

Dömölki-(Baeza-Yates)-Gonnet

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

x=*bbba**cbba**babaca**bbba*



The diagram shows three red arrows pointing down to the first three characters of the string 'bbba'. Above the first arrow is the number '1', above the second is '6', and above the third is '17'. This indicates the bit positions of the first, sixth, and seventeenth bits of the string.

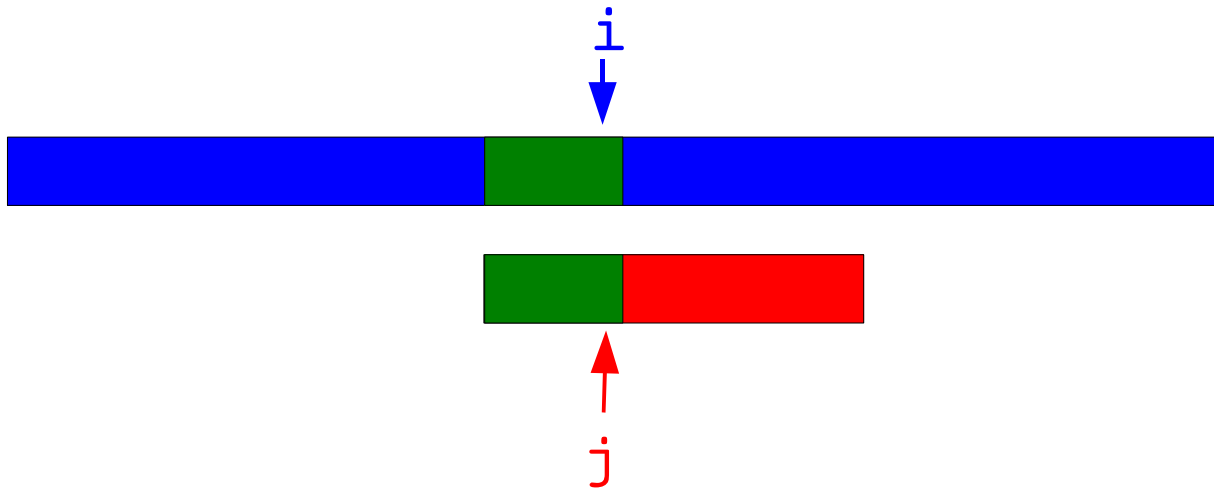
“Sprint rather than plod!”

- 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” \mathbf{s}

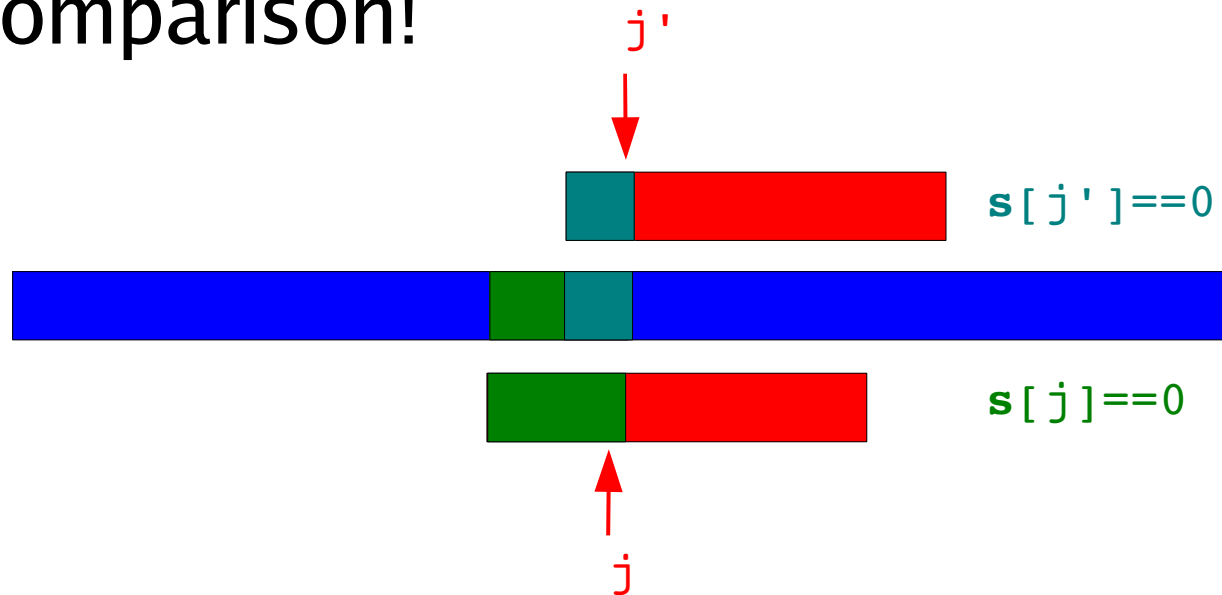
We define a vector \mathbf{s} – the state of matching so far – by:

$$\mathbf{s}[j] = 0 \text{ iff } \mathbf{x}[i-j+1 .. i] = \mathbf{p}[1..j]$$



Using a “state vector” s

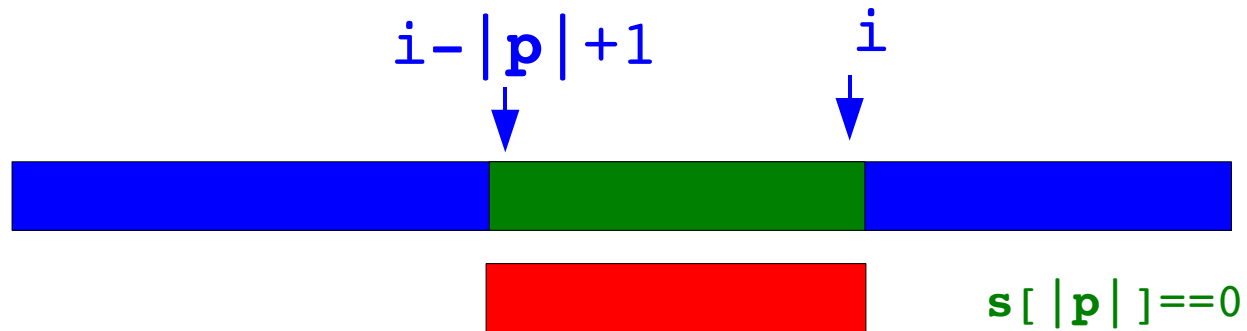
Notice that s holds information about more than one comparison!



Conceptually, p is positioned $|p|$ places along x :
 s tries to match p at positions $i-|p|+1 .. i$

