String Matching

Find all occurrences of a pattern in a text

String Matching Problem:
Given text array $T[1..n]$ and pattern array $P[1..m]$ of characters from alphabet $\Sigma$, find all $s$ such that $T[s+1..s+m] = P[1..m]$, i.e., $P$ occurs with shift $s$ in $T$.

Example

<table>
<thead>
<tr>
<th>row</th>
<th>row</th>
<th>row</th>
<th>your</th>
<th>boat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$s = 12$

$\Sigma = \{a, b, o, r, t, u, w, y\}$

$T$: yoyoyoyo
$P$: yoyo

String Matching

- Simple problem with many applications
  - text editing
  - pattern recognition
• Algorithms

- Naive $O((n-m+1)m)$ worst case
- Rabin and Karp $O((n-m+1)m)$ worst case, but better on average
- Finite Automaton $O(n+m | \Sigma |)$
- Knuth-Morris-Pratt $O(n+m)$
- Boyer and Moore $O((n-m+1)m+| \Sigma |)$ worst case, but better (best overall) in practice

Naive String Matching

Naive(T, P)

\[
\begin{align*}
n &= \text{length(T)} \\
m &= \text{length(P)} \\
\text{for } s &= 0 \text{ to } n-m \quad O(n-m+1) \\
&\quad \text{if } P[1..m] = T[s+1..s+m] \quad O(m) \\
&\quad \text{then print } "\text{Pattern occurs with shift} \ s" \\
\end{align*}
\]

This algorithm takes $O((n-m+1)m)$ time.

However, there is more information in a failed match:

\[
\begin{array}{cccccccc}
T: & a & a & a & a & b & a & a & \ldots \\
\hline
P: & a & a & a & a & a & \\
\hline
\end{array}
\]

\[s = s + m\]

No need to consider ________________
Rabin-Karp Algorithm

- Let characters be digits in radix-$|\Sigma|$ notation.
- Choose a prime number $q$ such that $|\Sigma| q$ fits within a computer word to speed computations.

- Algorithm:
  Compute $(P \mod q)$
  Compute $(T[s+1, \ldots, s+m] \mod q)$ for $s = 0 \ldots n-m$
  Test against $P$ only those sequences in $T$ having the same (mod $q$) value
- $(T[s+1, \ldots, s+m] \mod q)$ can be incrementally computed by subtracting the high-order digit, shifting, adding the low-order bit, all in modulo $q$ arithmetic.

Example

\[ \Sigma = \{0, 1, \ldots, 9\} \]
\[ P = 12, \ P \mod 3 = 0 \]
\[ q = 3 \]

\begin{figure}[h]
\centering
\begin{array}{cccccccc}
5 & 5 & 3 & 1 & 2 & 2 & 7 & 3 & 1 \\
\hline
1 & 2 & 1 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\end{figure}
### Analysis

The Rabin-Karp algorithm takes $\Theta((n - m + 1)m)$ time in the worst case. 

$O(n) + O(m(v + n/q))$ average case, $v = \#\text{valid shifts}$

If $q \geq m$ and $v = O(1)$, then $O(n+m)$.

---

### Finite Automata

A finite automata $M = (Q, q_0, A, \Sigma, \delta)$, where

- $Q =$ set of states $(s_i)$
- $q_0 =$ start state $(s_0)$
- $A =$ set of accepting states
- $\Sigma =$ input alphabet
- $\delta =$ transition function $Q \times \Sigma \rightarrow Q$

---

### Example

Here is a finite automaton accepting strings with an even number of “a”s.

$\Sigma = \{a, b, c\}$.
\[ \delta(s_0, a) = s_1 \]
\[ \delta(s_0, b) = \delta(s_0, c) = s_0 \]
\[ \delta(s_1, a) = s_2 \]
\[ \delta(s_1, b) = \delta(s_1, c) = s_1 \]
\[ \delta(s_2, a) = s_1 \]
\[ \delta(s_2, b) = \delta(s_2, c) = s_2 \]

\[ A = \{s_2\} \]

Consider input string \( w \). If \( w \) ends at state \( s \in A \), then the FA accepts \( w \); otherwise, the FA rejects \( w \).

Example: \( \text{str} = \text{bccabaccaba} \)
Accept

---

**String Matching FA**

1. Compute FA accepting \( P \) (\( m+1 \) states)
2. Run FA with input string \( T \), printing shift whenever accepting state is reached.

**Example**

\[ P = \text{yoyo, m=4} \]
\[ T = \text{spin your yoyo} \]
\[ \Sigma = \{i, n, o, p, r, s, u, y\} \]
\[ \Sigma = \Sigma - \{y, o\} \]
Analysis

Computing $\delta$: $O(m|\Sigma|)$

$\text{FA-Matcher}(T, \delta, m)$ ; $O(n)$

$n = \text{length}(T)$

$s = s_0$

for $i = 1$ to $n$

$s = \delta(s, T[i])$

if $s = s_m$

then print “Pattern occurs with shift” (i-m)

This algorithm takes $O(n + m|\Sigma|)$ time.
Knuth-Morris-Pratt Algorithm

- Utilize a prefix array $\pi[1..m]$, where $\pi[q]$ contains information to compute $\delta(q, a)$ for ($a \in \Sigma$), the pattern shift for a mismatch on $P[q]$.

- $\pi$ requires only $O(m)$ time (as opposed to $O(m|\Sigma|)$ for $\delta$).

