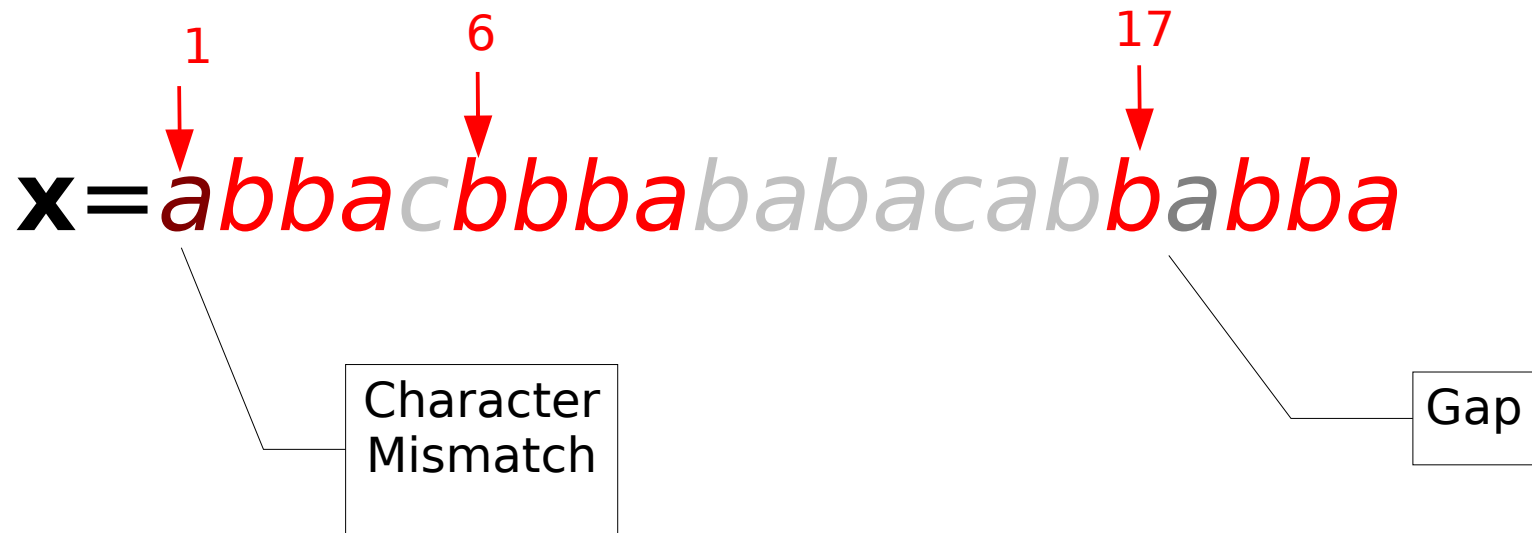


Approximate pattern matching

Given string $\mathbf{x} = \text{abbacbbbabacabbba}$ and pattern $\mathbf{p} = \text{bbba}$ find all “almost”-occurrences of \mathbf{p} in \mathbf{x}



String distance

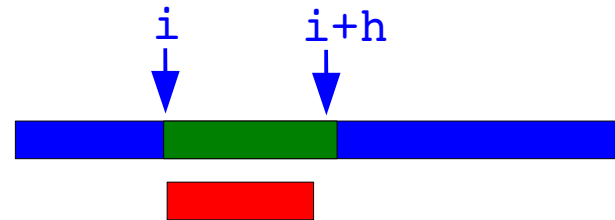
- A number of string-distances have been suggested, e.g.:
 - *Hamming* distance: $d(\mathbf{x}, \mathbf{y})$ = number of characters that differs between \mathbf{x} and \mathbf{y}
 - $d(abca, abaa) = 1$, $d(abca, abab) = 2$
 - *Levenshtein* distance: $d(\mathbf{x}, \mathbf{y})$ = number of deletions and insertions needed to transform \mathbf{x} into \mathbf{y} :
 - $d(abca, abaa) = 2$, $d(abca, aba) = 1$
 - *Edit* distance: $d(\mathbf{x}, \mathbf{y})$ = number of insertions, deletions, or substitutions needed to transform \mathbf{x} into \mathbf{y}
 - $d(abca, abaa) = 1$, $d(abca, cca) = 2$

k-Approximate matching

Given string \mathbf{x} and pattern \mathbf{p} find all indices in \mathbf{x} where:

$$d(\mathbf{x}[i..i+h], \mathbf{p}) \leq k$$

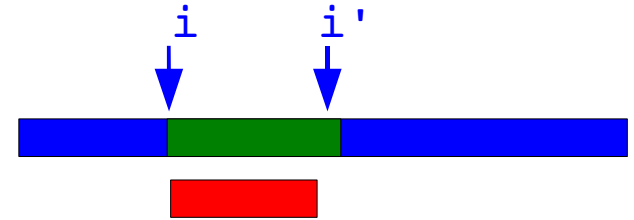
$$d(\text{■}, \text{■}) \leq k$$



Generic problem for the various distance functions d

Generic Algorithm

```
for i=1..|x|:  
  if d(x[i..i'],p) <= k for some i'>i:  
    report match at i
```



Time usage: $O(n^2 \cdot \text{"time to calculate distance"})$

(But see Sect. 10.1 for a $O(nm)$ dynamic programming algorithm that works for most distance functions)

