Shift – And method

2015. 04. 27

Yoonsung Joh

Contents

- **String matching methods based on bit operations or on arithmetic. (Previous methods were based on character comparison)**

1. *Shift-And* method

2. *agrep*: The *Shift-And* method with errors

3. Using Fast Fourier Transform for match-counts

1. The Shift-And method

Shift-And method

- Bit oriented method that solves the **exact matching problem.**

- Find all matched positions of $P[1..i]$ in T.

- When $i=m$, that is the answer of the exact matching problem.

1. The Shift-And method

Definition

M

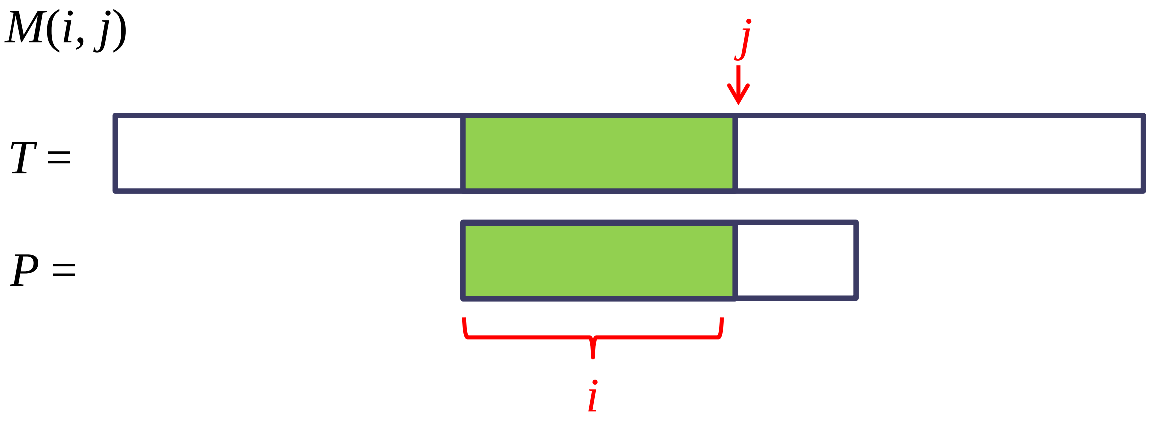
Let M be an m by n+1 binary value array, with $1 \leq i \leq m$ and $1 \leq j \leq n$. Entry $M(i, j)$ is 1 if and only if the first i characters of P exactly match the i characters of T ending at character j. Otherwise the entry is zero.

1. The Shift-And method

Definition

M

Let M be an m by n+1 binary value array, with $1 \leq i \leq m$ and $1 \leq j \leq n$. Entry $M(i, j)$ is 1 if and only if the first i characters of P exactly match the i characters of T ending at character j. Otherwise the entry is zero.



1. The Shift-And method

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a										
a										
c										

1. The Shift-And method

ex) for i=1

find the position of “a” in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a										
a										
c										

1. The Shift-And method

ex) for i=1

find the position of "a" in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	1		1	1	1		1	1		
a										
c										

1. The Shift-And method

ex) for i=1

find the position of "a" in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a										
c										

1. The Shift-And method

ex) for i=2

find the position of "aa" in the Text

P \ T	a	b	**a**	**a**	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a				**1**						
c										

1. The Shift-And method

ex) for i=2

find the position of “aa” in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a				1	1					
c										

1. The Shift-And method

ex) for i=2

find the position of "aa" in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a				1	1			1		
c										

1. The Shift-And method

ex) for i=2

find the position of "aa" in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	**1**	**1**	0	0	**1**	0	0
c										

1. The Shift-And method

ex) for i=3

find the position of “aac” in the Text

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c						1				

1. The Shift-And method

ex) for i=3

find the position of "aac" in the Text

P \ T	a	b	a	a	a	c	**a**	**a**	**c**	b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c						1			**1**	

1. The Shift-And method

ex)

for i=3

find the position of "aac" in the Text

P \ *T*	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c	0	0	0	0	0	**1**	0	0	**1**	0

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

1. The Shift-And method

	1	2	3	4	5	6	7	8	9	10
P \ ***T***	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a		**?**			**?**	**?**	**?**		**?**	**?**
c										

candidates for *i*=2

1. The Shift-And method

P \ *T*	1	2	3	4	5	6	7	8	9	10
	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**								
c										

a ≠ b

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**	0	**1**						
c										

a = a

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**	0	**1**	**1**					
c										

a = a

1. The Shift-And method

	1	2	3	4	5	6	7	8	9	10
P \ ***T***	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**	0	**1**	**1**	**0**				
c										

a ≠ c

1. The Shift-And method

P \ T	a (1)	b (2)	a (3)	a (4)	a (5)	c (6)	a (7)	a (8)	c (9)	b (10)
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**	0	**1**	**1**	**0**	0	**1**		
c										

a = a

1. The Shift-And method

T	1	2	3	4	5	6	7	8	9	10
	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a	0	**0**	0	**1**	**1**	**0**	0	**1**	**0**	0
c										

a ≠ c

1. The Shift-And method

P* \ *T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	**1**	**1**	0	0	**1**	0	0
c					**?**	**?**			**?**	

candidates for *i*=3

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	**1**	**1**	0	0	**1**	0	0
c	0	0	0	0	**0**					

c ≠ a

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	**1**	**1**	0	0	**1**	0	0
c	0	0	0	0	**0**	**1**				

c = c

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	**1**	**1**	0	0	**1**	0	0
c	0	0	0	0	**0**	**1**	0	0	**1**	0

c = c

1. The Shift-And method

We can also calculate *M* by column wise

P \ ***T***	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1									
a	0									
c	0									

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0								
a	0	0								
c	0	0								

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1							
a	0	0	0							
c	0	0	0							

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1						
a	0	0	0	1						
c	0	0	0	0						

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1					
a	0	0	0	1	1					
c	0	0	0	0	0					

1. The Shift-And method

We can also calculate *M* by column wise

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0				
a	0	0	0	1	1	0				
c	0	0	0	0	0	1				

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1			
a	0	0	0	1	1	0	0			
c	0	0	0	0	0	1	0			

1. The Shift-And method

We can also calculate *M* by column wise

	1	2	3	4	5	6	7	8	9	10
P \\ ***T***	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1		
a	0	0	0	1	1	0	0	1		
c	0	0	0	0	0	1	0	0		

1. The Shift-And method

We can also calculate *M* by column wise

P \ ***T***	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	
a	0	0	0	1	1	0	0	1	0	
c	0	0	0	0	0	1	0	0	1	

1. The Shift-And method

We can also calculate *M* by column wise

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

1. The Shift-And method

Definition

vector $U(x)$

$U(x)$ is set to 1 for the positions in P where character x appears.

ex)

P
a
b
a
c
d
c
a
b

$U(a)$	$U(b)$	$U(c)$
1	0	0
0	1	0
1	0	0
0	0	1
0	0	0
0	0	1
1	0	0
0	1	0

1. The Shift-And method

Definition

vector *U*(*x*)

U(*x*) is set to 1 for the positions in *P* where character *x* appears.

ex)

P
a
b
a
c
d
c
a
b

U(*a*)	*U*(*b*)	*U*(*c*)
1	0	0
0	**1**	0
1	0	0
0	0	1
0	0	0
0	0	1
1	0	0
0	**1**	0

1. The Shift-And method

Definition

vector *U*(*x*)

$U(x)$ is set to 1 for the positions in P where character x appears.

ex)

P
a
b
a
c
d
c
a
b

$U(a)$	$U(b)$	$U(c)$
1	0	0
0	1	0
1	0	0
0	0	**1**
0	0	0
0	0	**1**
1	0	0
0	1	0

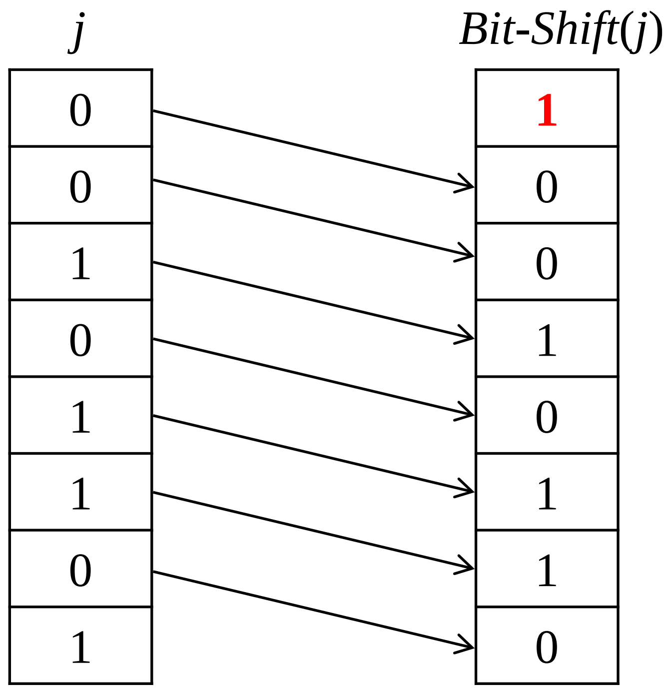
1. The Shift-And method

Definition

***Bit-Shift*(*j*)**

Bit-Shift(*j*) is the vector derived by shifting the vector for column *j* down by one position and setting that first to 1. The previous bit in position n disappears. In other words, *Bit-Shift*(*j*) consists of 1 followed by the first *n*-1 bits of column *j*.