Using a “state vector” s

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



Example

$i=0$



x = *bbbacbbbababacabbbba*

p = *bbba* $s[1] == 1$

p = *bbba* $s[2] == 1$

p = *bbba* $s[3] == 1$

p = *bbba* $s[4] == 1$

s = 1111

Example

i=1



x=*bbbacbbbababacabbbba*

p=*bbba* *s[1]==0*

p=*bbba* *s[2]==1*

p=*bbba* *s[3]==1*

p=*bbba* *s[4]==1*

s=0111

Example

i=2



x=*bbbacbbbababacabbbba*

p=*bbba* **s**[1]==0

p=*bbba* **s**[2]==0

p=*bbba* **s**[3]==1

p=*bbba* **s**[4]==1

s=0011

Example

$i=3$



$x = bbbacbbbababacabbbba$

$p = b$ *bbba* $s[1] == 0$

$p = bb$ *ba* $s[2] == 0$

$p = bbb$ *a* $s[3] == 0$

$p = bbba$ $s[4] == 1$

$s = 0001$

Example

$i=4$



$x = bbbacbbbababacabbba$

$p = b$ *bbba* $s[1] == 1$

$p = b$ *b* *ba* $s[2] == 1$

$p = b$ *bb* *a* $s[3] == 1$

$p = b$ *bbba* $s[4] == 0$

$s = 1110$

Match at $i-4+1=1$

Example

i=5



x=*bbbacbbbababacabbba*

p=*b*bbba **s**[1]==1

p=*bb*ba **s**[2]==1

p=*bbb*a **s**[3]==1

p=*bbba* **s**[4]==1

s=1111

Example

i=6



x=*bbbacbbbababacabbba*

p=*bbba* **s**[1]==0

p=*bbba* **s**[2]==1

p=*bbba* **s**[3]==1

p=*bbba* **s**[4]==1

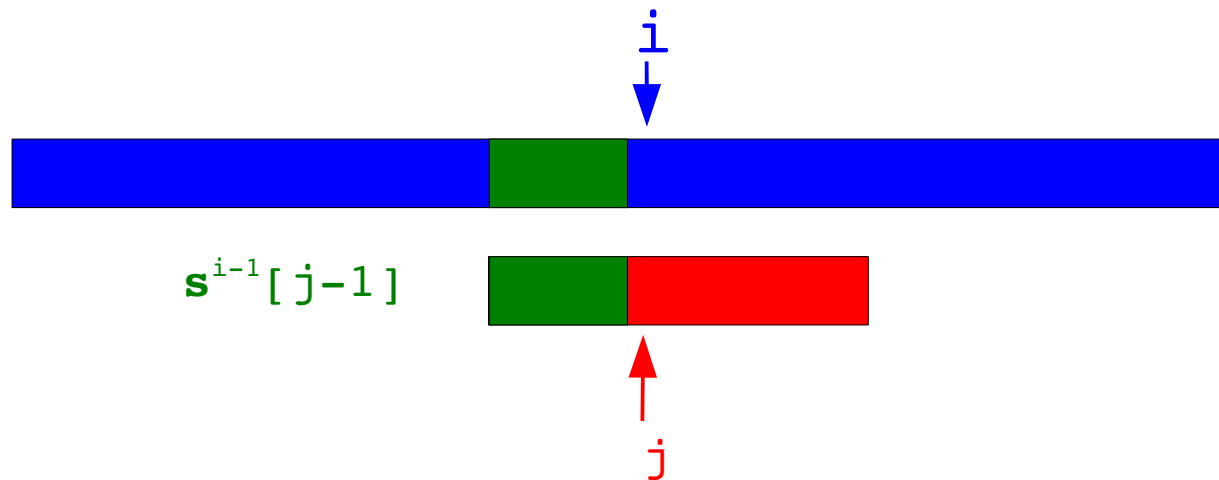
s=0111

Example

$i=7$
↓
 $x = bbbacbbbababacabbba$
 $p = bbba \quad s[1] == 0$
 $p = bbba \quad s[2] == 0$
 $p = bbba \quad s[3] == 1$
 $p = bbba \quad s[4] == 1$
 $s = 0011$