The k -mismatch problem

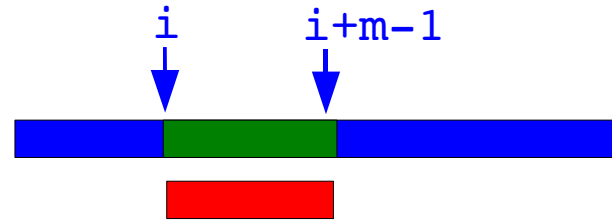
Let, for strings \mathbf{x} and \mathbf{y} , $|\mathbf{x}|=|\mathbf{y}|=n$, $d(\mathbf{x},\mathbf{y}) = |\{i \mid i=1..n, \mathbf{x}[i] \neq \mathbf{y}[i]\}|$
(the *Hamming* distance)

The k -mismatch problem:

Given string \mathbf{x} and pattern \mathbf{p} find all indices in \mathbf{x} where:

$$d(\mathbf{x}[i..i+m-1], \mathbf{p}) \leq k$$

$$d(\text{ [green bar] }, \text{ [red bar] }) \leq k$$



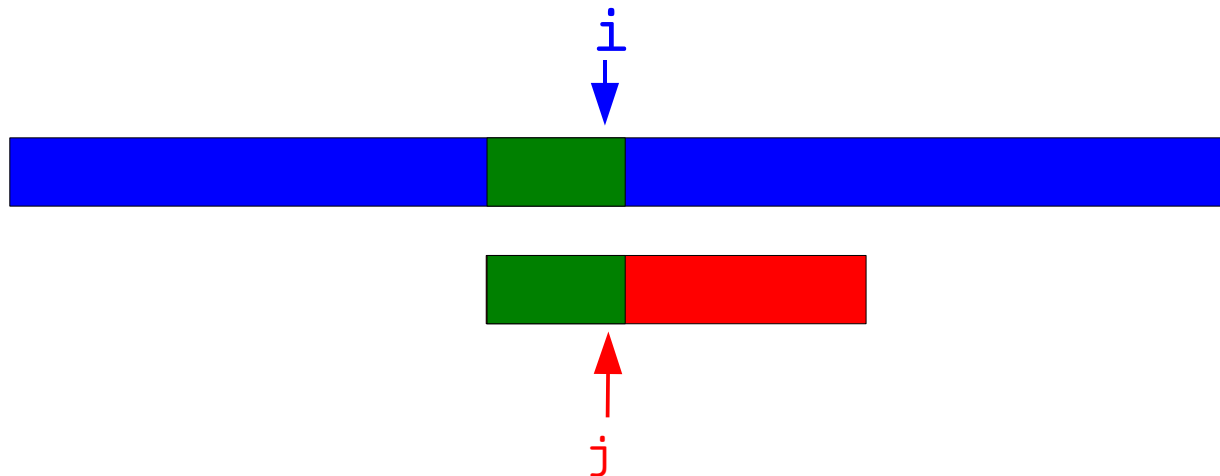
Simple k-mismatch algorithm

```
for i=1..|x|:  
  count = 0  
  for j=1..|p|:  
    if x[i]!=p[j]: count = count + 1  
  if count <= k: report match at i
```

Time usage: $O(|\mathbf{x}||\mathbf{p}|)$

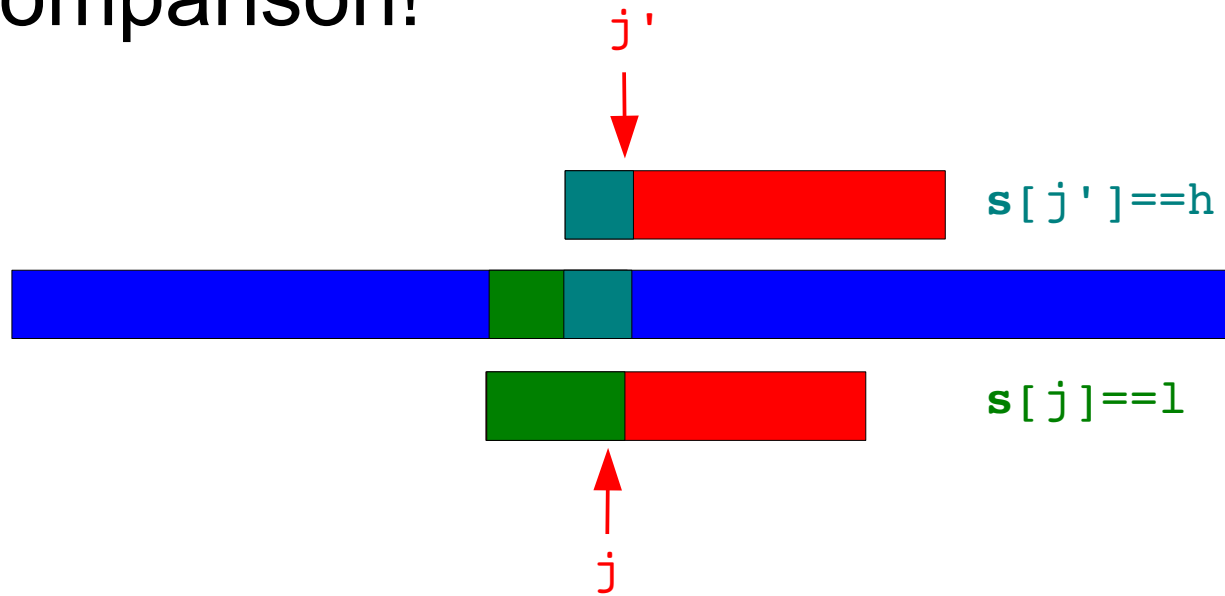
Bit-vector approach

- Inspired by SHIFT-and-OR
- Use a state bit-vector \mathbf{s} to speed up the simple algorithm
- For each index j in \mathbf{p} , \mathbf{s} uses $\log(|\mathbf{p}|+1)$ bits
- $\mathbf{s}[j] = d(\mathbf{x}[i-j+1..i], \mathbf{p}[1..j])$