ex)



1. The Shift-And method

P \ *T*	a	b	a	a	a	c	a	a	c	b
	1	2	3	4	5	6	7	8	9	10
a	1									
a	0									
c	0									

$$U(\underset{T(1)}{\underset{\uparrow}{a}}) \quad = \quad \mathbf{M(1)}$$

1
0
0

1. The Shift-And method

P \ T	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	**1**	**0**								
a	**0**	**0**								
c	**0**	**0**								

***Bit-Shift*(1)** **AND** ***U*(*b*)** **=** **M(2)**

***T*(2)**

1	1		0		0
0	1	**AND**	0	=	0
0	0		0		0

1. The Shift-And method

P \ T	1	2	3	4	5	6	7	8	9	10
	a	b	**a**	a	a	c	a	a	c	b
a	1	**0**	**1**							
a	0	**0**	**0**							
c	0	**0**	**0**							

***Bit-Shift*(2)** **AND** ***U*(*a*)** **=** **M(3)**

***T*(3)**

$$\begin{pmatrix}0\\0\\0\end{pmatrix} \quad \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \textbf{AND} \quad \begin{pmatrix}1\\1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 **a**	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	**1**	**1**						
a	0	0	**0**	**1**						
c	0	0	**0**	**0**						

***Bit-Shift*(3)** **AND** ***U*(*a*)** **=** **M(4)**

***T*(4)**

1	1		1		1
0	1	**AND**	1	**=**	1
0	0		0		0

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 a	5 **a**	6 c	7 a	8 a	9 c	10 b
a	1	0	1	**1**	**1**					
a	0	0	0	**1**	**1**					
c	0	0	0	**0**	**0**					

***Bit-Shift*(4)** **AND** ***U*(*a*)** **=** **M(5)**

***T*(5)**

$$\begin{bmatrix}1\\1\\0\end{bmatrix} \quad \begin{bmatrix}1\\1\\1\end{bmatrix} \quad \textbf{AND} \quad \begin{bmatrix}1\\1\\0\end{bmatrix} = \begin{bmatrix}1\\1\\0\end{bmatrix}$$

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 **c**	7 a	8 a	9 c	10 b
a	1	0	1	1	**1**	**0**				
a	0	0	0	1	**1**	**0**				
c	0	0	0	0	**0**	**1**				

***Bit-Shift*(5)** **AND** ***U*(*c*)** **=** **M(6)**

***T*(6)**

$$\begin{bmatrix}1\\1\\0\end{bmatrix} \quad \begin{bmatrix}1\\1\\1\end{bmatrix} \quad \textbf{AND} \quad \begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

1. The Shift-And method

P \ *T*	1 a	2 b	3 a	4 a	5 a	6 c	7 a	8 a	9 c	10 b
a	1	0	1	1	1	0	1	1	0	0
a	0	0	0	1	1	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

$$M(j) = \textit{Bit-Shift}(j-1) \ \textbf{AND} \ U(T(j))$$

1. The Shift-And method

Time complexity

- In worst case the number of bit operations is clearly $\Theta(nm)$.

- if $m \leq$ word size
  → "*Bit-Shift*(j-1) AND U($T(j)$)" can be calculated in constant time
  → $\Theta(n)$

2. *agrep*: The Shift-And method with errors

agrep

- Amplification of the Shift-And method by **finding inexact occurrences** of a pattern in a text.
- Find all occurrences of P in T **within given k mismatches**.
- Find all matched positions of $P[1..i]$ in T allowing up to k mismatches.

2. *agrep*: The Shift-And method with errors

agrep

- *Shift-And* method by finding inexact occurrences of a pattern in a text.

  (occurs with a “small” number of *mismatches* or *inserted* or *deleted* characters)

2. *agrep*: The Shift-And method with errors

agrep

- *Shift-And* method by finding inexact occurrences of a pattern in a text.

  (occurs with a "small" number of *mismatches* or *inserted* or *deleted* characters)

ex)

	1	2	3	4	5	6	7	8	9	10	11
T =	*a*	***a***	***t***	***a***	***t***	***c***	*c*	*c*	*c*	*a*	*a*
P =		***a***	***t***	***a***	***t***	***c***					

Pattern is found at Text position 2 with 0 mismatch.

2. *agrep*: The Shift-And method with errors

agrep

- *Shift-And* method by finding inexact occurrences of a pattern in a text.

  (occurs with a "small" number of *mismatches* or *inserted* or *deleted* characters)

ex)

	1	2	3	4	5	6	7	8	9	10	11
T =	*a*	*a*	*t*	***a***	***t***	***c***	***c***	***c***	*c*	*a*	*a*
P =				***a***	***t***	***a***	***t***	***c***			

Pattern is found at Text position 4 with 2 mismatch.

2. *agrep*: The Shift-And method with errors

agrep

- *Shift-And* method by finding inexact occurrences of a pattern in a text.

  (occurs with a "small" number of *mismatches* or *inserted* or *deleted* characters)

ex)

	1	2	3	4	5	6	7	8	9	10	11
T =	*a*	*a*	*t*	*a*	***t***	***c***	***c***	***c***	***c***	*a*	*a*
P =					***a***	***t***	***a***	***t***	***c***		

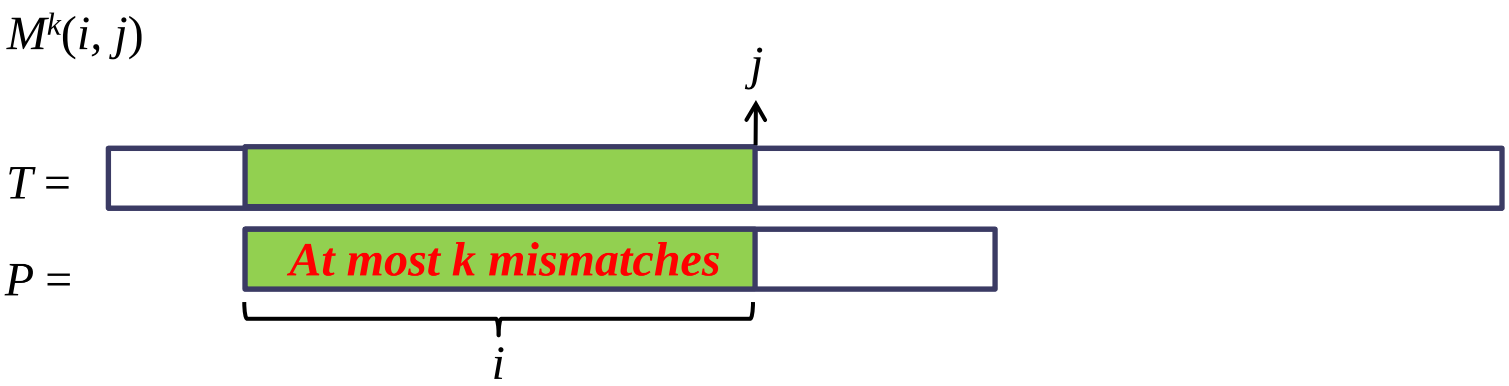
Pattern is found at Text position 5 with 4 mismatch.

2. *agrep*: The Shift-And method with errors

Definition

M^k

For two strings P and T of lengths n and m, let M^k be a binary-valued array, where $M^k(i, j)$ is 1 if and only if at least i-k of the first i characters of P match the i characters up through character j of T.



2. *agrep*: The Shift-And method with errors

Definition

M^k

For two strings P and T of lengths n and m, let M^k be a binary-valued array, where $M^k(i, j)$ is 1 if and only if at least i-k of the first i characters of P match the i characters up through character j of T.

ex)

$T =$

1	2	3	4	5	6	7	8	9	10	11
a	*a*	*t*	***a***	***t***	*c*	*c*	*c*	*c*	*a*	*a*

$P =$

			a	***t***	*a*	*t*	*c*			

$M^0(2,5) = 1$

2. *agrep*: The Shift-And method with errors

Definition

M^k

For two strings P and T of lengths n and m, let M^k be a binary-valued array, where $M^k(i, j)$ is 1 if and only if at least i-k of the first i characters of P match the i characters up through character j of T.

ex)

	1	2	3	4	5	6	7	8	9	10	11
T =	*a*	*a*	*t*	***a***	***t***	***c***	*c*	*c*	*c*	*a*	*a*
P =				***a***	***t***	***a***	*t*	*c*			

$M^0(2,5) = 1$

$M^0(3,6) = 0$

$M^1(3,6) = 1$

2. *agrep*: The Shift-And method with errors

Definition

M^k

For two strings P and T of lengths n and m, let M^k be a binary-valued array, where $M^k(i, j)$ is 1 if and only if at least i-k of the first i characters of P match the i characters up through character j of T.

ex)

T =

1	2	3	4	5	6	7	8	9	10	11
a	*a*	*t*	***a***	***t***	***c***	***c***	*c*	*c*	*a*	*a*

