A bit-vector SHIFT-and-OR approach to exact pattern matching

\[ x = bbbacbbbabababacaabbba \]
“Sprint rather than plod!”

• An algorithm that does **not** attempt to reduce the number of comparisons
  - Essentially the “simple” $O(|p||x|)$ algorithm

• Instead the trick is to make each comparison very fast
Using a “state vector” $s$

We define a vector $s$ (depends on $i$) – the state of matching so far – by:

$$s[j] = 0 \text{ iff } x[i-j+1 .. i] = p[1 .. j]$$
Using a “state vector” \( s \)

Notice that \( s \) holds information about more than one comparison!

Conceptually, \( p \) is positioned \(|p|\) places along \( x \): \( s \) tries to match \( p \) at positions \( i-|p|+1 \ldots i \)
Using a “state vector” $s$

When $s[|p|]==0$ we have an occurrence of $p$ in $x$ at $i-|p|+1$
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 0 \]

\[ p = \text{bbba} \quad s[1] == 1 \]

\[ p = \text{bbba} \quad s[2] == 1 \]

\[ p = \text{bbba} \quad s[3] == 1 \]

\[ p = \text{bbba} \quad s[4] == 1 \]

\[ s = 1111 \]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ p = \text{bbba} \quad \text{if } s[1] == 0 \]
\[ p = \text{bbba} \quad \text{if } s[2] == 1 \]
\[ p = \text{bbba} \quad \text{if } s[3] == 1 \]
\[ p = \text{bbba} \quad \text{if } s[4] == 1 \]

\[ s = 0111 \]
Example

\[ i = 2 \]

\[ x = \text{bbbacbbbababacacabbbba} \]

\[ p_1 = \text{bbba} \quad s_1 = 0 \]

\[ p_2 = \text{bbba} \quad s_2 = 0 \]

\[ p_3 = \text{bbba} \quad s_3 = 1 \]

\[ p_4 = \text{bbba} \quad s_4 = 1 \]

\[ s = 0011 \]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 3 \]

\[ p = \text{bbba} \]

\[ s[1] == 0 \]

\[ p = \text{bbba} \]

\[ s[2] == 0 \]

\[ p = \text{bbba} \]

\[ s[3] == 0 \]

\[ p = \text{bbba} \]

\[ s[4] == 1 \]

\[ s = 0001 \]
Example

\[ \begin{align*}
  i &= 4 \\
  x &= \text{bbbacbbbababacabbbba} \\
  p_1 &= \text{bbba} \quad s_1[1] = 1 \\
  p_2 &= \text{bbba} \quad s_2[2] = 1 \\
  p_3 &= \text{bbba} \quad s_3[3] = 1 \\
  p_4 &= \text{bbba} \quad s_4[4] = 0 \\
  s &= 1110
\end{align*} \]

Match at \( i-4+1 = 1 \)
Example

\[ x = \text{bbbac} \text{cbbbababacabbbba} \]
\[ p = \text{bbba} \]
\[ s[1] == 1 \]
\[ p = \text{bbba} \]
\[ s[2] == 1 \]
\[ p = \text{bbba} \]
\[ s[3] == 1 \]
\[ p = \text{bbba} \]
\[ s[4] == 1 \]
\[ s = 1111 \]
Example

\[i = 6\]

\[x = \text{bbbacbbbababacabbbba}\]

\[p = \text{bbba}\]

\[s[1] = 0\]

\[p = \text{bbba}\]

\[s[2] = 1\]

\[p = \text{bbba}\]

\[s[3] = 1\]

\[p = \text{bbba}\]

\[s[4] = 1\]

\[s = 0111\]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ p = \text{bbba} \quad s[1] == 0 \]

\[ p = \text{bbba} \quad s[2] == 0 \]

\[ p = \text{bbba} \quad s[3] == 1 \]

\[ p = \text{bbba} \quad s[4] == 1 \]

\[ s = 0011 \]
Let $s^i$ be the state vector in iteration $i$, then

$$s^i[j] = s^{i-1}[j-1] \text{ OR } t$$

where $t = 0$ if $p[j] = x[i]$ and $t = 1$ otherwise.
Let \( s^i \) be the state vector in iteration \( i \), then

\[
s^i[j] = s^{i-1}[j - 1] \text{ OR } t
\]

where \( t = 0 \) if \( p[j] = x[i] \) and \( t = 1 \) otherwise.
Special cases...

