String Matching

- T = text
- P = pattern
- n = text length
- m = pattern length
- Z = alphabet size
Naïve Method

QUIZ: Complexity?
Knuth-Morris-Pratt Algorithm

Idea of Jumping

(a)

(b)
Jump Table

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>P[i]</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>π[i]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

KMP Algorithm

```
KMP-MATCHER(T, P)
1  n ← length[T]
2  m ← length[P]
3  π ← COMPUTE-PREFIX-FUNCTION(P)
4  q ← 0
5  for i ← 1 to n
6     do while q > 0 and P[q + 1] ≠ T[i]
7        do q ← π[q]
8     if P[q + 1] = T[i]
9        then q ← q + 1
10     if q = m
11        then print “Pattern occurs with shift” i – m
12        q ← π[q]
```
Computing Jumps

\[
\begin{array}{cccccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & P_q \\
\text{a} & \text{b} & \text{a} & P_k \\
\end{array}
\]

**Compute-Prefix-Function** \(P\)

1. \(m \leftarrow \text{length}(P)\)
2. \(\pi[1] \leftarrow 0\)
3. \(k \leftarrow 0\)
4. \(\text{for } q \leftarrow 2 \text{ to } m\)
5. \(\text{do while } k > 0 \text{ and } P[k+1] \neq P[q]\)
6. \(\text{do } k \leftarrow \pi[k]\)
7. \(\text{if } P[k+1] = P[q]\)
8. \(\text{then } k \leftarrow k + 1\)
9. \(\pi[q] \leftarrow k\)
10. \(\text{return } \pi\)

- Jumps can be computed by preprocessing the pattern, by comparing the pattern with itself.
- \(\pi[q]\) is the length of the longest prefix of \(P\), that is a proper suffix of \(P\).
- The worst-case complexity is \(O(m)\).

Complexity

- The running time for \text{Compute-Prefix-Function} is:
  - \(O(m)\)
- The total jump cannot exceed \((m+n)\). Thus complexity is \(O(m+n)\).
FSM for Detecting String Which Ends with Even 1s

Figure 34.5 A simple two-state finite automaton with state set $Q = \{0, 1\}$, start state $q_0 = 0$, and input alphabet $\Sigma = \{a, b\}$. (a) A tabular representation of the transition function $\delta$. (b) An equivalent state-transition diagram. State 1 is the only accepting state (shown blackened). Directed edges represent transitions. For example, the edge from state 1 to state 0 labeled $b$ indicates $\delta(1, b) = 0$. This automaton accepts those strings that end in an odd number of $a$'s. More precisely, a string $x$ is accepted if and only if $x = yz$, where $y = \varepsilon$ or $y$ ends with a $b$, and $z = ax^k$, where $k$ is odd. For example, the sequence of states this automaton enters for input $abaa$ (including the start state) is $(0, 1, 0, 1, 0, 1)$, and so it accepts this input. For input $ababa$, the sequence of states is $(0, 1, 0, 0, 1, 0)$, and so it rejects this input.
State Transition Diagram for String Matching

Complexity

- Fastest Compute State Transition is:
  - $O(mz)$
- Total WC running time is:
  - $O(n + mz)$
Boyer-Moore Algorithm
(1976)

Boyer-Moore Algorithm

• Comparing from the Right to Left:
  – in the pattern, each time, there is a mismatch, see how many position the pattern can be shifted left.

• More Look ahead in the Preprocessing
  – bring into consideration the character that caused the mismatch while considering what to do next.
Example

A string searching example consisting of:

STING
STING
STING
STING
STING
STING
STING

Complexity

- Boyer-Moore string search algorithm never uses more than M+N character comparisons, and uses about N/M steps of the alphabet is not small and the pattern is not long.
Computing Hash Value

\[ 14152 \equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \]
\[ = (7 - 3) \cdot 10 + 2 \pmod{13} \]
\[ = 8 \pmod{13} \]

RK Algorithm

\[
\text{Rabin-Karp-Matcher}(T, P, d, q) \\
1. \text{ } n = \text{length}[T] \\
2. \text{ } m = \text{length}([P]) \\
3. \text{ } h = d^{m-1} \pmod{q} \\
4. \text{ } p = 0 \\
5. \text{ } t_0 = 0 \\
6. \text{ for } i = 1 \text{ to } m \\
7. \text{ do } p = (dp + P[i]) \pmod{q} \\
8. \text{ } t_0 = (dt_0 + T[i]) \pmod{q} \\
9. \text{ for } s = 0 \text{ to } n - m \\
10. \text{ do if } p = t_s \\
11. \text{ then if } P[1..m] = T[s+1..s+m] \\
12. \text{ then “Pattern occurs with shift” } s \\
13. \text{ if } s < n - m \\
14. \text{ then } t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \pmod{q}
\]

- Radix is d. The prime is q.
Complexity

• In the worst case the running time is $O((n-m+1)m)$.
  – (case $T=a^n$ and $P=a^m$)
  – Each evaluation after the first one is $O(1)$ in text.

• Average Case Complexity
  – Only one match in most cases $O(1)$
  – Thus running time is $O(n+m)$. 