NCKU Programming Contest Training Course

2016/04/20

Jingfei Yang

http://myweb.ncku.edu.tw/~e84016184/StringMatching.pdf

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

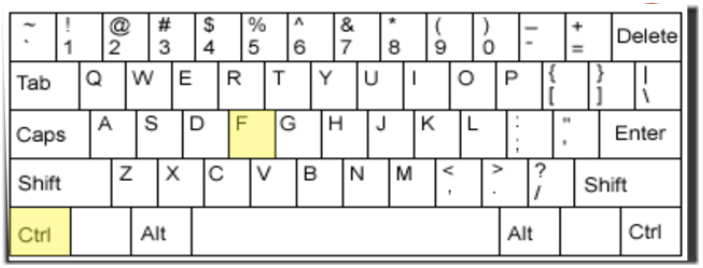
String Basic

- 字串 string
  - 字元的有序序列 $A = a_0 a_1 \ldots a_{n-1}$
  - $a_i \in$ 字元集，n 是字串的長度
- 子字串 substring
  - $A[i,j] = a_i a_{i+1} a_{i+2} \ldots a_j$ (A 連續的一段)
- 子序列 subsequence
  - $B = a_{q_1} a_{q_2} a_{q_3} \ldots a_{q_m}, 0 \le q_1 < q_2 < \cdots < q_m < n$ (不連續)
- 後綴 suffix
  - A 的一個子字串 $S_A(k) = a_k a_{k+1} a_{k+2} \ldots a_n,\ 0 \le k < n$
- 前綴 prefix
  - A 的一個子字串 $P_A(h) = a_0 a_1 a_2 \ldots a_h,\ 0 \le h < n$

String Basic

- S = “abcbbab”
  - 子字串： “bcb”，“bba”，…
  - 子序列： “acb”，“bbb”，…
  - 前綴： “abcb”，“ab”，…
  - 後綴： “bbab”，“ab”，…

String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

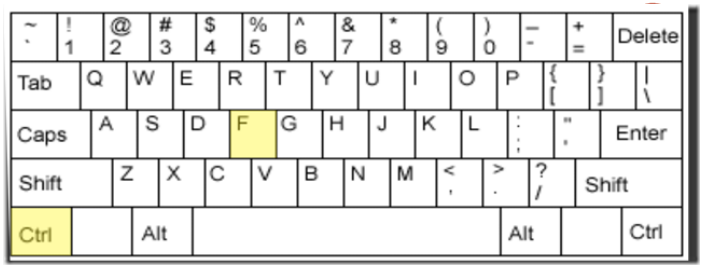
```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

a	b	c	d	e	f	g
c	d	e				

made by Jingfei

String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

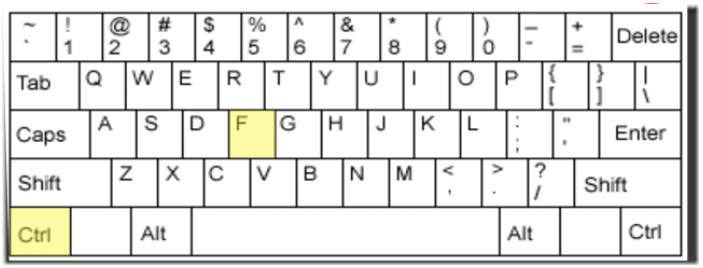
```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

a	b	c	d	e	f	g
	c	d	e			

made by Jingfei

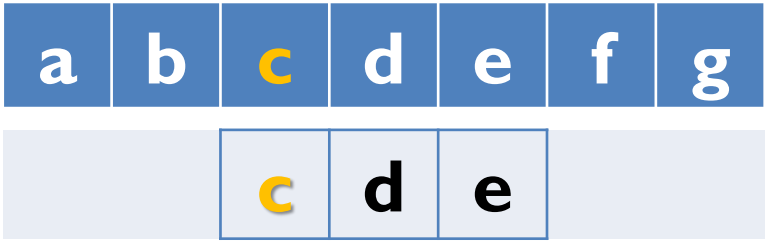
String Matching



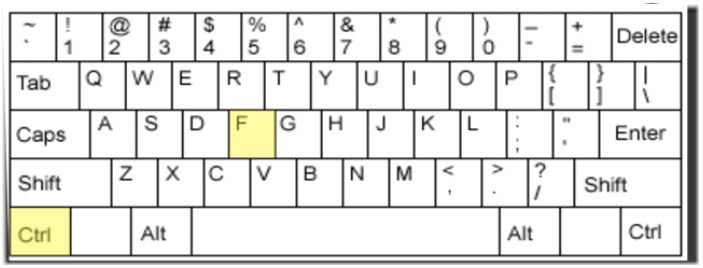
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”



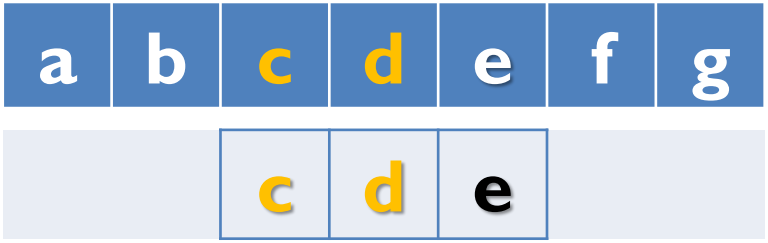
String Matching



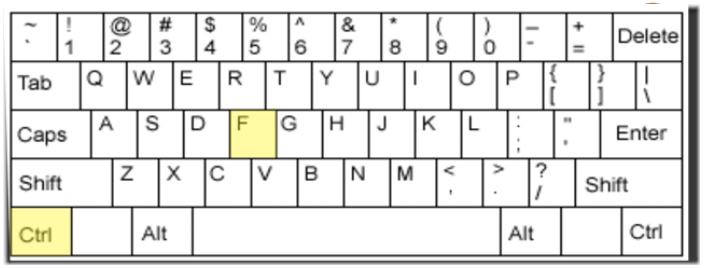
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”



String Matching

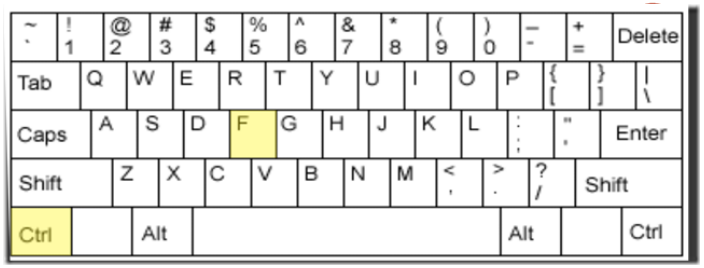


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = "abcdefg"
  B = "cde"