P =

a	***t***	***a***	***t***	*c*

$M^0(2,5) = 1$

$M^0(3,6) = 0$

$M^1(3,6) = 1$

$M^0(4,7) = 0$

$M^1(4,7) = 0$

$M^2(4,7) = 1$

2. *agrep*: The Shift-And method with errors

Definition

M^k

For two strings P and T of lengths n and m, let M^k be a binary-valued array, where $M^k(i, j)$ is 1 if and only if at least i-k of the first i characters of P match the i characters up through character j of T.

ex)

T =

1	2	3	4	5	6	7	8	9	10	11
a	*a*	*t*	***a***	***t***	***c***	***c***	*c*	*c*	*a*	*a*

P =

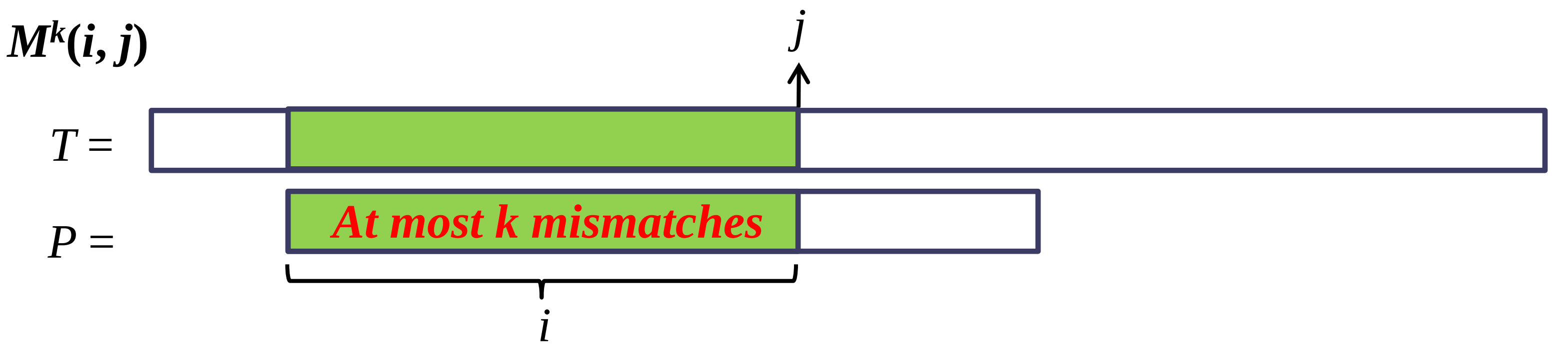
a	***t***	***a***	***t***	*c*

$M^0(2,5) = 1$ $M^0(3,6) = 0$ $M^0(4,7) = 0$

$M^1(2,5) = 1$ $M^1(3,6) = 1$ $M^1(4,7) = 0$

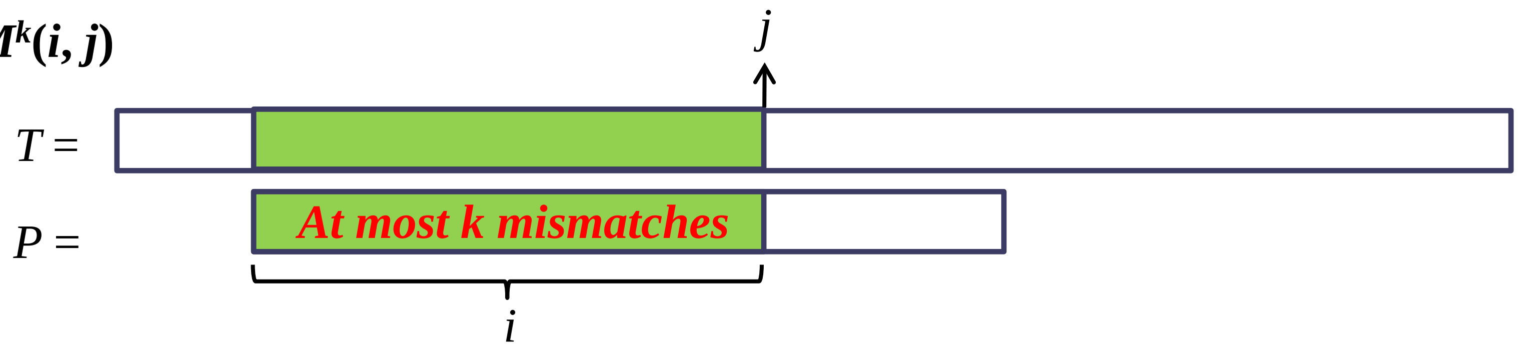
$M^2(2,5) = 1$ $M^2(3,6) = 1$ $M^2(4,7) = 1$