Prefix Array

Example

<table>
<thead>
<tr>
<th>n_e_y_o_y_o_d_y_n_e_y_o</th>
</tr>
</thead>
<tbody>
<tr>
<td>y_o_y_o_s</td>
</tr>
</tbody>
</table>

$s = 2$

How far can we shift $P$ over and be assured of catching all matches?

Since we have matched up to yoyo and yo is a suffix of yoyo, then we can shift over by 2 and start testing at $P[3]$.

Prefix Array

$\pi[q]$ answers the question:

If we have matched $P[1..q]$ in $T$, but $P[q+1]$ does not match, then what is the longest prefix of $P$, $P[1..k]$, that is a suffix of $P[1..q]$?

We can then start matching again from $P[k+1]$. 
\[ \pi[q] = \max\{k \mid k < q \text{ and } P[1..k] \text{ is a suffix of } P[1..q]\} \]

**Example**

\[
P = \begin{array}{cccc}
1 & 2 & 3 & 4 & 5 \\
y & o & y & o & s
\end{array}
\]

\[
\pi = \begin{array}{cccc}
1 & 2 & 3 & 4 & 5 \\
0 & 0 & 1 & 2 & 0
\end{array}
\]

**Pseudocode**

Compute-Prefix-Function(P)

\[
m = \text{length}(P) \\
\pi[1] = 0 \quad \text{; } \text{k must be less than } q \\
k = 0 \\
\text{for } q = 2 \text{ to } m \quad \text{; } O(m) \text{ amortized} \\
\quad \text{while } k > 0 \text{ and } P[k+1] \neq P[q] \\
\quad \quad k = \pi[k] \\
\quad \text{if } P[k+1] = P[q] \\
\quad \quad \text{then } k = k + 1 \quad \text{; prefix increased by one} \\
\quad \pi[q] = k \\
\text{return } \pi
\]

**Pseudocode**

KMP-Matcher(T, P)

\[
n = \text{length}(T)
\]
m = length(P)  
\(\pi = \text{Compute-Prefix-Function}(P)\) \(\); O(m) amortized  
q = 0  
for i = 1 to n \(\); O(n) amortized  
\(\text{while } q > 0 \text{ and } P[q+1] \neq T[i] \); where do we move to in P?  
\(q = \pi[q]\)  
if \(P[q+1] = T[i]\) \(\); matches so far  
then \(q = q + 1\)  
if \(q = m\)  
then print “Pattern occurs with shift” (i-m)  
\(q = \pi[q]\)  

This algorithm takes \_______\ time

---

**Boyer-Moore Algorithm**

- Most efficient (on average) when P is long and \(\Sigma\) is large
- Matches pattern from right to left
- Utilizes two heuristics

**Bad Character Heuristic**

Example
Good Suffix Heuristic

Example
Information For Bad Character Heuristic

Compute-Last-Occurrence(P, m \Sigma)

foreach a \in \Sigma
\lambda[a] = 0
for j = 1 to m
\lambda[P[j]] = j
return \lambda

Running time: O(| \Sigma | + m)

If mismatch at P[j] \neq T[s+j], then shift (j - \lambda[T[s+j]]).

Note: Shift could be negative, in which case ignore the shift value and use Good Suffix shift which always has a positive value.

Information for Good Suffix Heuristic

\gamma[j] = m - \max\{ k | 0 \leq k < m \text{ and } P[j+1..m] \sqsupset P_k \text{ or } P_k \sqsupset P[j+1..m]\} \\
\sqsupset \text{ means suffix (note: } x \sqsupset x \text{)}

If match j+1..m and P[j] \neq T[s+j], shift right \geq \gamma[j]

Examples

googoo

\[ \begin{array}{c}
3 & 3 & 3 & 3 & 3 & 1 & 1 \\
\end{array} \]

j = 0, P_3 \sqsupset P[1..6]

googo

\[ \begin{array}{c}
3 & 3 & 3 & 3 & 2 & 1 \\
\end{array} \]
Pseudocode

Compute-Good-Suffix(P, m)
π = Prefix(P)
P’ = reverse(P)
π’ = Prefix(P’)
for j = 0 to m ; O(m)
    γ[j] = m - π[m]
for l = 1 to m
    j = m - π’[l]
    if γ[j] > 1 - π’[l]
    then γ[j] = 1 - π’[l]
return γ

Example

m = 4
P = yoyo, π = ______
P’ = oyoy, π’ = ______
γ = ______
γ = ______

Boyer-Moore-Matcher

Boyer-Moore-Matcher(T, P, Σ)
n = length(T)
m = length(P)
λ = Compute-Last-Occurrence(P, m, Σ) ; O(|Σ| + m)
γ = Compute-Good-Suffix(P, m) ; O(m)
s = 0
while $s \leq n-m$ ; $O(n-m+1)$
    $j = m$
    while $j > 0$ and $P[j] = T[s+j]$ ; $O(m)$
        $j = j - 1$
    if $j = 0$
        then print “Pattern occurs with shift” $s$
        $s = s + \gamma[0]$
    else $s = s + \max(\gamma[j], j - \lambda[T[s+j]])$

Close to naive
$O((n-m+1)m + |\Sigma|)$
Boyer-Moore-Matcher is actually best in practice

Example

$T = \text{soyoyo}$
$P = \text{yoyo}$
$\gamma = \underline{\gamma}$
$\Sigma = \{0, s, y\}$
$\lambda = \underline{\lambda}$

\[
\gamma = \frac{2}{\text{yoyo}}
\]
Match

Applications