String Matching



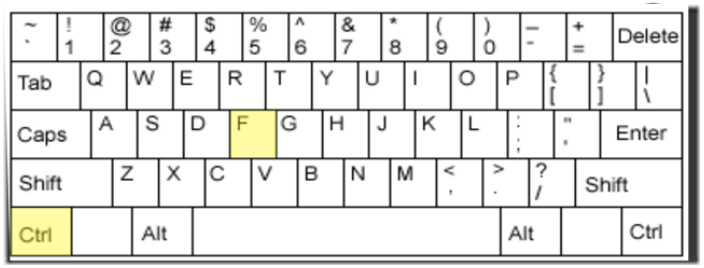
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

a	b	c	d	e	f	g
			c	d	e	

String Matching



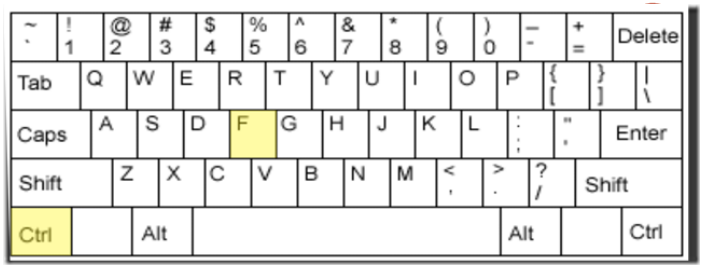
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

a	b	c	d	e	f	g
				c	d	e

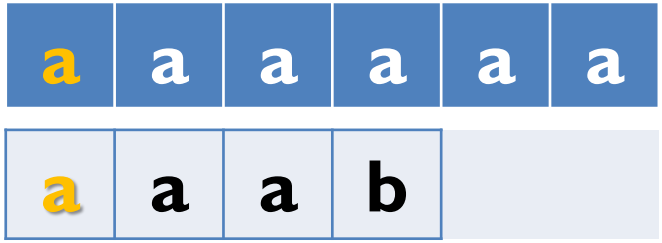
String Matching



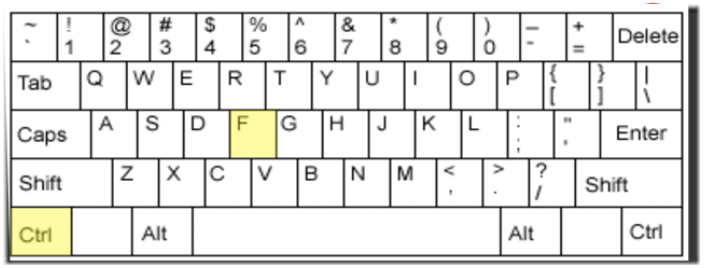
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



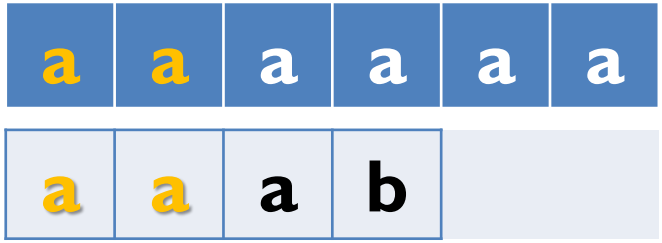
String Matching



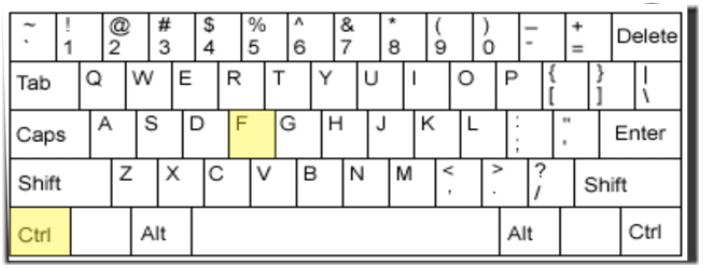
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



String Matching

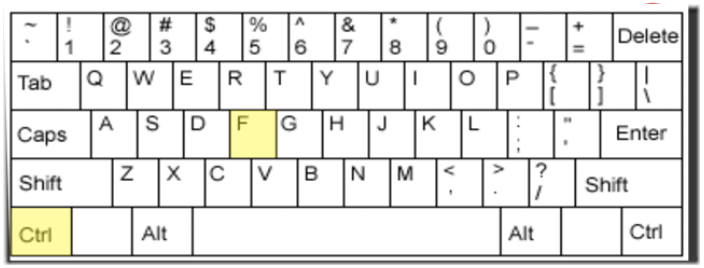


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

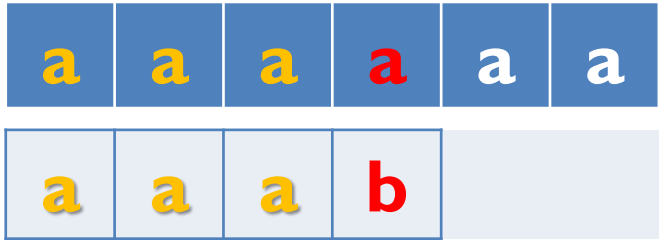
String Matching



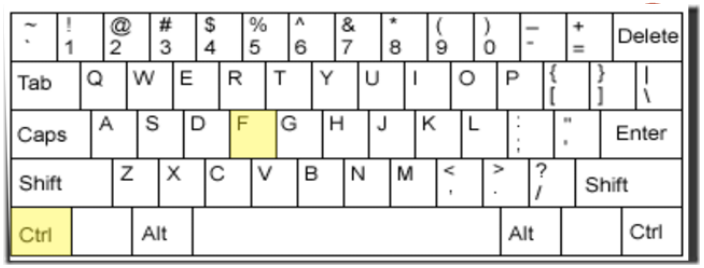
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