2. *agrep*: The Shift-And method with errors

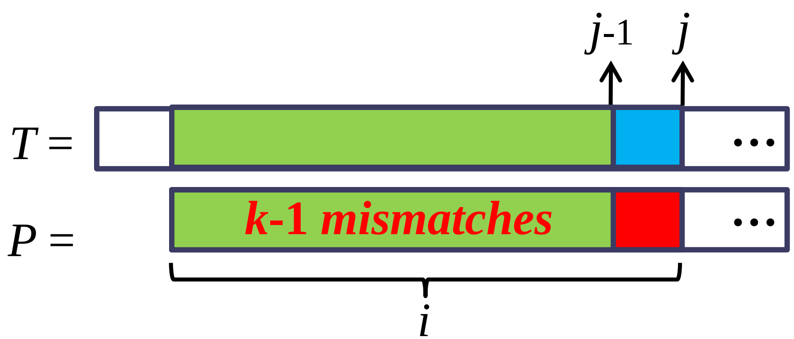


2. *agrep*: The Shift-And method with errors

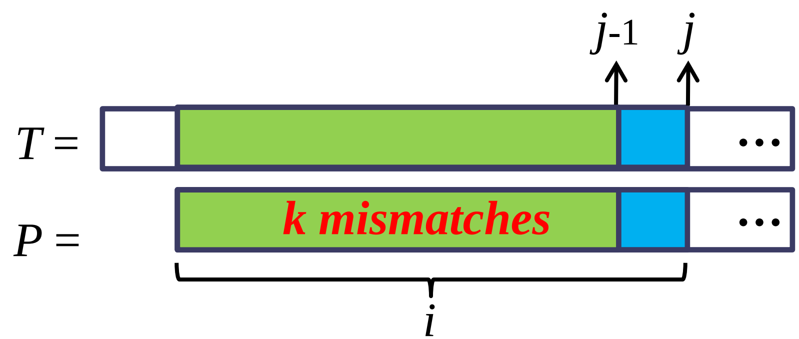
$\mathbf{M^k(i, j)}$



1) $M^{k-1}(i-1, j-1)=1$

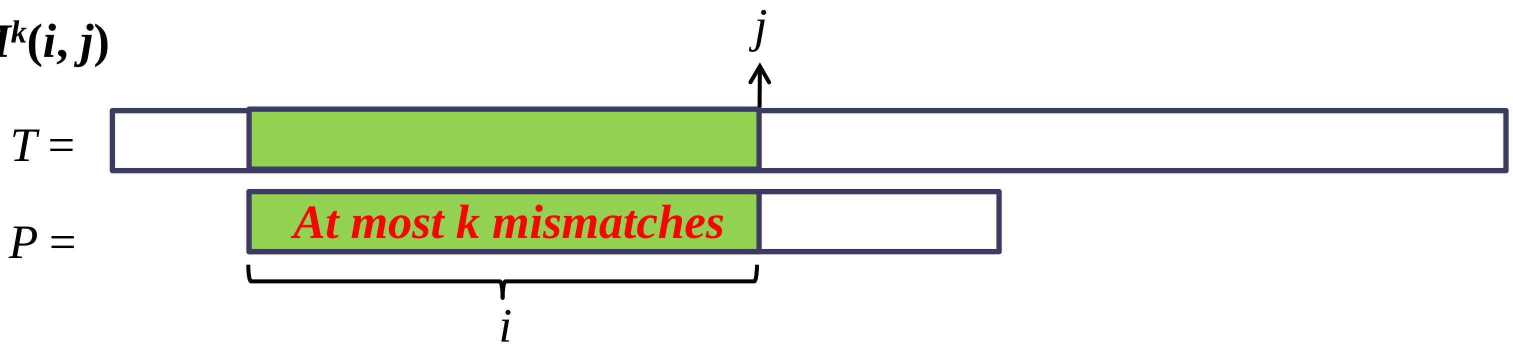


2) $M^{k}(i-1, j-1) = 1$ & $T(j)=P(i)$

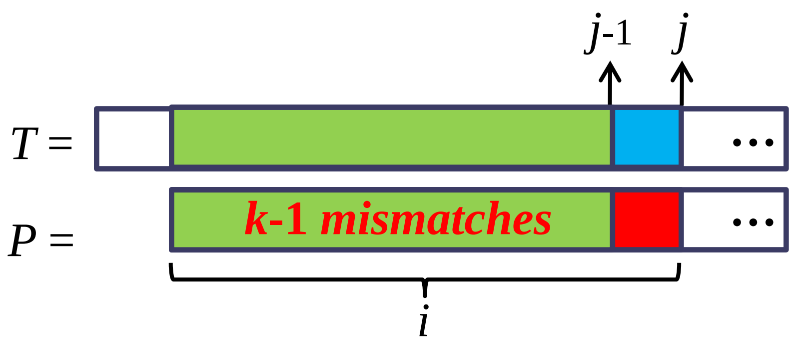


2. *agrep*: The Shift-And method with errors

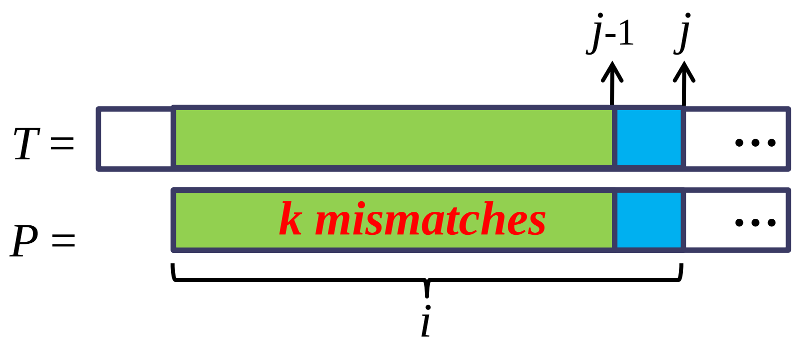
$M^k(i, j)$



1) $M^{k-1}(i-1, j-1)=1$



2) $M^{k}(i-1, j-1) = 1 \quad \& \quad T(j)=P(i)$



$$M^k(j) \quad = \quad Bit\text{-}Shift(\, M^{k-1}(j-1)\,) \qquad \text{OR} \qquad [\; Bit\text{-}Shift(M^k(j-1)) \;\text{ AND }\; U(T(j)) \;]$$

$$M^k(j) \;=\; \textit{Bit-Shift}(M^{k-1}(j\text{-}1)) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^k(j\text{-}1)) \;\text{AND}\; U(T(j))\,]$$

ex)

	1	2	3	4	5	6	7	8
T =	*a*	*a*	*t*	*a*	*t*	*c*	*c*	*a*
P =	*a*	*t*	*c*	*g*				

Find $M^2(5)$

$$M^{k}(j) \;=\; \textit{Bit-Shift}(M^{k-1}(j-1)) \quad \textbf{OR} \quad [\, \textit{Bit-Shift}(M^{k}(j-1)) \;\textbf{AND}\; U(T(j)) \,]$$

ex)

$T =$

1	2	3	4	5	6	7	8
a	*a*	*t*	*a*	*t*	*c*	*c*	*a*

$P =$

a	*t*	*c*	*g*

Find $M^2(5)$

$M^1(i, j)$

	$i \backslash j$	0	1 (*a*)	2 (*a*)	3 (*t*)	4 (*a*)	5 (*t*)	6 (*c*)	7 (*c*)	8 (*a*)
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

$M^2(i, j)$

	$i \backslash j$	0	1 (*a*)	2 (*a*)	3 (*t*)	4 (*a*)	5 (*t*)	6 (*c*)	7 (*c*)	8 (*a*)
a	1	0	1	1	1	1				
t	2	0	0	1	1	1	?			
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$\boldsymbol{M^{k}(j) \quad = \quad \textit{Bit-Shift}(\, M^{k\text{-}1}(j\text{-}1)\,) \quad \textbf{OR} \quad [\, \textit{Bit-Shift}(M^{k}(j\text{-}1)) \ \textbf{AND} \ U(T(j))\,]}$$

			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	$M^1(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	$M^2(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1				
t	2	0	0	1	1	1				
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$M^2(5) \quad = \quad \textit{Bit-Shift}(\, M^1(4)\,) \quad \text{OR} \quad [\, \textit{Bit-Shift}(M^2(4)) \quad \text{AND} \quad U(\, T(4)\,)\,]$$

$$M^{k}(j) \;=\; \textit{Bit-Shift}(\,M^{k-1}(j\text{-}1)\,) \quad \textbf{OR} \quad [\,\textit{Bit-Shift}(M^{k}(j\text{-}1)) \;\textbf{AND}\; U(T(j))\,]$$

$M^1(i, j)$

		j: 0	1	2	3	4	5	6	7	8
	i		a	a	t	a	t	c	c	a
a	1	0	1	1	1	**1**	1	1	1	1
t	2	0	0	1	1	**0**	1	0	0	0
c	3	0	0	0	0	**1**	0	1	0	0
g	4	0	0	0	0	**0**	0	0	1	0

$M^2(i, j)$

		j: 0	1	2	3	4	5	6	7	8
	i		a	a	t	a	t	c	c	a
a	1	0	1	1	1	1				
t	2	0	0	1	1	1				
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$M^2(5) \;=\; \textit{Bit-Shift}(\,\mathbf{M^1(4)}\,) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^2(4)) \quad \text{AND} \quad U(\,T(4)\,)\,]$$

$M^1(4)$

1
0
1
0

$$M^{k}(j) \;=\; \textit{Bit-Shift}(\,M^{k-1}(j-1)\,) \quad \textbf{OR} \quad [\,\textit{Bit-Shift}(M^{k}(j-1)) \;\textbf{AND}\; U(T(j))\,]$$

	$M^1(i, j)$									
			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	i \ *j*	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	**1**	1	1	1	1
t	2	0	0	1	1	**0**	1	0	0	0
c	3	0	0	0	0	**1**	0	1	0	0
g	4	0	0	0	0	**0**	0	0	1	0

	$M^2(i, j)$									
			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	i \ *j*	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1				
t	2	0	0	1	1	1				
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$M^{2}(5) \;=\; \textit{Bit-Shift}(\,M^{1}(4)\,) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^{2}(4)) \quad \text{AND} \quad U(\,T(4)\,)\,]$$

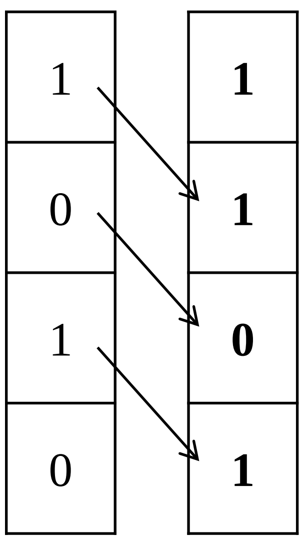
$M^1(4)$	***Bit-Shift*(M^1 (4))**
1	**1**
0	**1**
1	**0**
0	**1**

$$M^k(j) = \textit{Bit-Shift}(M^{k-1}(j-1)) \text{ OR } [\textit{Bit-Shift}(M^k(j-1)) \text{ AND } U(T(j))]$$

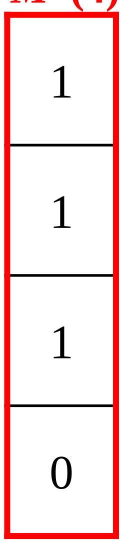
			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	$M^1(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	$M^2(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	**1**				
t	2	0	0	1	1	**1**				
c	3	0	0	0	1	**1**				
g	4	0	0	0	0	**0**				

$$M^2(5) = \textit{Bit-Shift}(M^1(4)) \text{ OR } [\textit{Bit-Shift}(\mathbf{M^2(4)}) \text{ AND } U(T(4))]$$



$M^2(4)$



$$M^k(j) = \textit{Bit-Shift}(M^{k-1}(j-1)) \quad \text{OR} \quad [\, \textit{Bit-Shift}(M^k(j-1)) \ \text{AND} \ U(T(j)) \,]$$

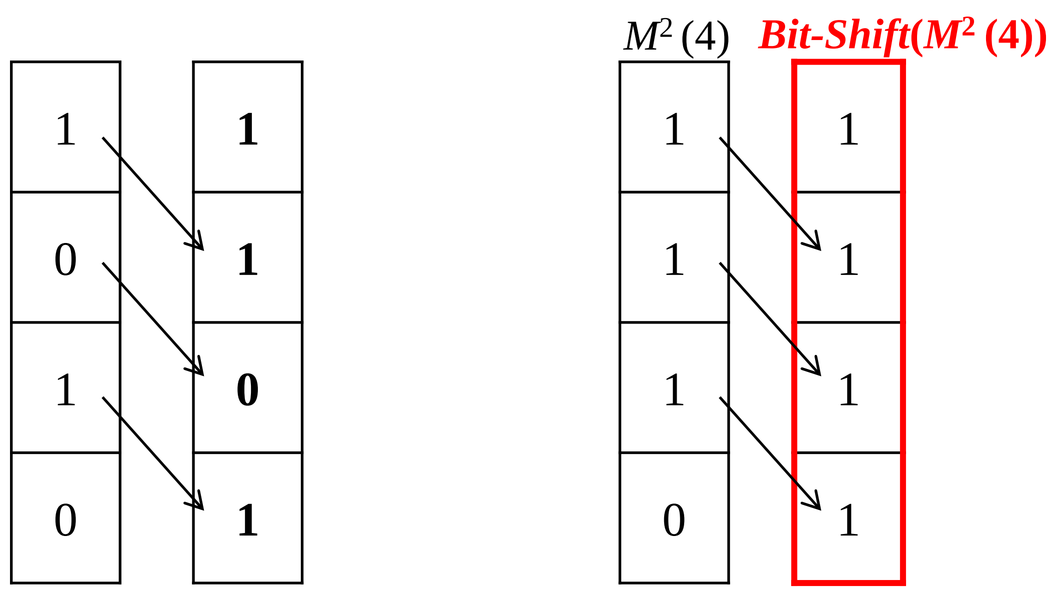
$M^1(i, j)$

	i \ j	0	1 (a)	2 (a)	3 (t)	4 (a)	5 (t)	6 (c)	7 (c)	8 (a)
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