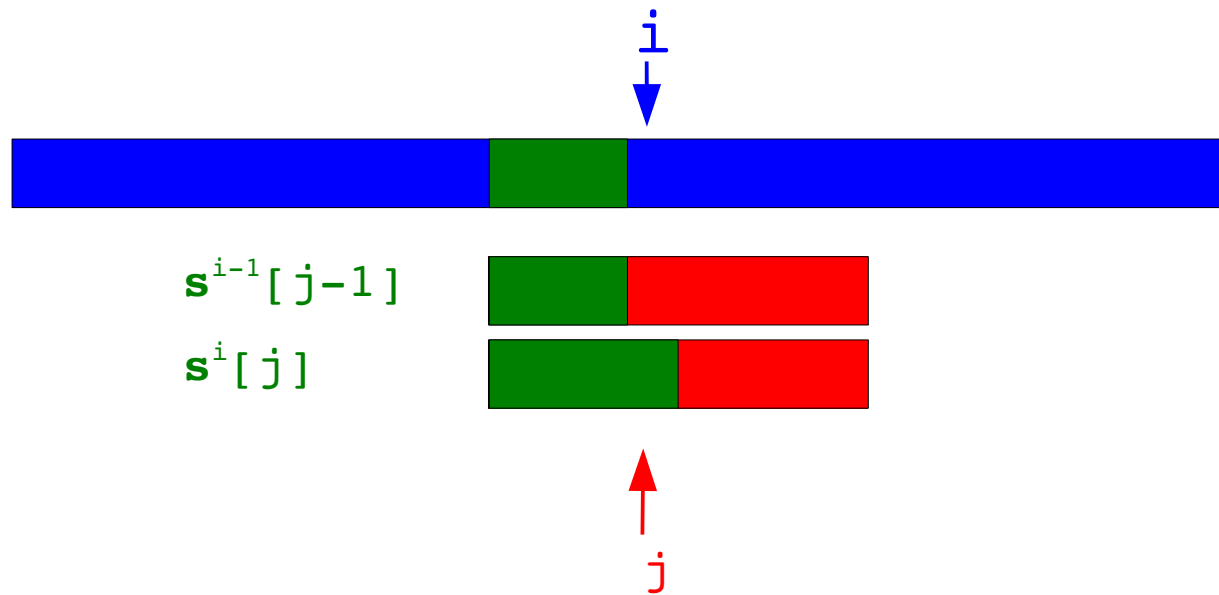
Constructing s

Let s^i be the state vector in iteration i .
Then $s^i[j] = s^{i-1}[j-1]$ OR t
where $t=0$ if $p[j] = x[i]$ and $t=1$ otherwise



Constructing s

Let s^i be the state vector in iteration i .
Then $s^i[j] = s^{i-1}[j-1]$ OR t
where $t=0$ if $p[j] = x[i]$ and $t=1$ otherwise



Special cases...

- For this to work:

- $\mathbf{s}^0 = 0\mathbf{1}^{|\mathbf{p}|}$

No non-empty substring matches the empty prefix of \mathbf{x}

- $\mathbf{s}^i[0] = 0$ for all i

$\mathbf{s}[1] == 0$ iff $\mathbf{p}[1] == \mathbf{x}[1]$, regardless of previous \mathbf{s}

Bit-matrix for **t**

The bit t , where $t=0$ if $\mathbf{p}[j]=\mathbf{x}[i]$ and $t=1$ otherwise can be pre-calculated and stored in a bit-matrix:

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

with rows indexed by the alphabet and columns indexed by indices in \mathbf{p}

Bit-matrix **t** for **p=bbba**

	1	2	3	4
t ['a' ,]:	1	1	1	0
t ['b' ,]:	0	0	0	1
t ['c' ,]:	1	1	1	1

Example

$i=0$
↓
 $x = bbbacbbbababacabbbba$

$p = bbbba$	$s^0[0] == 0$
$p = bbba$	$s^0[1] == 1$
$p = bbba$	$s^0[2] == 1$
$p = bbba$	$s^0[3] == 1$
$p = bbba$	$s^0[4] == 1$

$s = 01111$

Example

$i=1$
↓
 $x = bbbacbbbababacabbbba$

		$t[b]$		
$p = bbbba$	$s^1[0] == 0$			$s^0[0] == 0$
$p = bbba$	$s^1[1] == 0$	0 OR	↙	$s^0[1] == 1$
$p = bbba$	$s^1[2] == 1$	0 OR	↙	$s^0[2] == 1$
$p = bbba$	$s^1[3] == 1$	0 OR	↙	$s^0[3] == 1$
$p = bbba$	$s^1[4] == 1$	1 OR	↙	$s^0[4] == 1$

$s = 00111$

Example

$i=2$
↓
 $x = bbbacbbbababacabbba$

		$t[b]$		
$p = bbba$	$s^2[0] == 0$			$s^1[0] == 0$
$p = bbba$	$s^2[1] == 0$	0 OR	↙	$s^1[1] == 0$
$p = bbba$	$s^2[2] == 0$	0 OR	↙	$s^1[2] == 1$
$p = bbba$	$s^2[3] == 1$	0 OR	↙	$s^1[3] == 1$
$p = bbba$	$s^2[4] == 1$	1 OR	↙	$s^1[4] == 1$