String Matching

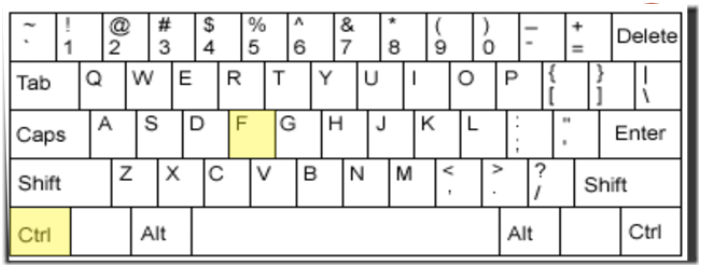


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

String Matching

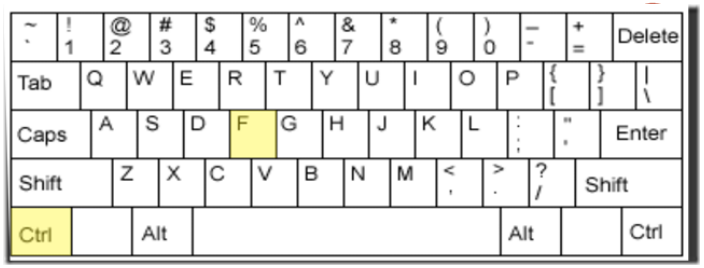


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

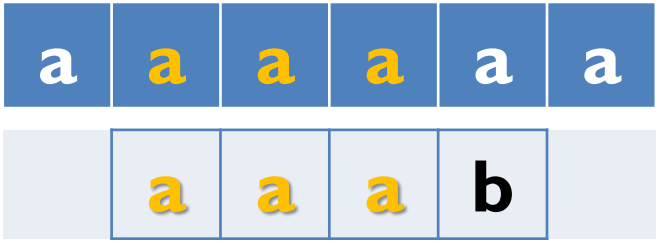
String Matching



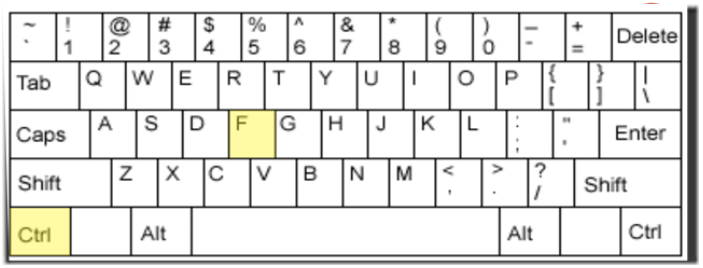
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



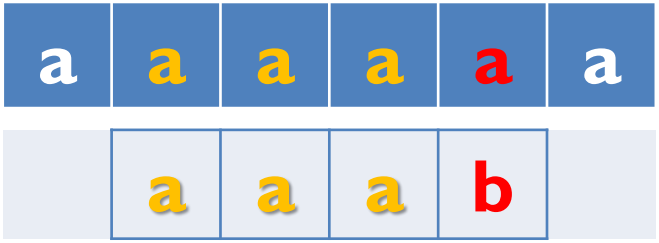
String Matching



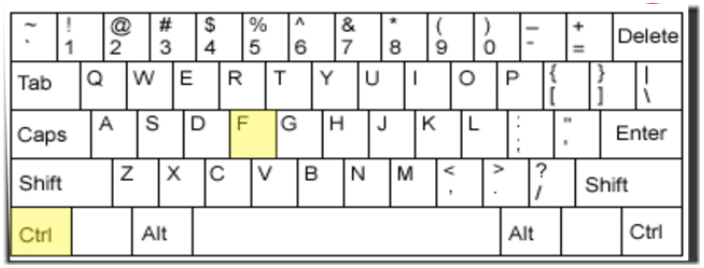
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