$M^2(i, j)$

	i \ j	0	1 (a)	2 (a)	3 (t)	4 (a)	5 (t)	6 (c)	7 (c)	8 (a)
a	1	0	1	1	1	**1**				
t	2	0	0	1	1	**1**				
c	3	0	0	0	1	**1**				
g	4	0	0	0	0	**0**				

$$M^2(5) = \textit{Bit-Shift}(M^1(4)) \quad \text{OR} \quad [\, \textit{Bit-Shift}(M^2(4)) \quad \text{AND} \quad U(T(4)) \,]$$

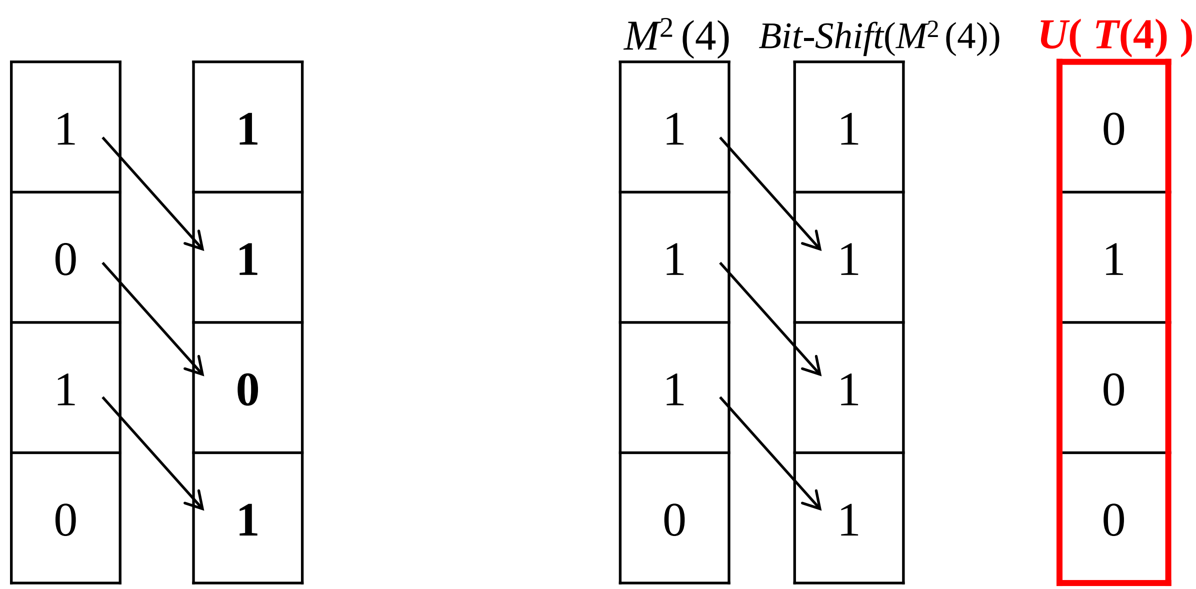


$$M^{k}(j) \;=\; \textit{Bit-Shift}(\,M^{k-1}(j-1)\,) \quad \textbf{OR} \quad [\,\textit{Bit-Shift}(M^{k}(j-1)) \ \textbf{AND} \ U(T(j))\,]$$

		$M^1(i,j)$								
			a	*a*	*t*	*a*	*t*	*c*	*c*	*a*
	i \ *j*	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

		$M^2(i,j)$								
			a	*a*	*t*	*a*	***t***	*c*	*c*	*a*
	i \ *j*	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1				
t	2	0	0	1	1	1				
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$M^2(5) \;=\; \textit{Bit-Shift}(\,M^1(4)\,) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^2(4)) \ \text{AND} \ \mathbf{U(\,\mathit{T}(4)\,)}\,]$$

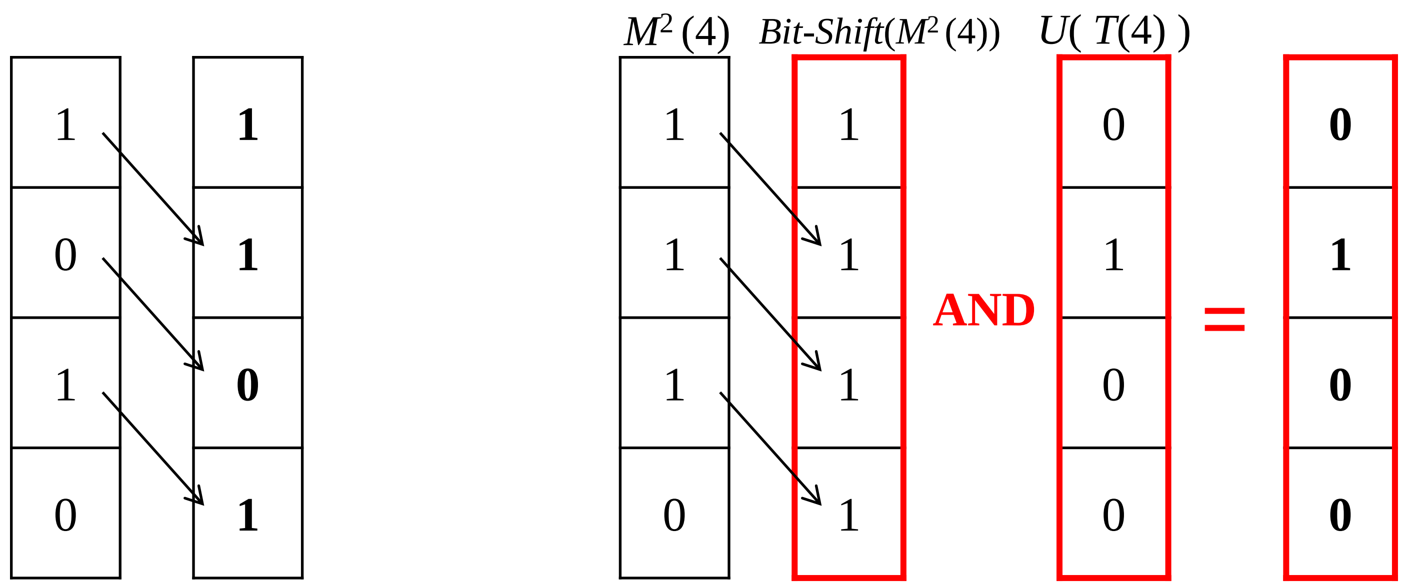


$$M^{k}(j) \;=\; \textit{Bit-Shift}(\,M^{k-1}(j-1)\,) \quad \textbf{OR} \quad [\,\textit{Bit-Shift}(M^{k}(j-1)) \;\textbf{AND}\; U(T(j))\,]$$

			a	a	t	a	t	c	c	a
	$M^1(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

			a	a	t	a	t	c	c	a
	$M^2(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1				
t	2	0	0	1	1	1				
c	3	0	0	0	1	1				
g	4	0	0	0	0	0				

$$M^2(5) \;=\; \textit{Bit-Shift}(\,M^1(4)\,) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^2(4)) \;\textbf{AND}\; U(\,T(4)\,)\,]$$



$$M^{k}(j) \;=\; \textit{Bit-Shift}(\,M^{k-1}(j-1)\,) \quad \textbf{OR} \quad [\,\textit{Bit-Shift}(M^{k}(j-1)) \;\; \textbf{AND} \;\; U(T(j))\,]$$

			a	a	t	a	t	c	c	a
	$M^1(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	1	1	1	1
t	2	0	0	1	1	0	1	0	0	0
c	3	0	0	0	0	1	0	1	0	0
g	4	0	0	0	0	0	0	0	1	0

			a	a	t	a	t	c	c	a
	$M^2(i, j)$									
	$i \backslash j$	0	1	2	3	4	5	6	7	8
a	1	0	1	1	1	1	**1**			
t	2	0	0	1	1	1	**1**			
c	3	0	0	0	1	1	**0**			
g	4	0	0	0	0	0	**1**			

$$\mathbf{M^2(5)} \;=\; \textit{Bit-Shift}(\,M^1(4)\,) \quad \text{OR} \quad [\,\textit{Bit-Shift}(M^2(4)) \quad \text{AND} \quad U(\,T(4)\,)\,]$$

$$\begin{bmatrix}\mathbf{1}\\ \mathbf{1}\\ \mathbf{0}\\ \mathbf{1}\end{bmatrix} = \begin{bmatrix}1\\0\\1\\0\end{bmatrix} \rightarrow \boxed{\begin{bmatrix}\mathbf{1}\\ \mathbf{1}\\ \mathbf{0}\\ \mathbf{1}\end{bmatrix}} \quad \textbf{OR} \quad \underbrace{\begin{bmatrix}1\\1\\1\\0\end{bmatrix}}_{M^2(4)} \rightarrow \underbrace{\begin{bmatrix}1\\1\\1\\1\end{bmatrix}}_{\textit{Bit-Shift}(M^2(4))} \;\textbf{AND}\; \underbrace{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}_{U(\,T(4)\,)} = \begin{bmatrix}\mathbf{0}\\ \mathbf{1}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}$$

2. *agrep*: The Shift-And method with errors

Time complexity

- The number of bit operations is $O(knm)$.
  $\rightarrow M^0(i, j),\ M^1(i, j),\ M^2(i, j),\ \ldots\ M^k(i, j)$

- When the pattern is relatively small, so that a column of any M^l fits into a few words, and k is also small. ($k < m \leq$ word size)
  $\rightarrow O(n)$

3. The match-count problem and Fast Fourier Transform

match-count problem

- Find the number of matches for P[1..*i*] and T for all positions.