Using state vector s

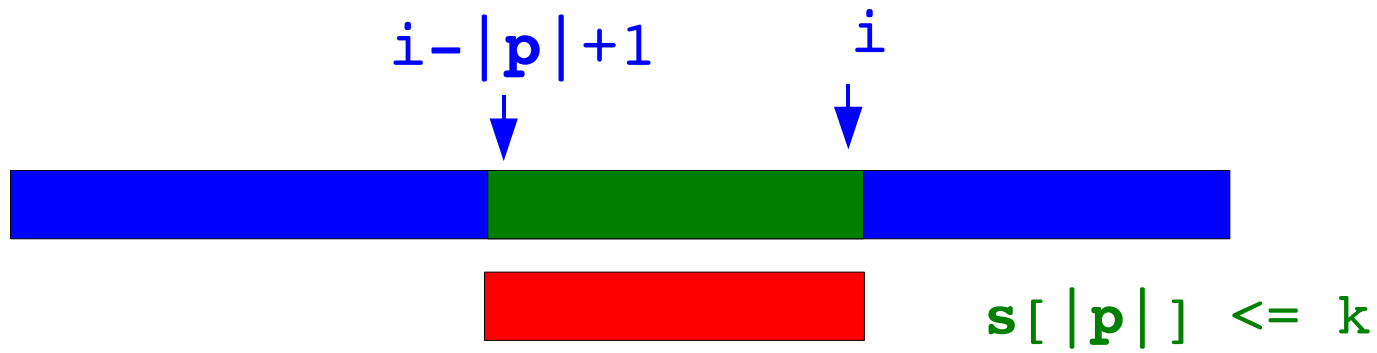
Notice that s holds information about more than one comparison!



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

Using state vector s

When $s[|p|] \leq k$ and $i \geq |p|$, we have an occurrence of p in x at $i - |p| + 1$



Example: 2-mismatch

$i=0$



x = *babacbbbababacabbbba*

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

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

p = *bbba* $s[3]==0$

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

s = 000 000 000 000

Example: 2-mismatch

$i=1$



$x = babacbbbababacabbbba$

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

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

$p =$ *bbba* $s[3] == 0$

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

$s = 000 \ 000 \ 000 \ 000$

Example: 2-mismatch

$i=2$



$x = babacbbbababacabbbba$

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

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

$p =$ *bbba* $s[3] == 0$

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

$s = 001\ 001\ 000\ 000$

Example: 2-mismatch

$i=3$
↓
 $x = babacbbbababacabbba$
 $p = bba$ $s[1] == 0$
 $p = bba$ $s[2] == 1$
 $p = bba$ $s[3] == 1$
 $p = bba$ $s[4] == 0$

$s = 000\ 001\ 001\ 000$

Example: 2-mismatch

$i=4$
↓
 $x = babacbbbababacabbba$
 $p = bba$ $s[1] == 1$
 $p = bba$ $s[2] == 1$
 $p = bba$ $s[3] == 2$
 $p = bba$ $s[4] == 1$

$s = 001\ 001\ 010\ 001$

Match at $i-4+1=1$

Example: 2-mismatch

$i=5$
↓
 $x = babacbbbababacabbba$
 $p = bbaa$ $s[1] == 1$
 $p = bbba$ $s[2] == 2$
 $p = bbaa$ $s[3] == 2$
 $p = bbba$ $s[4] == 3$

$s = 001\ 010\ 010\ 011$

Example: 2-mismatch

$i=6$



$x = babacbbbababacabbba$

$p = bbaa$ $s[1] == 0$

$p = bba$ $s[2] == 1$

$p = bba$ $s[3] == 2$

$p = bba$ $s[4] == 3$

$s = 000 \ 001 \ 010 \ 011$

Example: 2-mismatch

$i=7$
↓
 $x = babacbbbababacabbba$

$p = bba$ $s[1] == 0$

$p = bba$ $s[2] == 0$

$p = bba$ $s[3] == 1$

$p = bba$ $s[4] == 3$

$s = 000\ 000\ 001\ 011$

Example: 2-mismatch

$i=8$
↓
 $x = babacbbbababacabbba$
 $p = bbaa$ $s[1] == 0$
 $p = bbba$ $s[2] == 0$
 $p = bbbba$ $s[3] == 0$
 $p = bbbba$ $s[4] == 2$

$s = 000\ 000\ 001\ 010$

Match at $i-4+1=5$

Example: 2-mismatch

$i=9$
↓
 $x = babacbbbababacabbba$
 $p = bba$ $s[1] == 1$
 $p = bba$ $s[2] == 1$
 $p = bba$ $s[3] == 1$
 $p = bba$ $s[4] == 0$