String Matching

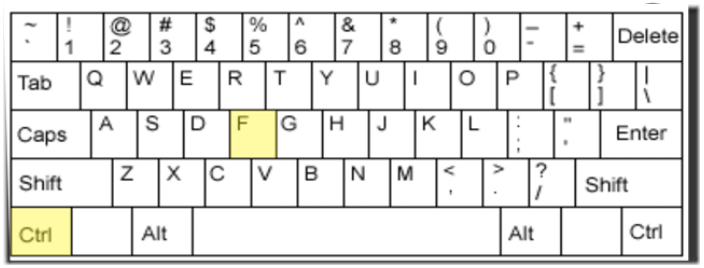


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

String Matching

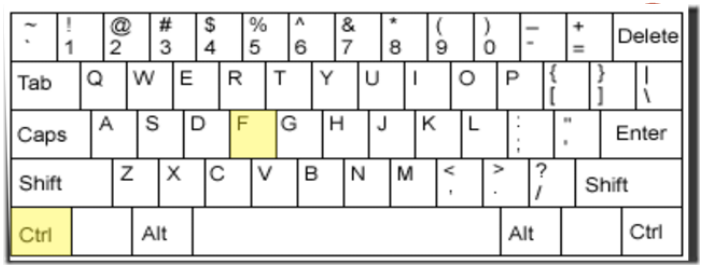


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

String Matching

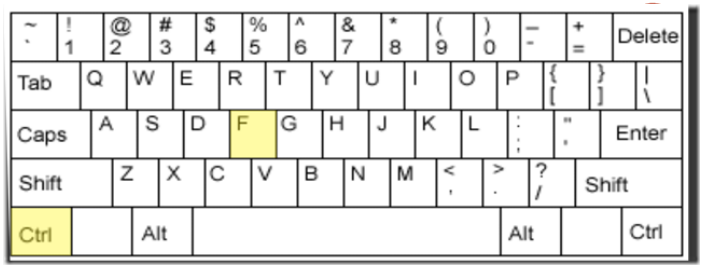


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
for(int i=0; i+lenB<=lenA; ++i){
    int mat=0;
    while(mat<lenB && A[i+mat]==B[mat]) ++mat;
    if(mat == lenB) print(i);
}
```

- 複雜度：~~$\mathcal{O}(|A|)$~~ $\mathcal{O}(|A||B|)$
- A = "aaaaaaa…aaa"
  B = "aaaaaaa…aab"

Outline

Hashing

↓

KMP

↓

Z Algorithm

↓

AC Automaton

Hashing

- 分類
  - 將字串分到有限的整數裡
  - 函數 $f: string \longmapsto \{0, 1, \ldots, \mathbb{Q} - 1\}$
- 要求
  - f 容易取得
  - 均勻
- 思考
  1. $f(A) \neq f(B) \Rightarrow A \neq B$
  2. $A \neq B \Rightarrow f(A) \neq f(B)$ → 不一定
  3. 分 n 類，碰撞機率 1/n

Hashing

- Rabin-Karp rolling hash function 定義
  - $f(A) = a_0 p^{n-1} + a_1 p^{n-2} + \cdots + a_{n-2} p + a_{n-1} \bmod q$
  - 類似： p 進位制，分成 q 類
  - p, q 取不同**質數** → 均勻
- 滾動
  1. $f(A) \equiv f(A[0, n-2])p + a_{n-1} \bmod q$
     → 計算 A 所有前綴的 hash value，$\mathcal{O}(|A|)$
  2. $f(A[i, j]) \equiv f(A[0, j]) - p^{j-i+1} f(A[0, i-1]) \bmod q$
     → 任何 A 子字串的 hash value，$\mathcal{O}(1)$
  3. 枚舉 A 長度為 |B| 的子字串，比較 hash value
     → $\mathcal{O}(N)$

Hashing

- A = "abcdefg"

a	b	c	d	e	f	g

$0p^2 + 1p^1 + 2p^0$

- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Hashing

- A = "abcdefg"

a	b	c	d	e	f	g
	$1p^2 + 2p^1 + 3p^0$					

- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Hashing

- A = "abcdefg"

a	b	c	d	e	f	g
		$2p^2 + 3p^1 + 4p^0$				

- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Matching!!

made by Jingfei

Hashing

- A = "abcdefg"

a	b	c	d	e	f	g
			$3p^2 + 4p^1 + 5p^0$			

- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Hashing

- A = "abcdefg"

a	b	c	d	e	f	g
				$4p^2 + 5p^1 + 6p^0$		

- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Hashing