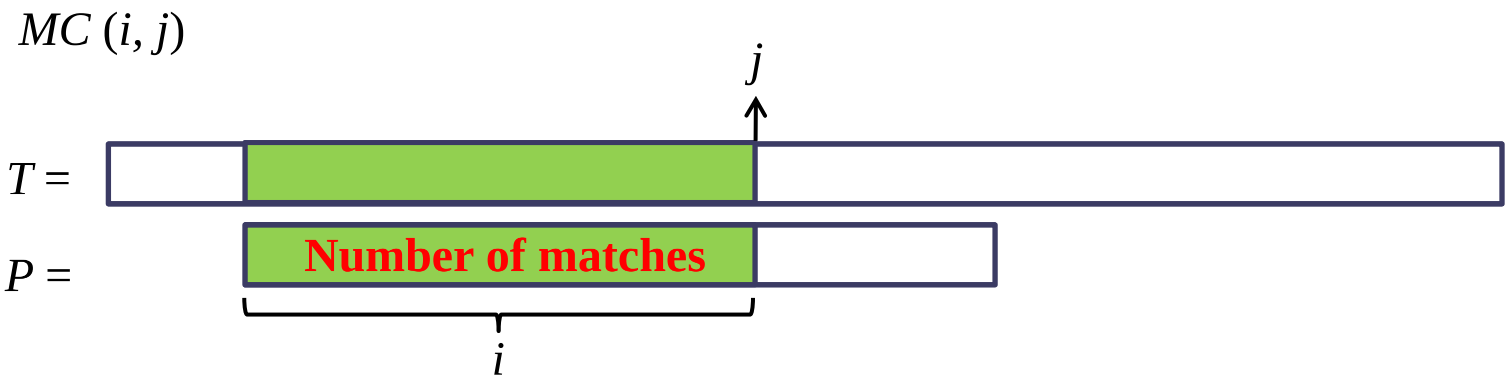
- Similar with *agrep* but we can solve much faster.

3. The match-count problem and Fast Fourier Transform

Definition

MC

The matrix *MC* is an n by $m+1$ integer-valued matrix, where entry $MC(i, j)$ is the number of characters of $P[1..i]$ that match $T[j-1+1..j]$



3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a										
a										
c										

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	**1**	0	**1**	**1**	**1**	0	**1**	**1**	0	0
a										
c										

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1									
c										

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1								
c										

a ≠ b

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	**1**							
c										

+1

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ ***T***	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	**2**						
c										

+1

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	**2**					
c										

+1

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1				
c										

a ≠ c

3. The match-count problem and Fast Fourier Transform

ex)

P \ *T*	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	**1**			
c										

+1

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	**2**		
c										

+1

a = a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	
c										

a ≠ c

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c										

a ≠ b

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0									

c ≠ a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1								

c ≠ b

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1							

$c \neq a$

3. The match-count problem and Fast Fourier Transform

ex)

P \ *T*	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1						

c ≠ a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2					

c ≠ a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2	3				

+1

c = c

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2	3	1			

$c \neq a$

3. The match-count problem and Fast Fourier Transform

ex)

P \ *T*	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2	3	1	1		

c ≠ a

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2	3	1	1	2	

+1

c = c

3. The match-count problem and Fast Fourier Transform

ex)

P \ T	a	b	a	a	a	c	a	a	c	b
a	1	0	1	1	1	0	1	1	0	0
a	1	1	1	2	2	1	1	2	1	0
c	0	1	1	1	2	3	1	1	2	3

c ≠ b

3. The match-count problem and Fast Fourier Transform

Time complexity

- This extension uses $\Theta(nm)$ additions and comparisons.

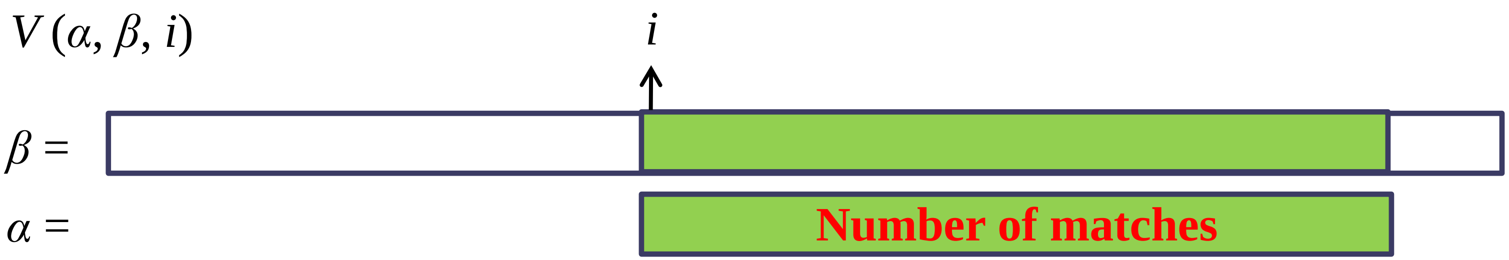
  → Now, We will apply **Fast Fourier Transform** to the match-count problem. Then the time complexity will be $\boldsymbol{O(n \cdot \log n)}$.

3. The match-count problem and Fast Fourier Transform

Definition

V

Define $V(\alpha, \beta, i)$ to be the number of characters of α and β that match when the left end of string α is opposite position i of string β. Define $V(\alpha, \beta)$ to be the vector whose i th entry is $V(\alpha, \beta, i)$.

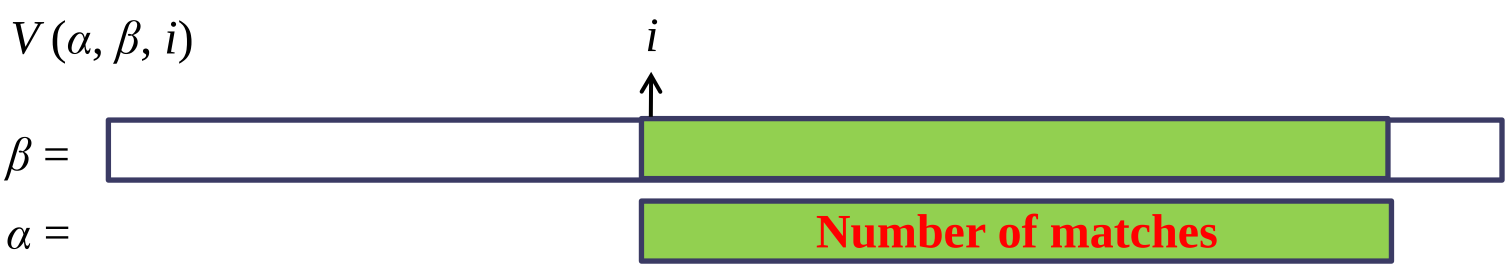


3. The match-count problem and Fast Fourier Transform

Definition

V

Define $V(\alpha, \beta, i)$ to be the number of characters of α and β that match when the left end of string α is opposite position i of string β. Define $V(\alpha, \beta)$ to be the vector whose i th entry is $V(\alpha, \beta, i)$.



- $\alpha = P, \beta = T$
- $|\alpha| = m \leq n = |\beta|$

3. The match-count problem and Fast Fourier Transform

Definition

V

Define $V(\alpha, \beta, i)$ to be the number of characters of α and β that match when the left end of string α is opposite position i of string β. Define $V(\alpha, \beta)$ to be the vector whose i th entry is $V(\alpha, \beta, i)$. $(-m+1 \leq i \leq n)$

ex) $V(\alpha, \beta, 2) = 4$

	1	2	3	4	5	6	7	8	9
$\beta =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$\alpha =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

3. The match-count problem and Fast Fourier Transform

Definition

V

Define $V(\alpha, \beta, i)$ to be the number of characters of α and β that match when the left end of string α is opposite position i of string β. Define $V(\alpha, \beta)$ to be the vector whose i th entry is $V(\alpha, \beta, i)$. $(\, -m+1 \le i \le n \,)$

ex) $V(\alpha, \beta, 4) = 2$

	1	2	3	4	5	6	7	8	9		
$\beta =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*		
$\alpha =$				*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

3. The match-count problem and Fast Fourier Transform

Definition

V

Define $V(\alpha, \beta, i)$ to be the number of characters of α and β that match when the left end of string α is opposite position i of string β. Define $V(\alpha, \beta)$ to be the vector whose i th entry is $V(\alpha, \beta, i)$. $(-m+1 \leq i \leq n)$

ex) $V(\alpha, \beta, -2) = 2$

	-2	-1	1	2	3	4	5	6	7	8	9
$\beta =$			*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$\alpha =$	*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*			

3. The match-count problem and Fast Fourier Transform

Time complexity

- For any fixed i, $V(\alpha, \beta, i)$ can be directly computed in $O(m)$ time.
- So $V(\alpha, \beta)$ can be computed in $O(nm)$ total time.