$s = 001\ 001\ 001\ 000$

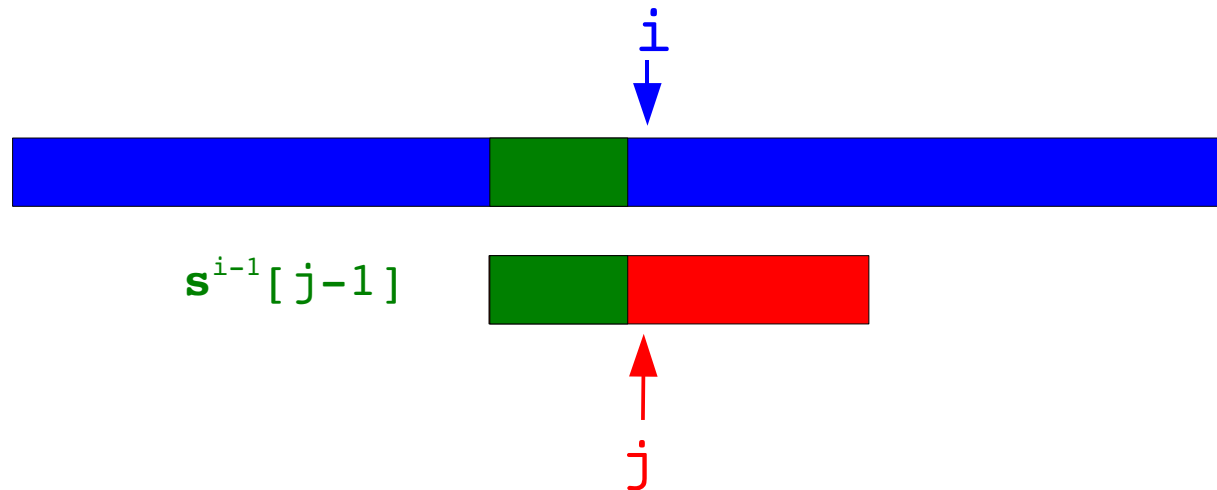
Match at $i-4+1=6$

Constructing s

Let s^i be the state vector in iteration i .

Then $s^i[j] = s^{i-1}[j-1] + t$

where $t=0$ if $p[j]=x[i]$ and $t=1$ otherwise



Special case: $s^i[0] = 0$ for all i

Table t

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

$$\mathbf{t}[h, j] = \begin{array}{ll} 0^{\log(|\mathbf{p}|+1)} & \text{if } \mathbf{p}[j] == h \\ 0^{\log(|\mathbf{p}|+1)-1} \mathbf{1} & \text{if } \mathbf{p}[j] != h \end{array}$$

with rows indexed by the alphabet and columns indexed by indices in \mathbf{p} , using $\log(|\mathbf{p}|+1)$ bits per cell.

Table t

Why use $\log(|p|+1)$ bits per cell?

To be able to add $t[h]$ and s a word at a time, silly!

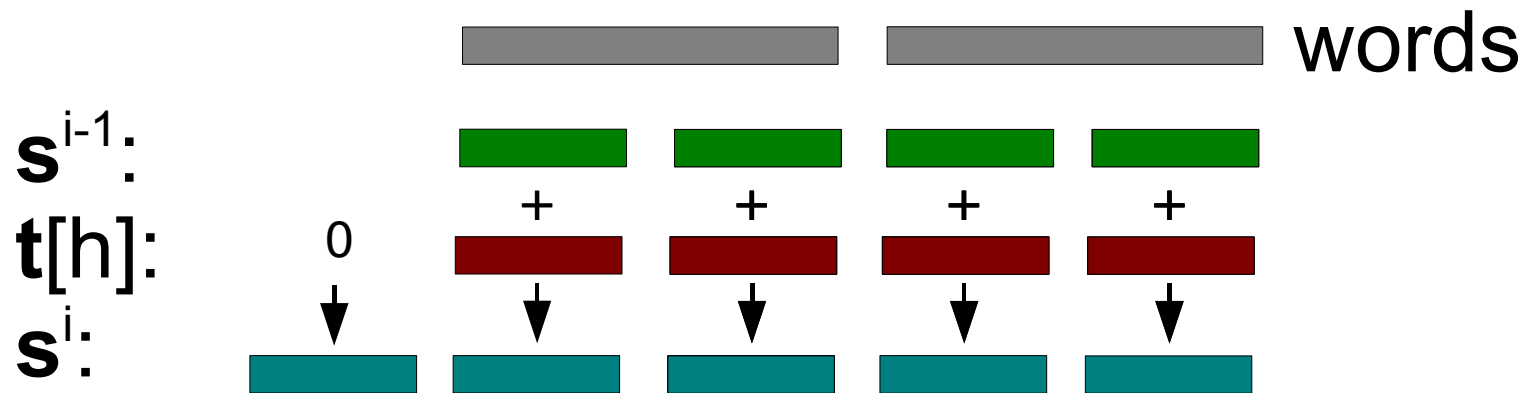


Table t for pattern p=bbba

	1	2	3	4
t ['a']:	001	001	001	000
t ['b']:	000	000	000	001
t ['c']:	001	001	001	001

Example

$i=0$



$x = babacbbbababacabbba$

$s^0[0] == 0$

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

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

p = *bbba* $s^0[3] == 0$

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

$s = 000\ 000\ 000\ 000\ 000$

Example

$i=1$



$x = babacbbbababacabbba$

$t[b]$

	$s^1[0] == 0$		$s^0[0] == 0$
$p = b$	$s^1[1] == 0$	0 +	$s^0[1] == 0$
$p = bb$	$s^1[2] == 0$	0 +	$s^0[2] == 0$
$p = bbb$	$s^1[3] == 0$	0 +	$s^0[3] == 0$
$p = bbba$	$s^1[4] == 1$	1 +	$s^0[4] == 0$