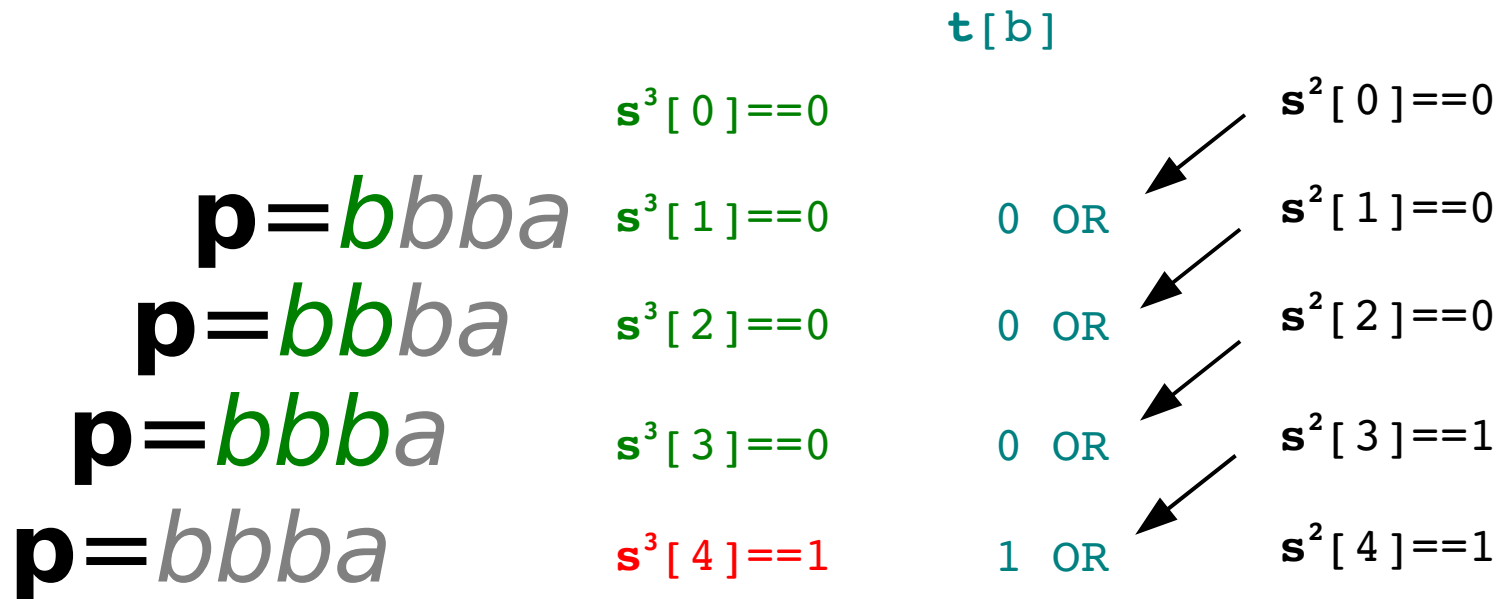
$s = 00011$

Example

i=3



x=*bbbacbbbabacabbba*



s=00001

Example

$i=4$



$x = bbbacbbbababacabbbba$

		$t[a]$	
$p = bbbba$	$s^4[0] == 0$		$s^3[0] == 0$
$p = bbba$	$s^4[1] == 1$	1 OR	$s^3[1] == 0$
$p = bba$	$s^4[2] == 1$	1 OR	$s^3[2] == 0$
$p = ba$	$s^4[3] == 1$	1 OR	$s^3[3] == 0$
$p = a$	$s^4[4] == 0$	0 OR	$s^3[4] == 1$

$s = 01110$

Example

$i=5$
↓
 $x = bbbacbbbababacabbbba$

		$t[c]$		
$p = bbbba$	$s^5[0] == 0$			$s^4[0] == 0$
$p = bbba$	$s^5[1] == 1$	1 OR	↙	$s^4[1] == 1$
$p = bbb a$	$s^5[2] == 1$	1 OR	↙	$s^4[2] == 1$
$p = bbba$	$s^5[3] == 1$	1 OR	↙	$s^4[3] == 1$
	$s^5[4] == 1$	1 OR	↙	$s^4[4] == 0$

