyset , \\ 1 + \max\{k \in E_{q-1}\} & \text{if } E_{q-1} \neq \emptyset . \end{cases}$$

KMP Algorithm Examples (HTML based)

- http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/kpm-example.html
- http://www-igm.univ-mlv.fr/~lecroq/string/examples/exp8.html

KMP Algorithm Examples (Applets)

- http://www-igm.univ-mlv.fr/~lecroq/string/node8.html
- http://www-sr.informatik.uni-tuebingen.de/~buehler/BM/BM.html

The Boyer-Moore Algorithm

- "If the pattern P is relatively long and the alphabet Σ is reasonably large, then [this algorithm] is likely to be the most efficient string-matching algorithm."
- Matches right to left, unlike KMP.
- This algorithm is NOT in the second edition of our book, but is in section 34.5 of the first edition.

Boyer-Moore Algorithm Discussion – Matcher Function (1)

Boyer-Moore-Matcher(T, P, Σ)

```
1.  n <- length[T]
2.  m <- length[P]
3.  λ <- Compute-Last-Occurrence-Function(P, m, Σ)
4.  γ <- Compute-Good-Suffix-Function(P, m)
5.  s <- 0
6.  while s <= n – m
7.        do j <- m
8.           while j > 0 and P[j] = T[s+j]
9.                 do j <- j – 1
10.          if j = 0
11.             then print "Pattern occurs at shift s"
12.                  s <- s + γ[0]
13.             else s <- s + max(γ[0], j – λ[T[s+j]] )
```

Boyer-Moore Algorithm
Discussion – Matcher Function (2)

- The function Boyer-Moore-Matcher(T,P, Σ) "looks remarkably like the naive string-matching algorithm." Indeed, commenting out lines 3-4 and changing lines 12-13 to s <- s + 1, results in a version of the naive string-matching algorithm.
- The Boyer-Moore Algorithm uses the greater of two heuristics to determine how much to shift next by.

Boyer-Moore Algorithm
Discussion – Heuristic Bad (1)

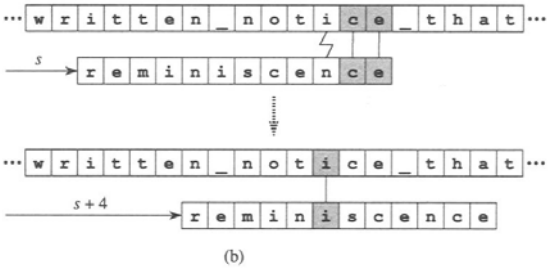
- The first heuristic, is the *bad-character heuristic*.
- In general, works as follows:
  P[j] != T[s+j] for some j, where 1<= j <= m. Let k be the largest index in the range 1 <= k <= m such that T[s+j] = P[k], if any such k exists. Otherwise let k = 0.
- We can safely increase by j – k, three cases to show this.

Boyer-Moore Algorithm
Discussion – Heuristic Bad (2)

- Case 1. k = 0, so increase by j.

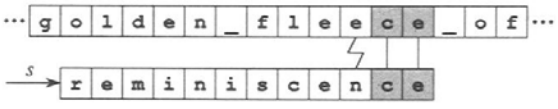
Boyer-Moore Algorithm
Discussion – Heuristic Bad (3)

- Case 2. k < j, so increase by j – k.



Boyer-Moore Algorithm
Discussion – Heuristic Bad (4)

- Case 3. k > j, resulting in a negative shift, but the good-suffix heuristic recommendation is ignored.



Boyer-Moore Algorithm
Discussion – Heuristic Bad (5)

Compute-Last-Occurrence-Function(P,m,Σ)

1. for each character a ∈ Σ
2. do λ[a] = 0
3. for j <- 1 to m
4. do λ[P[j]] <- j
5. return λ

- The running time of this procedure is O(|Σ| + m).

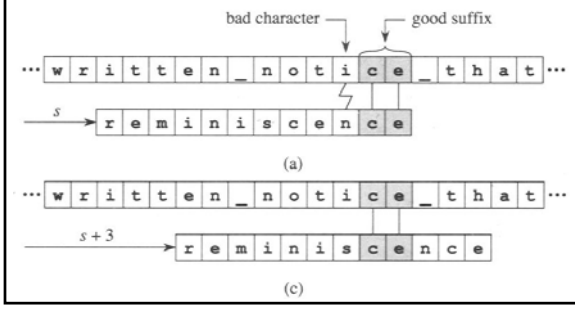
Boyer-Moore Algorithm Discussion – Heuristic Good (1)

- Define the relation Q ~ R for strings Q and R to mean that Q ⊃ R or R ⊃ Q.
- If two strings are similar, then we can align them with their rightmost characters matched, and no pair of aligned characters will disagree.
- The relation "~" is symmetric.
- Q ~ R and S ~ R imply Q ~ S

Boyer-Moore Algorithm Discussion – Heuristic Good (2)

- "If P[j] != T[s+j], where j < m, then the *good-suffix heuristic* says that we can safely advance by
  $\gamma[j] = m - \max\{k: 0 <= k < m \text{ and } P[j+1..m] \sim P_k\}$"
- "$\gamma[j]$ is the least amount we can advance s and not cause any characters in the "good suffix" T[s + j + 1..s + m] to be mismatched against the new alignment of the pattern."
- $\gamma[j] > 0$ for all j = 1..m, which ensures that this algorithm makes progress.

Boyer-Moore Algorithm Discussion – Heuristic Good (3)



Boyer-Moore Algorithm Discussion – Heuristic Good (4)

Compute-Good-Suffix-Function(P,m)

```
1.  π <- Compute-Prefix-Function(P)
2.  P' = reverse(P)
3.  π' <- Compute-Prefix-Function(P')
4.  For j <- 0 to m
5.        do γ[j] <- m – π[m]
6.  For i <- 1 to m
7.        do j <- m – π'[i]
8.           if γ[j] > i - π'[i]
9.              then γ[j] <- i - π'[i]
10. return γ
```

- This function has running time O(m).

Boyer-Moore Algorithm Discussion – Running Time

- Worst case is O((n – m + 1)m + |Σ|)
  - Compute-Last-Occurrence-Function takes time O(m + |Σ|).
  - Compute-Good-Suffix-Function takes time O(m).
  - O(m) time is spent validating each valid shift s.

Boyer-Moore Algorithm Examples (HTML based)

- http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html

Boyer-Moore Algorithm Examples (Applets)

- http://www-igm.univ-mlv.fr/~lecroq/string/node14.html
- http://www.blarg.com/~doyle/bmi.html
- http://www-sr.informatik.uni-tuebingen.de/~buehler/BM/BM.html

References

1. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. 2st Edition. MIT, 2001.
2. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*. 1st Edition. MIT, 1990.