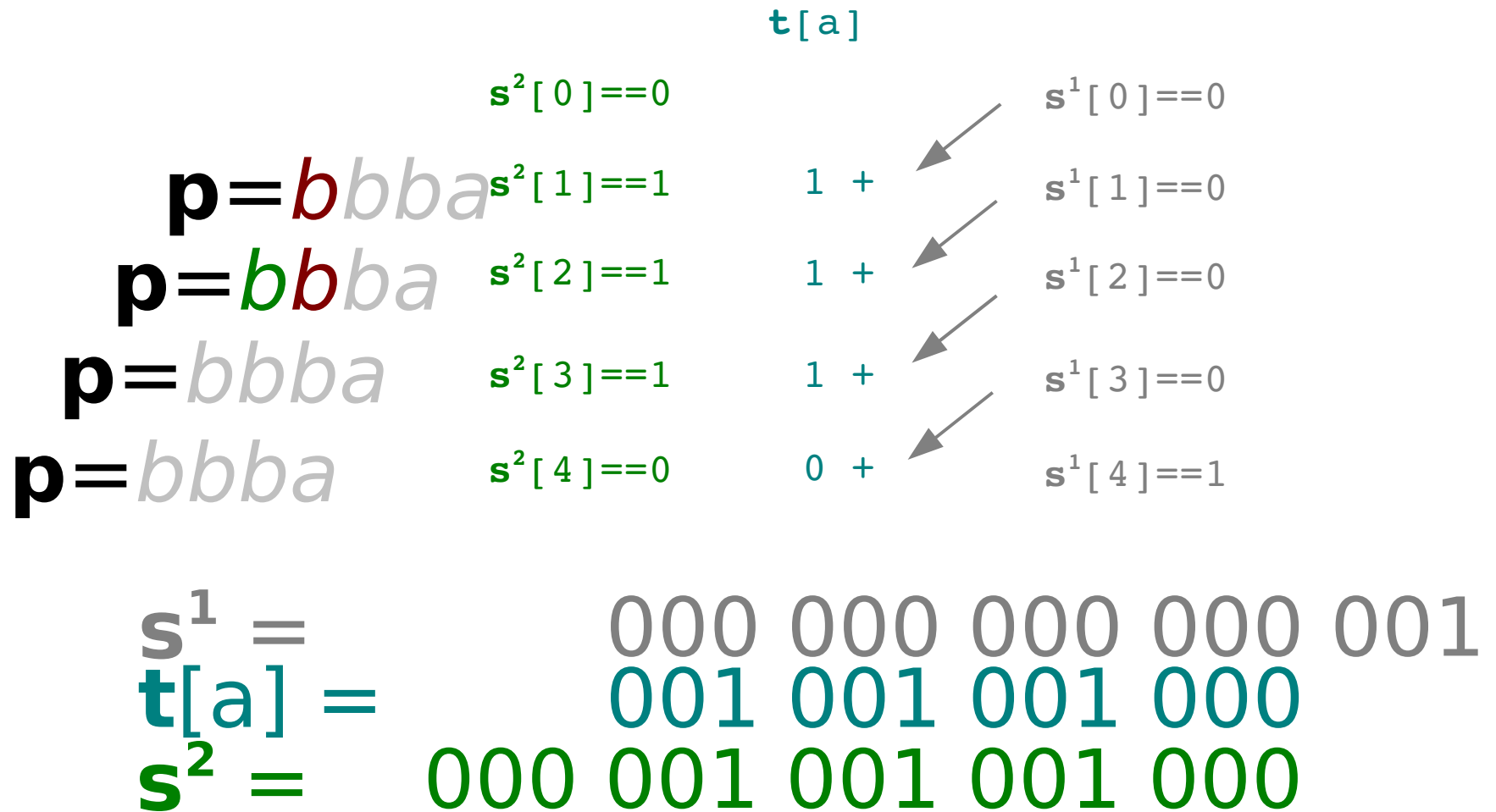
$s^0 =$	000	000	000	000	000
$t[b] =$	000	000	000	001	
$s^1 =$	000	000	000	000	001

Example

$i=2$



$x = babacbbbababacabbba$



Example

$i=3$



$x = babacbbbababacabbba$

		$t[b]$		
	$s^3[0] == 0$			$s^2[0] == 0$
$p = b$ <i>bbba</i>	$s^3[1] == 0$	0 +		$s^2[1] == 1$
$p = b$ <i>bb</i> <i>a</i>	$s^3[2] == 1$	0 +		$s^2[2] == 1$
$p = b$ <i>bb</i> <i>b</i> <i>a</i>	$s^3[3] == 1$	0 +		$s^2[3] == 1$
$p = b$ <i>bb</i> <i>b</i> <i>b</i> <i>a</i>	$s^3[4] == 2$	1 +		$s^2[4] == 0$