3. The match-count problem and Fast Fourier Transform

Definition

V_a

Define $V_a(\alpha, \beta, i)$ to be the number of matches of character *a* that occur when the start of string α is positioned opposite position *i* of string β. $V_a(\alpha, \beta)$ is the $(n+m)$-length vector holding these values.

ex)

$V(\alpha, \beta, 2) = 4$

	1	2	3	4	5	6	7	8	9
$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

$V_c(\alpha, \beta, 2) = 2$

	1	2	3	4	5	6	7	8	9
$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

3. The match-count problem and Fast Fourier Transform

ex)

$V_a(\alpha, \beta, 2) = 0$

$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

$V_t(\alpha, \beta, 2) = 1$

$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

$V_c(\alpha, \beta, 2) = 2$

$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

\+ $V_g(\alpha, \beta, 2) = 1$

$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

$V(\alpha, \beta, 2) = 4$

$T =$	*a*	*c*	*c*	*c*	*t*	*g*	*t*	*c*	*c*
$P =$		*a*	*a*	*c*	*t*	*g*	*c*	*c*	*g*

3. The match-count problem and Fast Fourier Transform

ex)

$V_a(\alpha, \beta, 2) = 0$

$T =$	a	c	c	c	t	g	t	c	c
$P =$		a	a	c	t	g	c	c	g

$V_t(\alpha, \beta, 2) = 1$

$T =$	a	c	c	c	t	g	t	c	c
$P =$		a	a	c	t	g	c	c	g

$V_c(\alpha, \beta, 2) = 2$

$T =$	a	c	c	c	t	g	t	c	c
$P =$		a	a	c	t	g	c	c	g

$+ \quad V_g(\alpha, \beta, 2) = 1$

$T =$	a	c	c	c	t	g	t	c	c
$P =$		a	a	c	t	g	c	c	g

$V(\alpha, \beta, 2) = 4$

$T =$	a	c	c	c	t	g	t	c	c
$P =$		a	a	c	t	g	c	c	g

$$V(\alpha, \beta, i) = V_a(\alpha, \beta, i) + V_t(\alpha, \beta, i) + V_c(\alpha, \beta, i) + V_g(\alpha, \beta, i)$$

3. The match-count problem and Fast Fourier Transform

The high-level approach

- How to compute $V_a(\alpha, \beta, i)$ for each i.

$$\begin{array}{llllllllll} \beta: & a & c & c & c & t & g & t & c & c \\ \bar{\beta}_c: & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

$$\begin{array}{lllllllll} \alpha: & a & a & c & t & g & c & c & g \\ \bar{\alpha}_c: & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

3. The match-count problem and Fast Fourier Transform

The high-level approach

- How to compute $V_a(\alpha, \beta, i)$ for each i.

$$\beta : \; a \; c \; c \; c \; t \; g \; t \; c \; c \qquad\qquad \alpha : \; a \; a \; c \; t \; g \; c \; c \; g$$

$$\bar{\beta}_c : \; 0 \; 1 \; 1 \; 1 \; 0 \; 0 \; 0 \; 1 \; 1 \qquad\qquad \bar{\alpha}_c : \; 0 \; 0 \; 1 \; 0 \; 0 \; 1 \; 1 \; 0$$

$$V(\alpha, \beta, -7) \qquad \begin{array}{ll} \bar{\beta}_c : & \overbrace{0 \; 1 \; 1 \; 1 \; 0 \; 0 \; 0 \; 1 \; 1}^{n} \\ \bar{\alpha}_c : & \underbrace{0 \; 0 \; 1 \; 0 \; 0 \; 1 \; 1 \; 0}_{m} \end{array}$$

3. The match-count problem and Fast Fourier Transform

The high-level approach

- How to compute $V_a(\alpha, \beta, i)$ for each i.

β : a c c c t g t c c

$\bar{\beta}_c$: 0 1 1 1 0 0 0 1 1

α : a a c t g c c g

$\bar{\alpha}_c$: 0 0 1 0 0 1 1 0

$V(\alpha, \beta, -6)$

$$\bar{\beta}_c : \qquad\qquad \overbrace{0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1}^{n}$$

$$\bar{\alpha}_c : \underbrace{0\ 0\ 1\ 0\ 0\ 1\ 1\ 0}_{m}$$

3. The match-count problem and Fast Fourier Transform

The high-level approach

- How to compute $V_a(\alpha, \beta, i)$ for each i.

β : a c c c t g t c c

$\bar{\beta}_c$: 0 1 1 1 0 0 0 1 1

α : a a c t g c c g

$\bar{\alpha}_c$: 0 0 1 0 0 1 1 0

$V(\alpha, \beta, -5)$

$\bar{\beta}_c$: $\overbrace{0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1}^{n}$

$\bar{\alpha}_c$: $\underbrace{0\ 0\ 1\ 0\ 0\ 1\ 1\ 0}_{m}$

3. The match-count problem and Fast Fourier Transform

The high-level approach

- How to compute $V_a(\alpha, \beta, i)$ for each i.

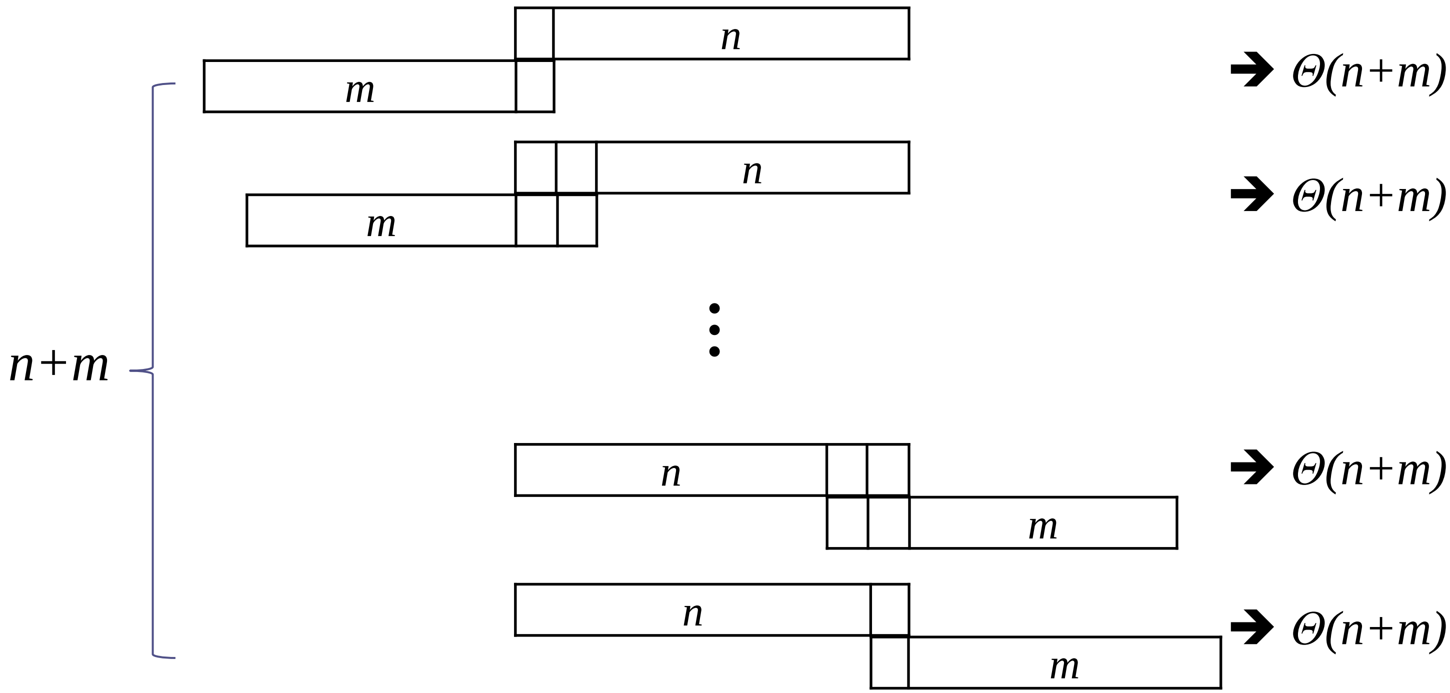
$$\beta : \ a\ c\ c\ c\ t\ g\ t\ c\ c \qquad\qquad \alpha : \ a\ a\ c\ t\ g\ c\ c\ g$$

$$\bar{\beta}_c : \ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1 \qquad\qquad \bar{\alpha}_c : \ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0$$

$$V(\alpha, \beta, 9) \qquad \begin{array}{ll} \bar{\beta}_c : & \overbrace{0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1}^{n} \\ \bar{\alpha}_c : & \underbrace{0\ 0\ 1\ 0\ 0\ 1\ 1\ 0}_{m} \end{array}$$

3. The match-count problem and Fast Fourier Transform

Time complexity of $V_a(\alpha, \beta)$



- $V_a(\alpha, \beta)$ can be computed in $\Theta((n+m)^2)$ total time.
  $\rightarrow \Theta(n^2)$

3. The match-count problem and Fast Fourier Transform

- **Handling wild cards in match-counts**
  - The wild card
    - The wild card symbol ϕ matches any other single character.