- 回到剛剛
  - $A \neq B \Rightarrow f(A) \neq f(B)$ → 不一定
  - 相等時重新檢查一次？
  - A = "aaaaaaa…aaa"
    B = "aaaaaaa…aab"
- q 取大一點 (long long 質數)
  → 碰撞機率小
- ex. $q \in 10^{15} \Rightarrow probability: 10^{-15}$
- ex. 2147483647

Outline

Hashing

↓

KMP

↓

Z Algorithm

↓

AC Automaton

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”

a	a	b	a	a	c	?	?	…
a	a	b	a	a	b			

made by Jingfei

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”

a	a	b	a	a	c	?	?	…
a	a	b	a	a	b			

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”

a	a	b	a	a	c	?	?	…
	a	a	b	a	a	b		

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”

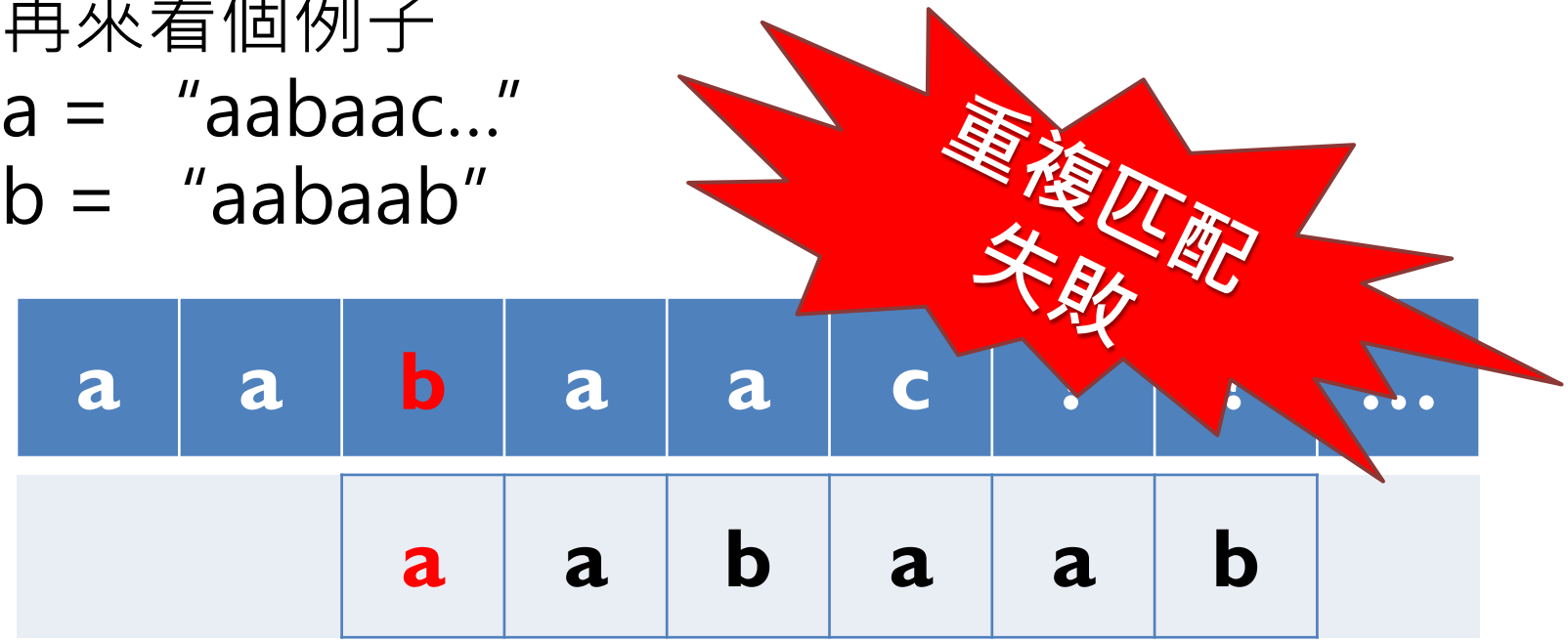
a	a	b	a	a	c	?	?	…
		a	a	b	a	a	b	

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”



KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”

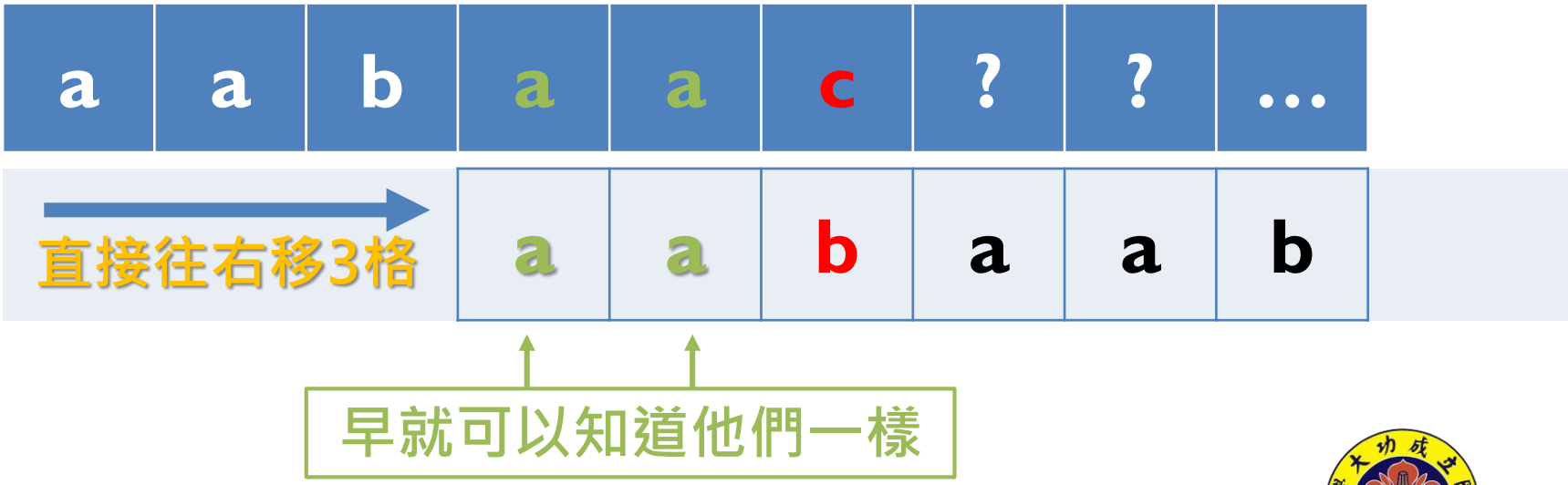
a	a	b	a	a	c	?	?	…
a	a	b	a	a	b			

KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

  b = “aabaab”



KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子

  a = “aabaac…”

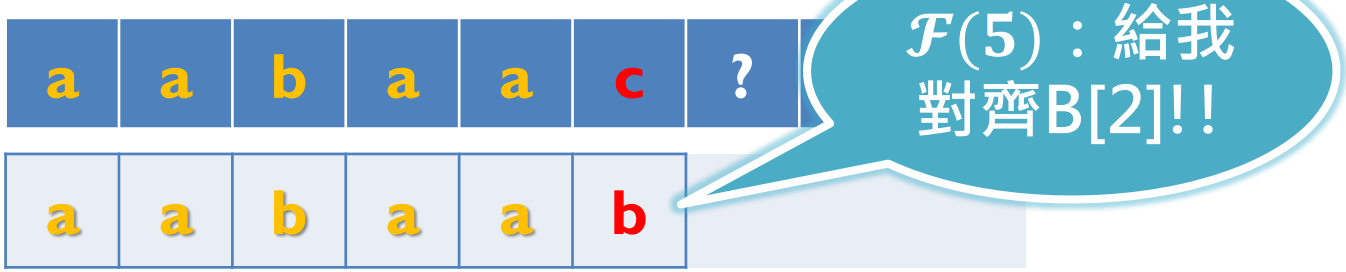
  b = “aabaab”

a	a	b	a	a	c	?	?	…
直接往右移3格			a	a	b	a	a	b

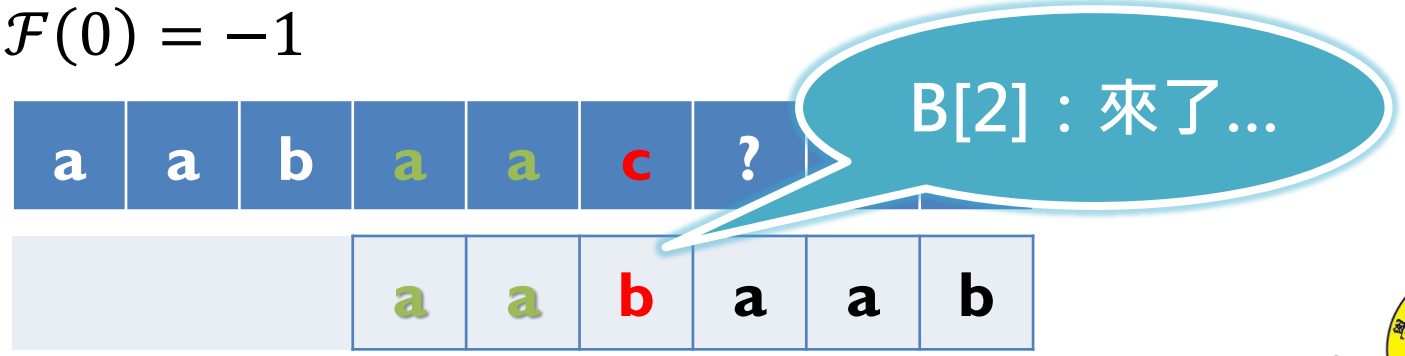
問題出在 B 有重複子字串

made by Jingfei

KMP

- Knuth-Morris-Pratt algorithm
- 怎麼處理 B ？
- 定義 **Fail function** (失敗函數)
  - 期望：$\mathcal{F}(i)$ 能知道匹配失敗時，B 要對齊哪裡繼續匹配
  - $\mathcal{F}_B(i) = \begin{cases} \max\{k\colon P_B(k) = B[0,k] = B[i-k,i]\}, if\ i \neq 0\ and\ at\ least\ a\ \boldsymbol{k}\ exists \\ -1,\ else \end{cases}$
  - $\mathcal{F}(0) = -1$
  - 

KMP

- Knuth-Morris-Pratt algorithm
- 怎麼處理 B ？
- 定義 **Fail function** (失敗函數)
  - 期望：$\mathcal{F}(i)$ 能知道匹配失敗時，B 要對齊哪裡繼續匹配
  - $\mathcal{F}_B(i) = \begin{cases} \max\{k\colon P_B(k) = B[0,k] = B[i-k,i]\}, if\ i \neq 0\ and\ at\ least\ a\ \boldsymbol{k}\ exists \\ -1,\ else \end{cases}$
  - $\mathcal{F}(0) = -1$
  - 

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1								