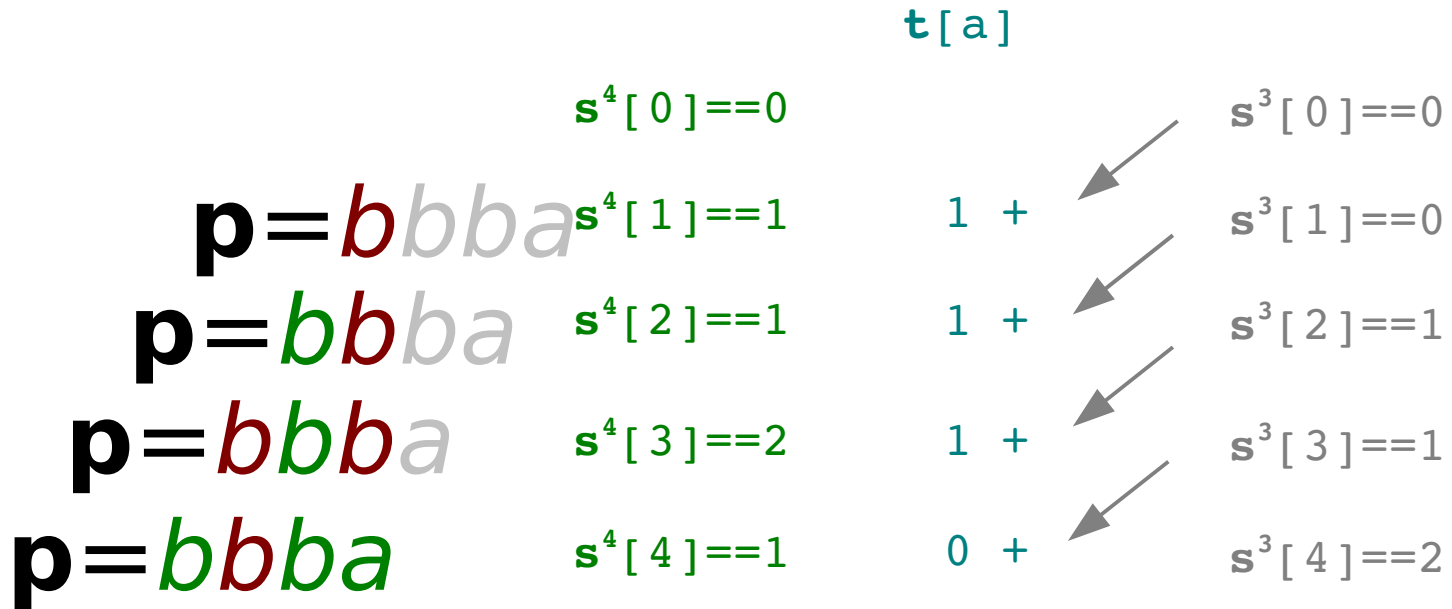
$s^2 =$	000	001	001	001	000
$t[b] =$	000	000	000	001	
$s^3 =$	000	000	001	001	010

Example

$i=4$



$x = babacbbbababacabbba$



$s^3 =$	000	000	001	001	010
$t[a] =$	001	001	001	000	
$s^4 =$	000	001	001	010	001

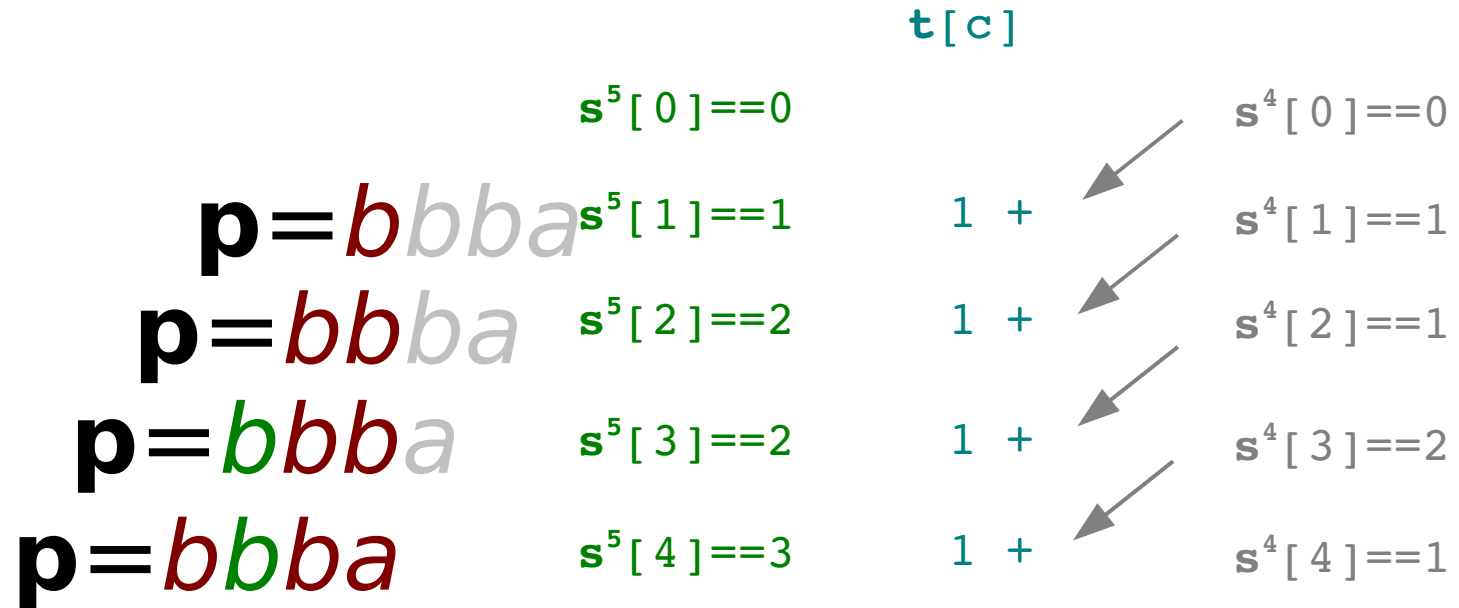
MATCH!