• For this to work:
  - $s^0 = 01^{|p|}$
  - $s^i[0] = 0$ for all $i$

  No non-empty substring matches the empty prefix of $x$

  $s[1] == 0$ iff $p[1] == x[1]$, regardless of previous $s$
The bit $t$, where $t=0$ if $p[j]=x[i]$ and $t=1$ otherwise can be pre-calculated and stored in a bit-matrix:

$$t[h,j] = \begin{cases} 0 & \text{if } p[j] == h \\ 1 & \text{if } p[j] != h \end{cases}$$

with rows indexed by the alphabet and columns indexed by indices in $p$. 
Bit-matrix $t$ for $p=bbba$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t[\text{'a',}]$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$t[\text{'b',}]$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t[\text{'c',}]$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 0 \]

\[ p = \text{bbba} \]

\[ s^0[0] = 0 \]

\[ p = \text{bbba} \]

\[ s^0[1] = 1 \]

\[ p = \text{bbba} \]

\[ s^0[2] = 1 \]

\[ p = \text{bbba} \]

\[ s^0[3] = 1 \]

\[ p = \text{bbba} \]

\[ s^0[4] = 1 \]

\[ s = 011111 \]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 1 \]

\[ p = \text{bbba} \]

\[ p = \text{bbba} \]

\[ p = \text{bbba} \]

\[ p = \text{bbba} \]

\[ s = \text{00111} \]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 2 \]

\[ p = \text{bbba} \]
\[ p = \text{bbba} \]
\[ p = \text{bbba} \]
\[ p = \text{bbba} \]

\[ t[b] \]

\[ s_1^2[0] = 0 \]
\[ s_1^2[1] = 0 \]
\[ s_1^2[2] = 0 \]
\[ s_1^2[3] = 1 \]
\[ s_1^2[4] = 1 \]

\[ s_1 = 00011 \]
Example

i = 3

\(x = \text{bbbacbbbababacabbbba}\)

\(p = \text{bbba}\)
\(p = \text{bbba}\)
\(p = \text{bbba}\)
\(p = \text{bbba}\)

\(s^3[0] = 0\)
\(s^3[1] = 0\)
\(s^3[2] = 0\)
\(s^3[3] = 0\)
\(s^3[4] = 1\)

\(t[b]\)

\(s^2[0] = 0\)
\(s^2[1] = 0\)
\(s^2[2] = 0\)
\(s^2[3] = 1\)
\(s^2[4] = 1\)

\(s = 00001\)
Example

\[ i = 4 \]

\[ x = bbbacbbbbababacabbbba \]

\[ p = bbbba \]
\[ p = bbbba \]
\[ p = bbbba \]
\[ p = bbbba \]
\[ s = 01110 \]
Example

\[ x = \text{bbbacbbbababacabbbba} \]

\[ i = 5 \]

\[ s_5[0] = 0 \quad s_5[1] = 1 \quad s_5[2] = 1 \quad s_5[3] = 1 \quad s_5[4] = 1 \]

\[ s_4[0] = 0 \quad s_4[1] = 1 \quad s_4[2] = 1 \quad s_4[3] = 1 \quad s_4[4] = 0 \]

\[ s = 011111 \]
Example

\[ i = 6 \]

\[ x = bbbacbbababacabbbba \]

\[ s_6[0] == 0 \]
\[ s_6[1] == 0 \]
\[ s_6[2] == 1 \]
\[ s_6[3] == 1 \]
\[ s_6[4] == 1 \]
\[ s_5[0] == 0 \]
\[ s_5[1] == 1 \]
\[ s_5[2] == 1 \]
\[ s_5[3] == 1 \]
\[ s_5[4] == 1 \]

\[ s = 00111 \]
The SHIFT–and–OR Algorithm

Preprocessing:
for c in α and j=1..|p|:
   t[c,j] = 1
for j=1..|p|:
   t[p[j],j] = 0

Main:
s = 01|p|
for i=1..|x|:
   s = (s >> 1) | t[x[i]]
   if s[|p|]==0: report i−|p|+1 as match
Time usage

- Preprocessing takes time $O(|\alpha||p|)$
- Main search takes time $O(|x||p|)$
Bit-operations

- If the word size is $w$ we can usually do bit-operations on $w$ bits in constant time
  - shift $w$ bits in time $O(1)$
  - OR $w$ bits in time $O(1)$

- Manipulating larger bit-vectors can be broken down into $w$ sized chunks
  - initialize $|p|$ long bit-vector to all 1s in time $O(|p|/w)$
  - shift $|p|$ long bit-vector in time $O(|p|/w)$
  - OR $|p|$ long bit-vector in time $O(|p|/w)$
Time usage (redux)

- Preprocessing takes time $O(|\alpha| |p|/w + |p|)$
- Main search takes time $O(|x||p|/w)$
- For small $|p|$ and $|\alpha|$ this approaches a search-time $O(|x|)$ with very little overhead
- For large $|p|$ and $|\alpha|$ the approach is not advisable