Backward nondeterministic DAWG matching

Slobodan Petrović

29.10.2013.
Contents

• Introduction – bit parallelism and search
• Shift algorithms – Shift-And and Shift-Or
• The BNDM algorithm
• The BNDMq algorithm
Introduction

• Deterministic Finite Automaton (DFA)
  – From any state and given a fixed input symbol, there is only one outgoing branch to the next state

• Nondeterministic Finite Automaton (NFA)
  – For the same input symbol, there can be more than one outgoing branch from each state
  – The machine makes copies of itself in such a situation – parallel processing
Introduction

• If, after performing such copy making and following one of the paths, an input symbol arrives that does not appear on any outgoing edge of the reached state, that machine is stopped – it becomes *inactive*.

• If any of the machines reaches the final state, the input string is accepted.
Introduction

• Any NFA can be transformed to a DFA
  – The general transformation algorithm has exponential complexity with respect to the length of the string determining the NFA
  – We, however, consider a simple special case, where this transformation can be performed in polynomial time
Introduction

• Our case (1)
  – Example – the search pattern is $w=\text{aabcc}$
  – The NFA – the classical case

  \[\text{Diagram:}
  \begin{tikzpicture}
  \node[state] (0) at (0,0) {$0$};
  \node[state] (1) at (1,0) {$1$};
  \node[state] (2) at (2,0) {$2$};
  \node[state] (3) at (3,0) {$3$};
  \node[state] (4) at (4,0) {$4$};
  \node[state] (5) at (5,0) {$5$};
  \draw (0) edge [loop above] node {$a$} (0);
  \draw (0) edge node {$a$} (1);
  \draw (1) edge node {$b$} (2);
  \draw (2) edge node {$c$} (3);
  \draw (3) edge node {$c$} (4);
  \draw (4) edge [loop above] node {$a$} (4);
  \end{tikzpicture}\]

  – Each input symbol drives creation of a new automaton, which starts at the state 0
Introduction

• Our case (2)

  – Example – the search pattern is \( w = aabcc \)
  – The NFA – the case with \( \varepsilon \)-transitions

  \[
  \begin{array}{cccccc}
  & 1 & \varepsilon & 2 & \varepsilon & 3 & \varepsilon & 4 & \varepsilon & 5 \\
 0 \quad a & 1 \quad a & 2 \quad b & 3 \quad c & 4 \quad c & \varepsilon \\
  \end{array}
  \]

  – \( \varepsilon \)-transitions – the automaton makes transitions without consuming any input
  – Recognizes all the suffixes of the pattern \( w \)
Introduction

- Our case (3)
  - Obviously, since this NFA recognizes all the suffixes of $w$, the corresponding DFA is the DAWG
Introduction

• Bit parallelism (1)
  
  – Suppose we are given the search string $S=aaabcaabcc$ and we are searching for the pattern $w=aabcc$ in it by means of the classical NFA (parallel processing of the symbols of $S$)

  ![Diagram](image)

  – Each time a symbol from $S$ arrives, the machine makes a copy of itself and starts from the 0 state
Introduction

• Bit parallelism (2)
  – Suppose the maximum number of machines running in parallel is \( m=|w| \), in our example \( m=5 \)
  – We denote by \( j \) the number of processed symbols from \( S \)
    • Then we have \( \min(j,m) \) machines running in parallel, for each \( j \)
  – After processing \( j \) symbols from \( S \), some of these machines are active and some are inactive
Introduction

• Bit parallelism (3)
  – Define *search status* in a computer word $D$ of $m$ bits
    • In our case $d_5d_4d_3d_2d_1$
  – We set $d_i=1$ if the corresponding machine is active after processing $j$ bits of $S$
  – Before processing any symbol, all the machines are active (since they are in the state 0) and consequently $D=1^m$ (all ones)
Introduction

• Bit parallelism (4)

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>d</th>
<th>i</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabbcc</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aabbcc</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>aabbcc</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>aabbcc</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>aabbcc</td>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ j=5 \quad j=6 \]
Introduction

• Bit parallelism (5)
  – The fact that the machine corresponding to the bit $d_5$ disappears when passing from $j=5$ to $j=6$ is expressed by shifting the word $D$ one position to the left
  – The fact that a new machine starting at the state 0 (which is always active) appears when passing from $j=5$ to $j=6$ is expressed by OR-ing the shifted word $D$ with 00001
Introduction

• Bit parallelism (6)
  – In our example, when passing from $j=5$ to $j=6$, what will the next symbol be?
    • Either a, b, or c
  – Which input symbol will keep which machine active, if it was active before that?
    • An a will keep the machine $d_1$ active, an a will keep the machine $d_2$ active, a b will keep the machine $d_3$ active, a c will keep the machine $d_4$ active, a c will keep $d_5$ active
Introduction

• Bit parallelism (7)
  – It is always like that, for any $j$
    • The new machine $d_1$, which is in the state 0, always expects an a, the machine $d_2$ always expect an a, the machine $d_3$ always expects a b, the machine $d_4$ always expects a c and the machine $d_5$ always expects a c
  – We can use this fact for updating the search status word $D$ automatically, after each shift and OR-ing with 1, by introducing the *bit masks*
Introduction