```
init:
    pi[0]=-1
    cur_pos=-1
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1							

```
B[cur_pos+1]!=B[i]
pi[i]=cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1						

```
B[cur_pos+1]!=B[i]
pi[i]=cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0					

```
B[cur_pos+1]==B[i]
pi[i]=++cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1				

```
B[cur_pos+1]==B[i]
pi[i]=++cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1	2			

```
B[cur_pos+1]==B[i]
pi[i]=++cur_pos
```

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1	2	3		

```
B[cur_pos+1]==B[i]
pi[i]=++cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1	2	3	4	

```
B[cur_pos+1]==B[i]
pi[i]=++cur_pos
```

made by Jingfei

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1	2	3	4	1

```
B[cur_pos+1]!=B[i]
pi[i]=pi[cur_pos]
```

KMP

- Fail function example

i	0	1	2	3	4	5	6	7	8
B	a	b	z	a	b	z	a	b	c
pi	-1	-1	-1	0	1	2	3	4	-1

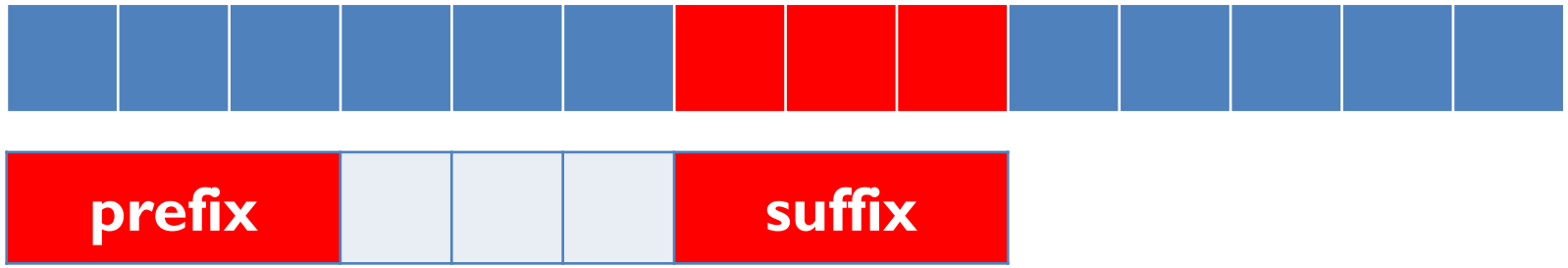
```
B[cur_pos+1]!=B[i]
pi[i]=pi[cur_pos]
```

KMP

- Fail function

```
void fail(string B, int *pi){
    int len = strlen(B);
    pi[0] = -1;
    for(int i=1, cur_pos=-1; i<len; ++i){
        while(cur_pos>=0 && B[i]!=B[cur_pos+1])
            cur_pos=pi[cur_pos];
        if(B[i]==B[cur_pos+1]) ++cur_pos;
        pi[i]=cur_pos;
    }
}
```

KMP

- Matching
- Fail function: 找出各後綴與前綴一樣的最大值
- 如果後綴 = 前綴 → 可直接位移
- 

KMP

- Matching
- Fail function: 找出各後綴與前綴一樣的最大值
- 如果後綴 = 前綴 → 可直接位移
- 

KMP

- Matching

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

init:

cur_pos = -1

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]!=B[cur_pos+1]

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

			i											
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos

KMP

- Matching

			i										
A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos

KMP

- Matching

					i									
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos

KMP

- Matching

								i						
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

								cur_pos		
	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

									i					
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

									cur_pos	
	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

										i				
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]!=B[cur_pos+1]

cur_pos=pi[cur_pos]

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

										i				
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

											i		
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

								cur_pos		
	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

													i	
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]==B[cur_pos+1]
++cur_pos

									cur_pos	
	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

													i	
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

A[i]==B[cur_pos+1]
++cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

made by Jingfei

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]

++cur_pos

cur_pos+1==len(B)

Match!!!

KMP

- Matching

													i	
A		x	a	b	z	a	b	z	a	b	z	a	b	c	d

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

A[i]==B[cur_pos+1]
++cur_pos
cur_pos+1==len(B)
cur_pos=pi[cur_pos]

KMP

- Matching

i

A	x	a	b	z	a	b	z	a	b	z	a	b	c	d

A[i]!=B[cur_pos+1]

cur_pos

	-1	0	1	2	3	4	5	6	7	8
B		a	b	z	a	b	z	a	b	c
pi		-1	-1	-1	0	1	2	3	4	-1

KMP

- Matching