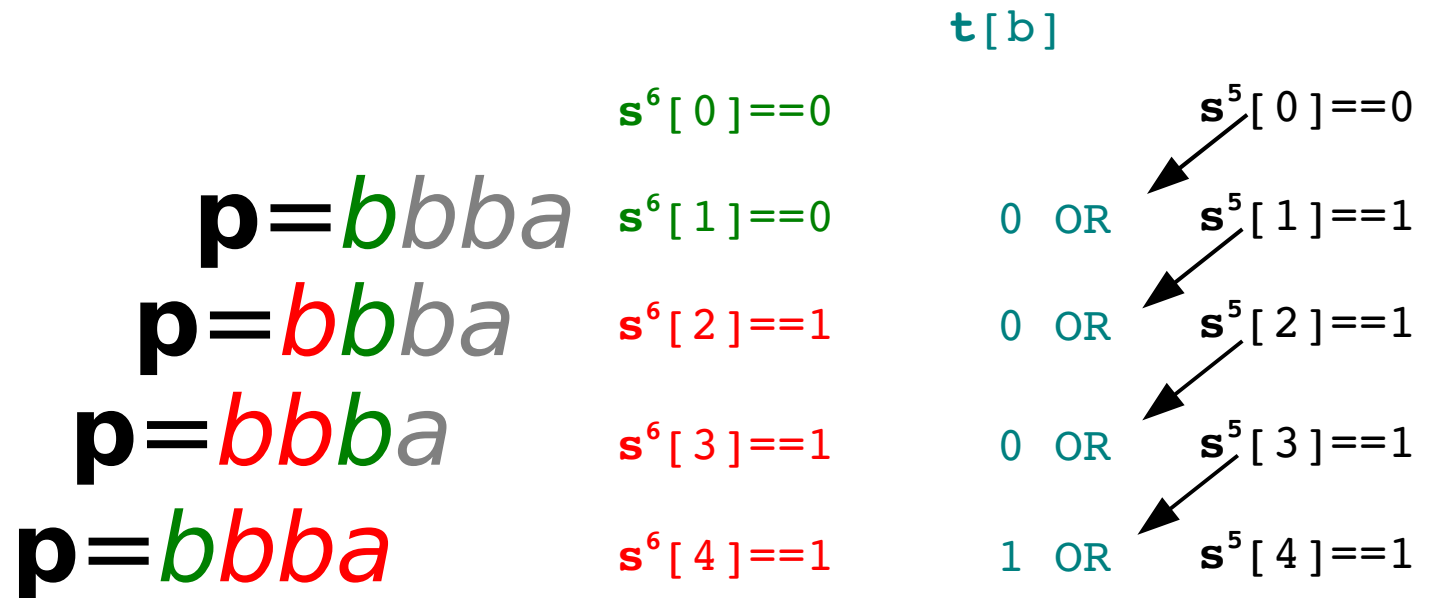
$s = 01111$

Example

$i=6$



$x = bbbacbbbababacabbbba$



$s = 00111$

The SHIFT-and-OR Algorithm

Preprocessing:

```
for c in  $\alpha$  and  $j=1..|p|$ :  
     $t[c, j] = 1$   
for  $j=1..|p|$ :  
     $t[p[j], j] = 0$ 
```

Main:

```
 $s = 01^{|p|}$   
for  $i=1..|x|$ :  
     $s = (s \gg 1) | t[x[i]]$   
    if  $s[|p|] == 0$ : report  $i-|p|+1$  as match
```

Time usage

- Preprocessing takes time $O(|\alpha||\mathbf{p}|)$
- Main search takes time $O(|\mathbf{x}||\mathbf{p}|)$

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 $|\mathbf{p}|$ long bit-vector to all 1s in time $O(|\mathbf{p}|/w)$
 - shift $|\mathbf{p}|$ long bit-vector in time $O(|\mathbf{p}|/w)$
 - OR $|\mathbf{p}|$ long bit-vector in time $O(|\mathbf{p}|/w)$

Time usage (redux)

- Preprocessing takes time $O(|\alpha||\mathbf{p}|/w + |\mathbf{p}|)$
- Main search takes time $O(|\mathbf{x}||\mathbf{p}|/w)$
- For small $|\mathbf{p}|$ and $|\alpha|$ this approaches a search-time $O(|\mathbf{x}|)$ with very little overhead
- For large $|\mathbf{p}|$ and $|\alpha|$ the approach is not advisable
- In the final project we will, in a bioinformatics setting, search for small “anchor”-strings over a small (DNA) alphabet