Example

$i=5$



$x = babacbbbababacabbba$



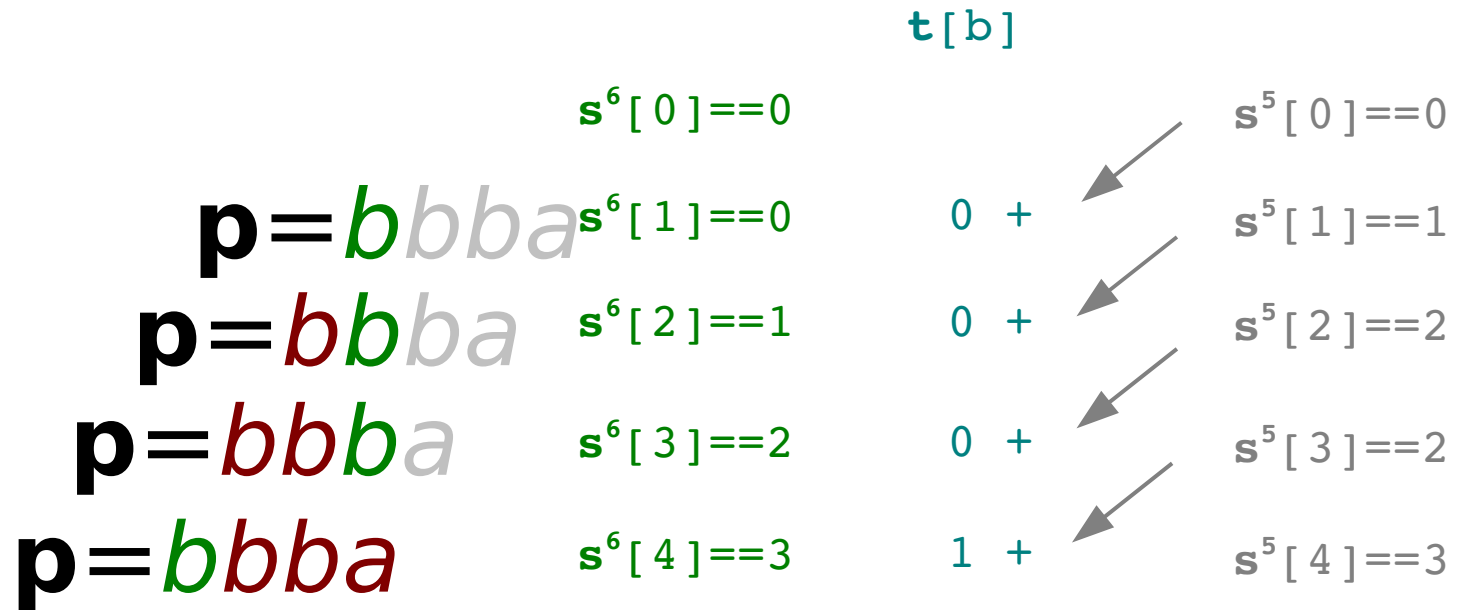
$s^4 =$	000	001	001	010	001
$t[c] =$	001	001	001	001	
$s^5 =$	000	001	010	010	011

Example

$i=6$



$x = babacbbbababacabbba$



$s^5 =$	000	001	010	010	011
$t[b] =$	000	000	000	001	
$s^6 =$	000	000	001	010	011

Algorithm

Preprocessing:

```
b = log(|p|+1)
for h in  $\alpha$  and j=1..|p|:
    t[h,j] =  $0^{b-1}1$ 
for j=1..|p|:
    t[p[j],j] =  $0^b$ 
s =  $0^{b(|p|+1)}$ 
```

Main:

```
for i=1..|x|:
    s = (s >> b) + t[x[i]]
    if s[|p|] <= k and i >= |p|:
        report i-|p|+1 as match
```

Runtime complexity

- Preprocessing takes time $O(|\alpha||\mathbf{p}|\log(|\mathbf{p}|+1)/w+|\mathbf{p}|)$
- Main search takes time $O(|\mathbf{x}||\mathbf{p}|\log(|\mathbf{p}|+1)/w)$

Improvements

When $k \ll |\mathbf{p}|$, we are wasting bits (and time)!

We count all mismatches, but we only need to know if there are more than k or not.

We can replace the $\log(|\mathbf{p}|+1)$ bit cells in \mathbf{s} with $\log(k+1)+1$ (for numbers $0..k$ and an overflow bit) and an additional vector, \mathbf{o} , remembering overflows

Updated operation

We update the operation

$$\mathbf{s} = (\mathbf{s} \gg b) + \mathbf{t}[\mathbf{x}[i]]$$

where $b = \log(|p|+1)$, to be

$$\mathbf{s} = (\mathbf{s} \gg b) + \mathbf{t}[\mathbf{x}[i]]$$

$$\mathbf{o} = (\mathbf{o} \gg b) \mid (\mathbf{s} \& \text{OF_MASK})$$

$$\mathbf{s} = \mathbf{s} \& \text{NEG_OF_MASK}$$

where $b = \log(k+1)+1$ and

$$\text{OF_MASK} = (10^{b-1})^{|p|}$$

$$\text{NEG_OF_MASK} = \sim\text{OF_MASK} = (01^b)^{|p|}$$

Algorithm

Preprocessing:

$b = \log(k+1)+1$

$\text{OF_MASK} = (10^{b-1})^{|\mathbf{p}|}$; $\text{NEG_OF_MASK} = \sim\text{OF_MASK}$

for h **in** α **and** $j=1..|\mathbf{p}|$: $\mathbf{t}[h,j] = 0^{b-1}1$

for $j=1..|\mathbf{p}|$: $\mathbf{t}[\mathbf{p}[j],j] = 0^b$

$\mathbf{s} = 0^{b(|\mathbf{p}|+1)}$; $\mathbf{o} = 0^{b(|\mathbf{p}|+1)}$

Main:

for $i=1..|\mathbf{x}|$:

$\mathbf{s} = (\mathbf{s} \gg b) + \mathbf{t}[\mathbf{x}[i]]$

$\mathbf{o} = (\mathbf{o} \gg b) | (\mathbf{s} \& \text{OF_MASK})$

$\mathbf{s} = \mathbf{s} \& \text{NEG_OF_MASK}$

if $\mathbf{s}[|\mathbf{p}|] + \mathbf{o}[|\mathbf{p}|] \leq k$ **and** $i \geq |\mathbf{p}|$:

report $i-|\mathbf{p}|+1$ as match

Example

$i=0$



x = *babacacbababacabbbba*

p = *bbba*

p = *bbba*

p = *bbba*

p = *bbba*

s⁰ == 000 000 000 000 000

o⁰ == 000 000 000 000 000

Example

$i=1$



$\mathbf{x} = \text{babacacbababacabbbba}$

$\mathbf{p} = \text{bbba}$

$\mathbf{p} = \text{bbba}$

$\mathbf{p} = \text{bbba}$

$\mathbf{p} = \text{bbba}$

$(\mathbf{s}^0 \gg \mathbf{b}) \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 000$

$\mathbf{t}[\text{'b'}] \quad == \quad \quad \quad 000 \quad 000 \quad 000 \quad 001$

$\mathbf{s}^{1'} \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 001$

$(\mathbf{o}^0 \gg \mathbf{b}) \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 000$

$(\mathbf{s}^{1'} \ \& \ \text{OF_MASK}) \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 000$

$\mathbf{o}^1 \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 000$

$\mathbf{s}^1 \quad == \quad 000 \quad 000 \quad 000 \quad 000 \quad 001$

Example

i=2



x=babacacbababacabbbba

p=*b*bbba

p=*bb*ba

p=bbba

p=bbba

(s¹ >> b) == 000 000 000 000 000

t['a'] == 001 001 001 000

s²' == 000 001 001 001 000

(o¹ >> b) == 000 000 000 000 000

(s²' & OF_MASK) == 000 000 000 000 000

o² == 000 000 000 000 000

s² == 000 001 001 001 000

Example

i=3



x=babacacbababacabbbba

p=*b*bbba

p=*b**b*bbba

p=*b**b**b*ba

p=bbba

(s² >> b) == 000 000 001 001 001

t['b'] == 000 000 000 000 001

s³ == 000 000 001 001 010

(o² >> b) == 000 000 000 000 000

(s³ & OF_MASK) == 000 000 000 000 000

o³ == 000 000 000 000 000

s³ == 000 000 001 001 010

Example

i=4



x=babacacbababacabbbba

p=*b*bbba

p=*bb*ba

p=*bbb*a

p=*bbba*

(s³ >> b) == 000 000 000 001 001

t['a'] == 001 001 001 000

s⁴ == 000 001 001 010 001

(o³ >> b) == 000 000 000 000 000

(s⁴ & OF_MASK) == 000 000 000 000 000

o⁴ == 000 000 000 000 000

s⁴ == 000 001 001 010 001

MATCH!

Example

i=5



x=babacacbababacabbba

p=*b*bba

p=*bb*ba

p=*bbb*a

p=*bbba*

(s ⁴ >> b)	==	000	000	001	001	010
t['c']	==		001	001	001	001
s ⁵	==	000	001	010	010	011
(o ⁴ >> b)	==	000	000	000	000	000
(s ⁵ & OF_MASK)	==	000	000	000	000	000
o ⁵	==	000	000	000	000	000
s ⁵	==	000	001	010	010	011

Example

i=6



x=babacacbababacabbba

p=bbba

p=bbba

p=bbba

p=bbba

(s⁵ >> b) == 000 000 001 010 010

t['a'] == 001 001 001 000

s⁶ == 000 001 010 011 010

(o⁵ >> b) == 000 000 000 000 000

(s⁶ & OF_MASK) == 000 000 000 000 000

o⁶ == 000 000 000 000 000

s⁶ == 000 001 010 011 010

MATCH!

Example

i=7



x=babacacbababacabbbba

p=bbba

p=bbba

p=bbba

p=bbba

(s ⁶ >> b)	==	000	000	001	010	011
t['c']	==		001	001	001	001
s ⁷	==	000	001	010	011	100
(o ⁶ >> b)	==	000	000	000	000	000
(s ⁷ & OF_MASK)	==	000	000	000	000	100
o ⁷	==	000	000	000	000	100
s ⁷	==	000	001	010	011	000

Example

i=8



x = *babacacbababacabbbba*

p = *bbba*

p = *bbba*

p = *bbba*

p = *bbba*

<code>(s⁷ >> b)</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>001</code>	<code>010</code>	<code>011</code>
<code>t['b']</code>	<code>==</code>		<code>000</code>	<code>000</code>	<code>000</code>	<code>001</code>
<code>s⁸</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>001</code>	<code>010</code>	<code>100</code>
<code>(o⁷ >> b)</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>000</code>
<code>(s⁸ & OF_MASK)</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>100</code>
<code>o⁸</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>000</code>	<code>100</code>
<code>s⁸</code>	<code>==</code>	<code>000</code>	<code>000</code>	<code>001</code>	<code>010</code>	<code>000</code>

Time complexity

- Preprocessing takes time $O(|\alpha||\mathbf{p}|\log(k)/w+|\mathbf{p}|)$
- Main search takes time $O(|\mathbf{x}||\mathbf{p}|\log(k)/w)$