```
1  void fail(string A, string B, int *pi){
2      int lenA = A.length();
3      int lenB = B.length();
4      for(int i=0, cur_pos=-1; i<lenA; ++i){
5          while(cur_pos>=0 && A[i]!=B[cur_pos+1])
6              cur_pos=pi[cur_pos];
7          if(A[i]==B[cur_pos+1]) ++cur_pos;
8          if(cur_pos+1==lenB){
9              /* Match!!! */
10             cur_pos=pi[cur_pos];
11         }
12     }
13 }
```

KMP

- Fail function + Matching
- Complexity
  - 關鍵: while-loop
  - cur_pos 每次只會 +1 或往前
  - 均攤後 $\mathcal{O}(|\mathrm{A}| + |\mathrm{B}|)$

Outline

Hashing

↓

KMP

↓

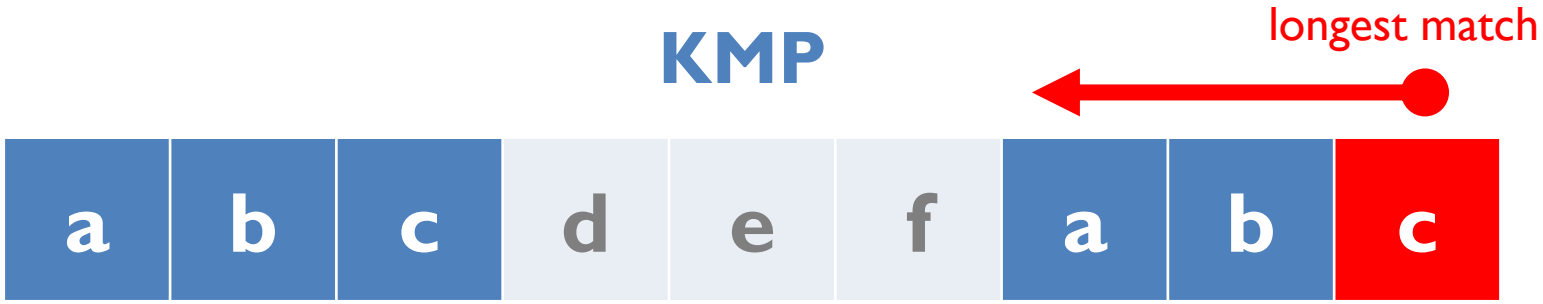
Z Algorithm

↓

AC Automaton

Z Algorithm

- vs KMP
  - S= "abcdefabc"



made by Jingfei

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z									

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9								

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0							

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0						

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0	1					

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0	1	4				

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0	1	4	0			

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0	1	4	0	0		

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	a	b	z	a	a	b	z	a	b
z	9	0	0	1	4	0	0	2	

Z Algorithm

- 定義
  - $Z_A(i) = \begin{cases} 0, if\ i = 0\ or\ A[i] \neq A[0] \\ \max\{k: A[0, k-1] = A[i, i+k-1]\}, else \end{cases}$
  - 由 $A[i]$ 開始的字串，可以和 A 自己匹配多長
  - ex.

i	0	1	2	3	4	5	6	7	8
A	**a**	**b**	**z**	**a**	**a**	**b**	**z**	**a**	**b**
z	9	0	0	1	4	0	0	2	0

Z Algorithm

- 假設一個字串 A 中，$Z(i) = z$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	**a**	**b**	**a**	**b**	**a**	**b**	**e**	**a**	**b**	**a**	**b**	**a**	**b**	**f**
z	14	0	4	0	2	0	0	**6**	0	4	0	2	0	0

Z Algorithm

- 假設一個字串 A 中，$Z(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

Z Algorithm

- 假設一個字串 A 中，$Z(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

Z Algorithm

- 假設一個字串 A 中，$\mathcal{Z}(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 令 $L = i,\ R = i+z-1,\ L \le j \le R,\ j' = j - L$

								L					R	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 假設一個字串 A 中，$\mathcal{Z}(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 令 $L = i,\ R = i+z-1,\ L \le j \le R,\ j' = j - L$
  - $case\ \times 3$

								L					R	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 假設一個字串 A 中，$\mathcal{Z}(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 令 $L = i,\ R = i+z-1,\ L \le j \le R,\ j' = j - L$
  - $\boldsymbol{case\ 1.\quad j' + Z(j') < z \implies Z(j) = Z(j')}$

L R

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 假設一個字串 A 中，$\mathcal{Z}(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 令 $L = i,\ R = i+z-1,\ L \le j \le R,\ j' = j - L$
  - $\boldsymbol{case\ 2.\quad j' + Z(j') > z \implies Z(j) = R - j + 1}$ (j 到 R 的長度)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
								L					R	
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 假設一個字串 A 中，$\mathcal{Z}(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 令 $L = i,\ R = i + z - 1,\ L \le j \le R,\ j' = j - L$
  - $\boldsymbol{case\ 3.}\quad \boldsymbol{j' + Z(j') = z \implies Z(j) \ge Z(j')}$ (剛好在邊界)

L R

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 假設一個字串 A 中，$Z(i) = z$
  - $A[k] = A[i+k],\ if\ 0 \le k < z$
  - $A[z] \ne A[i+z]$
  - 更新 $L = j,\ R = j + Z(j) - 1,\ L \le j \le R,\ j' = j - L$
  - $\boldsymbol{case\ 3.\quad j' + Z(j') = z \implies Z(j) \ge Z(j')}$

										L			R	
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	a	b	a	b	a	b	e	a	b	a	b	a	b	f
z	14	0	4	0	2	0	0	6	0	4	0	2	0	0

j' j

made by Jingfei

Z Algorithm

- 只會前進不會後退 → $\mathcal{O}(N)$