    - Ex) $V(\alpha, \beta, 1) = 7$

	1	2	3	4	5	6	7	8
$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	$\boldsymbol{a}$	$\boldsymbol{t}$	c	t	$\boldsymbol{g}$	t

3. The match-count problem and Fast Fourier Transform

- **Handling wild cards in match-counts**
  - The wild card
    - The wild card symbol ϕ matches any other single character.
    - When the wild cards only occur in one of the two strings…
    - Ex) $V_a(\alpha, \beta, 1) = 2$

The wild card symbol ϕ changes to 1

	1	2	3	4	5	6	7	8
$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	a	t	c	t	g	t

	1	2	3	4	5	6	7	8
$\bar{\beta}_a =$	1	0	1	1	0	0	1	1
$\bar{\alpha}_a =$	1	0	1	0	0	0	0	0

3. The match-count problem and Fast Fourier Transform

- When the wild cards only occur in one of the two strings…

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	a	t	c	t	g	t

3. The match-count problem and Fast Fourier Transform

- When the wild cards only occur in one of the two strings…

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	a	t	c	t	g	t

$V_a(\alpha, \beta, 1) = 2$

$\overline{\beta}_a =$	1	0	1	1	0	0	1	1
$\overline{\alpha}_a =$	1	0	1	0	0	0	0	0

$V_t(\alpha, \beta, 1) = 2$

$\overline{\beta}_t =$	0	0	1	1	0	1	1	0
$\overline{\alpha}_t =$	0	0	0	1	0	1	0	1

$V_c(\alpha, \beta, 1) = 1$

$\overline{\beta}_c =$	0	0	1	1	1	0	1	0
$\overline{\alpha}_c =$	0	0	0	0	1	0	0	0

$V_g(\alpha, \beta, 1) = 2$

$\overline{\beta}_g =$	0	1	1	1	0	0	1	0
$\overline{\alpha}_g =$	0	1	0	0	0	0	1	0

3. The match-count problem and Fast Fourier Transform

- When the wild cards only occur in one of the two strings…

$V(\alpha, \beta, 1)$

$\beta =$	*a*	*g*	ϕ	ϕ	*c*	*t*	ϕ	*a*
$\alpha =$	*a*	*g*	***a***	***t***	*c*	*t*	***g***	*t*

$V_a(\alpha, \beta, 1) = 2$

$\overline{\beta}_a=$	1	0	1	1	0	0	1	1
$\overline{\alpha}_a=$	1	0	1	0	0	0	0	0

$V_t(\alpha, \beta, 1) = 2$

$\overline{\beta}_t=$	0	0	1	1	0	1	1	0
$\overline{\alpha}_t=$	0	0	0	1	0	1	0	1

$V_c(\alpha, \beta, 1) = 1$

$\overline{\beta}_c=$	0	0	1	1	1	0	1	0
$\overline{\alpha}_c=$	0	0	0	0	1	0	0	0

$V_g(\alpha, \beta, 1) = 2$

$\overline{\beta}_g=$	0	1	1	1	0	0	1	0
$\overline{\alpha}_g=$	0	1	0	0	0	0	1	0

$$V(\alpha, \beta, 1) = V_a(\alpha, \beta, 1) + V_t(\alpha, \beta, 1) + V_c(\alpha, \beta, 1) + V_g(\alpha, \beta, 1)$$

3. The match-count problem and Fast Fourier Transform

- When the wild cards occur in both α and β

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	c	ϕ	c	t	ϕ	t

3. The match-count problem and Fast Fourier Transform

- When the wild cards occur in both α and β

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	c	ϕ	c	t	ϕ	t

$V_a(\alpha, \beta, 1) = 4$

$\bar{\beta}_a =$	1	0	1	1	0	0	1	1
$\bar{\alpha}_a =$	1	0	1	1	0	0	1	0

$V_t(\alpha, \beta, 1) = 3$

$\bar{\beta}_t =$	0	0	1	1	0	1	1	0
$\bar{\alpha}_t =$	0	0	0	1	0	1	1	1

$V_c(\alpha, \beta, 1) = 3$

$\bar{\beta}_c =$	0	0	1	1	1	0	1	0
$\bar{\alpha}_c =$	0	0	0	1	1	0	1	0

$V_g(\alpha, \beta, 1) = 3$

$\bar{\beta}_g =$	0	1	1	1	0	0	1	0
$\bar{\alpha}_g =$	0	1	0	1	0	0	1	0

3. The match-count problem and Fast Fourier Transform

- When the wild cards occur in both α and β

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	c	ϕ	c	t	ϕ	t

$V_a(\alpha, \beta, 1) = 4$

$\overline{\beta}_a =$	1	0	1	1	0	0	1	1
$\overline{\alpha}_a =$	1	0	1	1	0	0	1	0

$V_t(\alpha, \beta, 1) = 3$

$\overline{\beta}_t =$	0	0	1	1	0	1	1	0
$\overline{\alpha}_t =$	0	0	0	1	0	1	1	1

$V_c(\alpha, \beta, 1) = 3$

$\overline{\beta}_c =$	0	0	1	1	1	0	1	0
$\overline{\alpha}_c =$	0	0	0	1	1	0	1	0

$V_g(\alpha, \beta, 1) = 3$

$\overline{\beta}_g =$	0	1	1	1	0	0	1	0
$\overline{\alpha}_g =$	0	1	0	1	0	0	1	0

Counted as a match for each characters

3. The match-count problem and Fast Fourier Transform

- When the wild cards occur in both α and β

$V(\alpha, \beta, 1)$

β =	a	g	ϕ	ϕ	c	t	ϕ	a
α =	a	g	c	ϕ	c	t	ϕ	t

$V_\phi(\alpha, \beta, 1) = 2$

$\bar{\beta}_\phi$=	0	0	1	1	0	0	1	0
$\bar{\alpha}_\phi$=	0	0	0	1	0	0	1	0

➔ The number of repeated

$V_a(\alpha, \beta, 1) = 4$

$\bar{\beta}_a$=	1	0	1	1	0	0	1	1
$\bar{\alpha}_a$=	1	0	1	1	0	0	1	0

$V_t(\alpha, \beta, 1) = 3$

$\bar{\beta}_t$=	0	0	1	1	0	1	1	0
$\bar{\alpha}_t$=	0	0	0	1	0	1	1	1

$V_c(\alpha, \beta, 1) = 3$

$\bar{\beta}_c$=	0	0	1	1	1	0	1	0
$\bar{\alpha}_c$=	0	0	0	1	1	0	1	0

$V_g(\alpha, \beta, 1) = 3$

$\bar{\beta}_g$=	0	1	1	1	0	0	1	0
$\bar{\alpha}_g$=	0	1	0	1	0	0	1	0

Counted as a match for each characters

3. The match-count problem and Fast Fourier Transform

- When the wild cards occur in both α and β

$V(\alpha, \beta, 1)$

$\beta =$	a	g	ϕ	ϕ	c	t	ϕ	a
$\alpha =$	a	g	c	ϕ	c	t	ϕ	t

$V_\phi(\alpha, \beta, 1) = 2$

$\overline{\beta}_\phi =$	0	0	1	1	0	0	1	0
$\overline{\alpha}_\phi =$	0	0	0	1	0	0	1	0

$V_a(\alpha, \beta, 1) = 4$

$\overline{\beta}_a =$	1	0	1	1	0	0	1	1
$\overline{\alpha}_a =$	1	0	1	1	0	0	1	0

$V_t(\alpha, \beta, 1) = 3$

$\overline{\beta}_t =$	0	0	1	1	0	1	1	0
$\overline{\alpha}_t =$	0	0	0	1	0	1	1	1

$V_c(\alpha, \beta, 1) = 3$

$\overline{\beta}_c =$	0	0	1	1	1	0	1	0
$\overline{\alpha}_c =$	0	0	0	1	1	0	1	0

$V_g(\alpha, \beta, 1) = 3$

$\overline{\beta}_g =$	0	1	1	1	0	0	1	0
$\overline{\alpha}_g =$	0	1	0	1	0	0	1	0

$$V(\alpha, \beta, 1) = V_a(\alpha, \beta, 1) + V_t(\alpha, \beta, 1) + V_c(\alpha, \beta, 1) + V_g(\alpha, \beta, 1) - 3V_\phi(\alpha, \beta, 1) = 7$$

3. The match-count problem and Fast Fourier Transform

- **Handling wild cards in match-counts**

  - $V(\alpha, \beta, i) = \sum_{x \neq \phi} V_x(\alpha, \beta, i) - 3V_\phi(\alpha, \beta, i)$

  - The match-count problem can be solved in $O(n \cdot \log n)$ time even if an unbounded number of wild cards are allowed in either P or T.