- Bit parallelism (8)
  - The bit mask for any symbol only depends on the search pattern, not on the search string
  - Because of that, we can pre-compute them
  - Given the search pattern \( w = w_1 w_2 \ldots w_m \), for the bit mask \( B[s] = b_1 b_2 \ldots b_m \) the following holds
    - If \( s = w_i \) then \( b_{m+1-i} = 1 \), otherwise \( b_{m+1-i} = 0 \)
Introduction

• Bit parallelism (9)
  – In our example, since $w=aabcc$, it is easy to see that
    • $B[a]=00011$
    • $B[b]=00100$
    • $B[c]=11000$
  – We can now update the search status word $D$ for each new input symbol $S_j$ in the following way
    • $D_j=((D_{j-1}<<1) \text{ OR } 1) \text{ AND } B[S_j]$
Introduction

• Bit parallelism (10)
  
  – In our example, for \( j = 6 \), we have

  \[
  D_6 = ((D_5 << 1) \text{ OR } 1) \text{ AND } B[S_6] = \\
  = ((01000 << 1) \text{ OR } 00001) \text{ AND } B[a] = \\
  = 10001 \text{ AND } 00011 = 00001
  \]
Shift algorithms

• Shift-And (Baeza-Yates, Gonnet, 1992)
  – We use the same formula as before for updating the search status word $D$
    • $D_j=(((D_{j-1}<<1) \text{ OR } 1) \text{ AND } B[S_j]$
  – A match is reported if $d_m=1$, for some $j$
    • That would mean that the machine $d_m$ has processed $m$ symbols from $S$ and is still active, i.e. it has reached the final state
Shift algorithms

• Shift-Or (1)
  – Very similar to Shift-And
  – Just complement the bit masks for each symbol and the search status word $D$
    • $d_i=0$ means then an active machine
  – In that case, OR-ing with 1 is not necessary
  – The status word update formula is
    • $D_j=(D_{j-1}<<1) \ OR \ B[S_j]$
Shift algorithms

- **Shift-Or (2)**
  - A match is reported if $d_m=0$, for some $j$
  - More efficient than Shift-And, since there is just one logical operation (OR) after shifting
The BNDM algorithm

• Simulates the BDM algorithm using bit parallelism (Navarro, Raffinot, 2000)

• Specific properties
  – We have a window moving forward and the search in the window is performed backwards
  – Since the NFA here has \( \epsilon \) transitions (i.e. there is no loop in the state 0), it eventually runs out of active states and then the window shift occurs, just like in BDM
The BNDM algorithm

• The match is reported if $d_m=1$
• The variable $\textit{last}$ corresponds to the longest prefix of $w$ matched in $S$, the same way as in BDM
• The formula for updating the search status word is

\[
D_j = (D_{j-1} \text{ AND } B[S_j]) << 1
\]
The BNDM algorithm

• Explanation of the update formula (1)
  – Only an active machine can receive a new symbol, no input symbol can re-activate any machine
  – The next input symbol determines on which *active machine* it will be possible to try to process it

• The bit mask determines which machine can process the input symbol (those having the corresponding 1 in the bit mask, i.e. $b_i=1$)

• To remain active, a machine must previously be active ($d_i=1$) and the bit mask for that machine must be 1 (therefore AND in the formula)
The BNDM algorithm

• Explanation of the update formula (2)
  – The shift to the left by one reflects the fact that we scan the window backwards

• Since the search in the window goes backwards, the bit masks have reverse order of bits compared to Shift-And and Shift-Or
BNDM - Backward nondeterministic DAWG Matching

**Input:** The search string $S$ and the search pattern $w$

**Output:** Position of the search pattern (if found) in the search string

**begin**

// Preprocessing

$n \leftarrow |S|; m \leftarrow |w|;$

for $c \in \Sigma$ do

$B[c] \leftarrow 0^m;$

end

for $i \leftarrow 1$ to $m$ do

$B[w_i] \leftarrow B[w_i] \mid 0^{i-1}10^{m-i};$

end

// Search

$i \leftarrow 0;$

while $i \leq n - m$ do

$j \leftarrow m; \text{last } \leftarrow m; D \leftarrow 1^m;$

while $D \neq 0^m$ do

$D \leftarrow D \& B[S_{i+j}];$

$j \leftarrow j - 1;$

if $D \& 10^{m-1} \neq 0^m$ then

if $j > 0$ then

last $\leftarrow j;$

end

else

Report an occurrence of $w$ at the position $i + 1$ in $S;$

end

end

$D \leftarrow D \ll 1;$

end

$i \leftarrow i + \text{last};$

end

end
The BNDM$q$ algorithm

- A small modification (Durian et al., 2009)
  - Instead of a single symbol, a $q$-gram is read (i.e. $q$ symbols) before testing the search state vector $D$
  - The search state word is first set to $F(i,q)$, where $i$ is the position of the first symbol in the $q$-gram
    \[ F(i,q) = B[S_i] \text{ AND } (B[S_{i+1}]<<1) \text{ AND } \ldots \text{ AND } (B[S_{i+q-1}]<<(q-1)) \]
  - The update formula for $D$
    - $D_j = (D_{j-1}<<1) \text{ AND } B[S_j]$