```
L = R = 0;
for(int i=1; i<len; i++){
    if(i>R) Z[i]=0; //Case 0
    else{
        int ip = i - L;
        if(ip+Z[ip] < Z[L]) Z[i]=Z[ip]; //Case 1
        else Z[i]=R-i+1; //Case 2, 3
    }
    while(i+Z[i] < len && A[i+Z[i]]==A[Z[i]])
        Z[i]++;
    if(i+Z[i]-1 > R){
        L=i;
        R=i+Z[i]-1;
    }
}
```

Z Algorithm

- Matching ?
- 把兩個字串接起來， B 在前，A 在後，
  中間用一個沒有出現過的字元連接
- 如 A="abaab",B="aab"
  令 S="aab$abaab"
  → 發現 B 是否能在 A 的某個位置被匹配，
  只要看那個位置的 Z value 是否等於 B 的長度
- 最長回文子字串

Outline

Hashing

↓

KMP

↓

Z Algorithm

↓

AC Automaton

AC Automaton

- KMP 複雜度 $\mathcal{O}(|\mathrm{A}| + |\mathrm{B}|)$
- 多字串匹配
  1. 一個字串 B 匹配很多字串 A_i

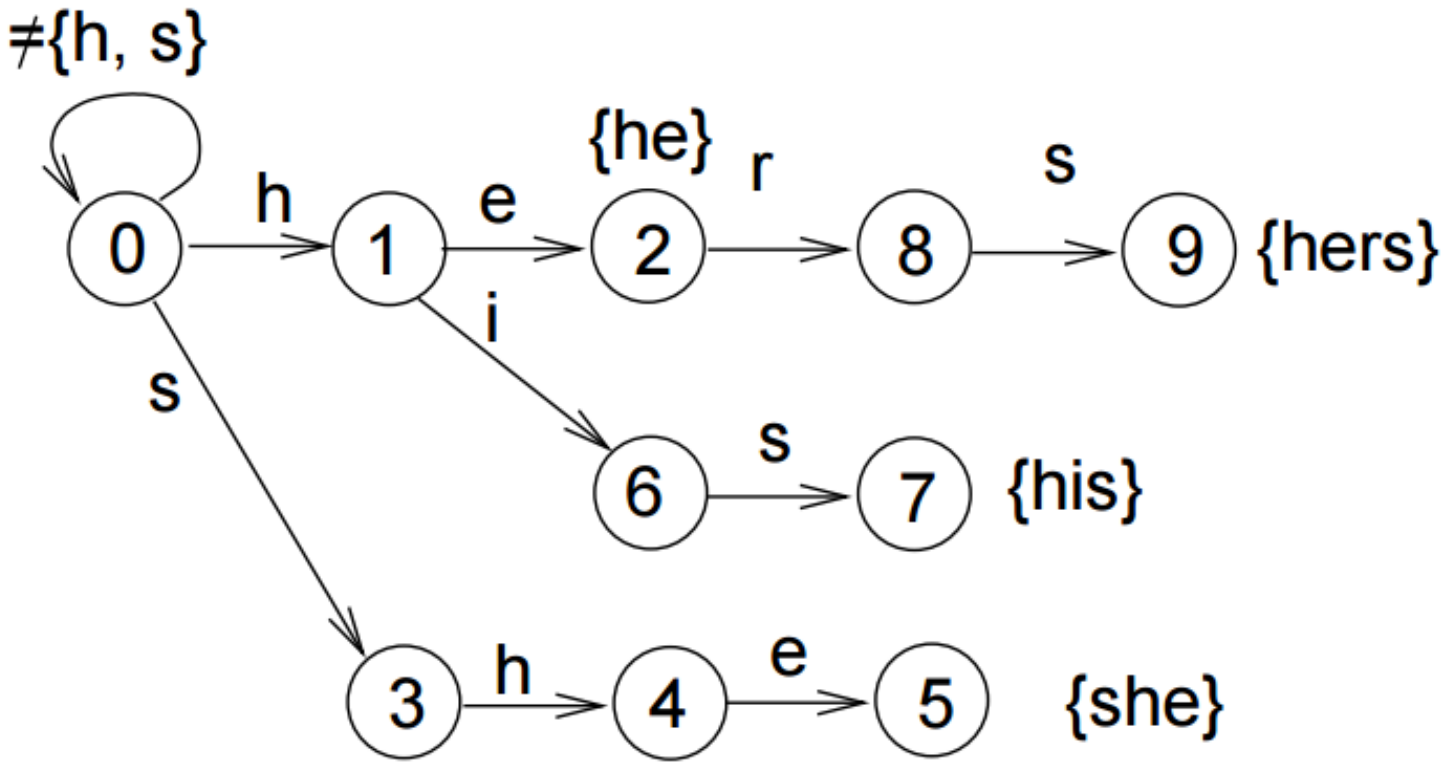
     $\rightarrow \mathcal{O}(\sum|\mathrm{A}_i| + |\mathrm{B}|)$

     $\rightarrow$ 線性
  2. 很多字串 B_i 匹配一個字串 A

     $\rightarrow \mathcal{O}(n|A| + \sum|B_i|)$

     $\rightarrow$ 弱弱的
- Trie：儲存多個字串
- AC 自動機 = KMP + Trie

AC Automaton

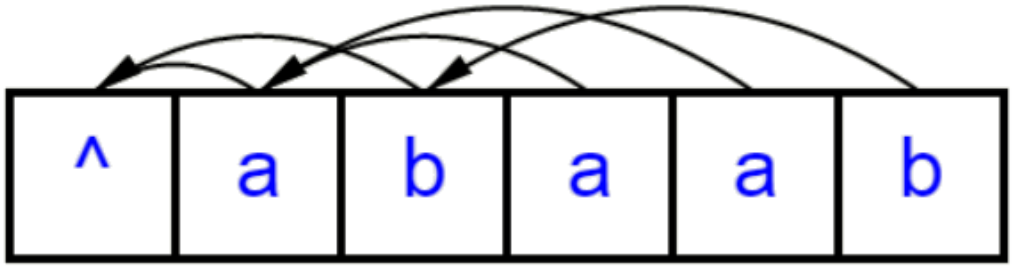


- Trie：儲存多個字串
- AC 自動機 = KMP + Trie

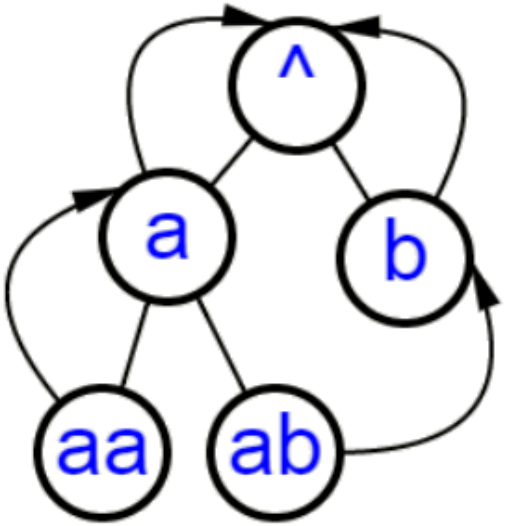
AC Automaton

- 比較 Fail function (圖)

KMP:



AC自動機:



AC Automaton

- 比較 Fail function (定義)
- KMP
  - $\mathcal{F}_B(i) = \begin{cases} \max\{k\colon P_B(k) = B[0,k] = B[i-k,i]\}, if\ i \neq 0\ and\ at\ least\ a\ \boldsymbol{k}\ exists \\ -1,\ else \end{cases}$
  - A[0,k] 是 A[0,i] 的前綴
- AC Automaton
  - $\mathfrak{F}_B(v) = \begin{cases} u,\ if\ \mathrm{B}_T(u) \text{ 是 } \mathrm{B}_T(v) \text{ 的前綴且 } |\mathrm{S_T}(u)| \text{ 最大} \\ v_0,\ else \end{cases}$
  - $\mathrm{B}_T(u)$ 是 $\mathrm{B}_T(v)$ 的前綴

AC Automaton

- 比較 Fail function (匹配失敗)
- KMP
  - 沿著 $\mathcal{F}(i)$, $\mathcal{F}^2(i)$, … 嘗試，直到 $\mathcal{F}^t(i) = -1$
- AC Automaton
  - 沿著 $\mathfrak{F}(v)$, $\mathfrak{F}^2(v)$, … 嘗試，直到 $\mathfrak{F}^t(v) = v_0$ (v_0: root)

AC Automaton

- 比較 Fail function (構造)
- KMP
  - 利用 $\mathcal{F}(i-1)$ 求出 $\mathcal{F}(i)$
- AC Automaton
  - 利用 $\mathfrak{F}(u)$ 求出 $\mathfrak{F}(v)$，u 為 v 的父節點
  - use BFS

AC Automaton

- $\mathcal{O}(\ |A| + \sum|B|\ )$
-

```
root->fail = NULL;
queue< Node* > que;
que.push_back(root);
while ( !que.empty() ) {
    Node *fa = que.front(); que.pop_front();

    for (auto it = fa->child.begin();
            it != fa->child.end(); it++) {
        Node *cur = it->second, *ptr = fa->fail;
        while ( ptr && !ptr->child.count(it->first) )
            ptr = ptr->fail;

        cur->fail = ptr ? ptr->child[it->first] : root;
        que.push(cur);
    }
}
```

made by Jingfei

Example

- POJ 3461
- UVA 455

Reference

- 歷屆PPT…… (electron, free999, louis6340, …)
- 2015 IOI camp 字串處理
  http://ioicamp.csie.org/content
  http://bobogei81123.github.io/ioi-lecture