Jheng-Huang Hong

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

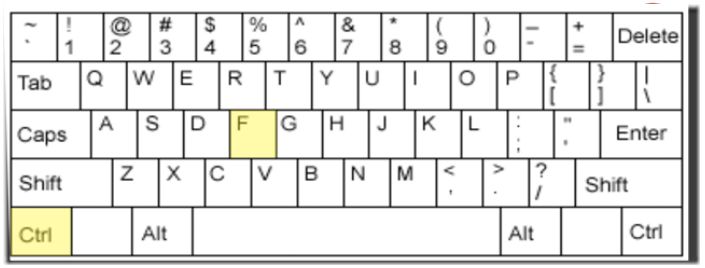
String Basic

- 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”, …

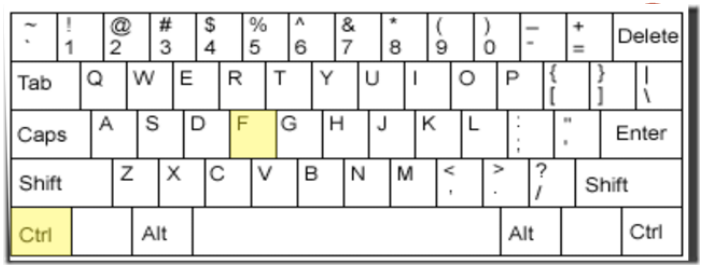
String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 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
-

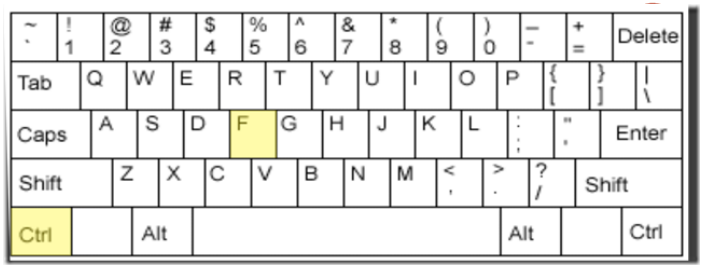
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

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

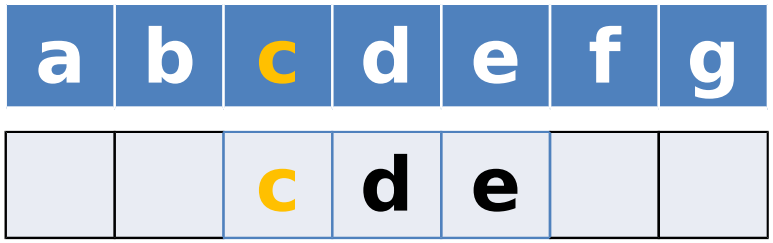
a	b	c	d	e	f	g
	c	d	e			

made by Jingfei

String Matching

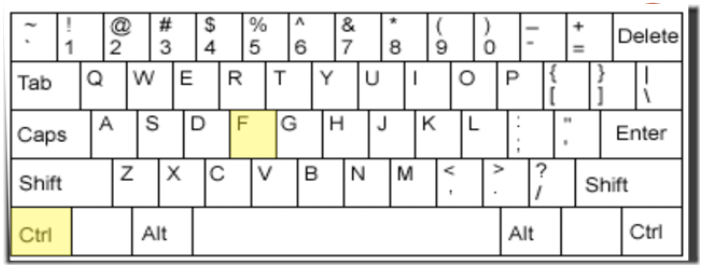


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

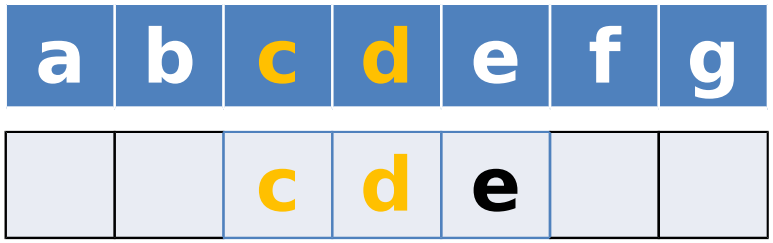


made by Jingfei

String Matching

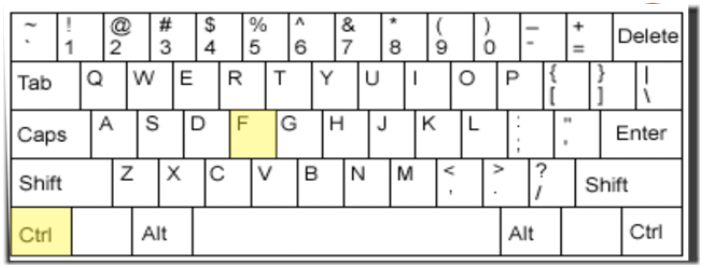


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”



made by Jingfei

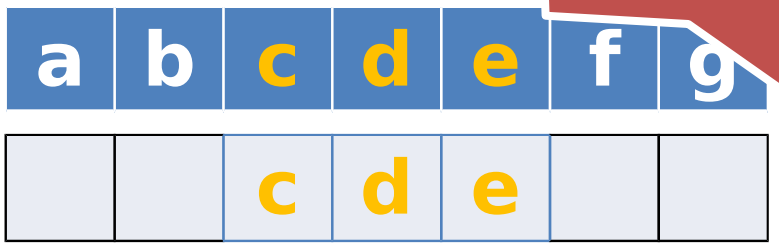
String Matching



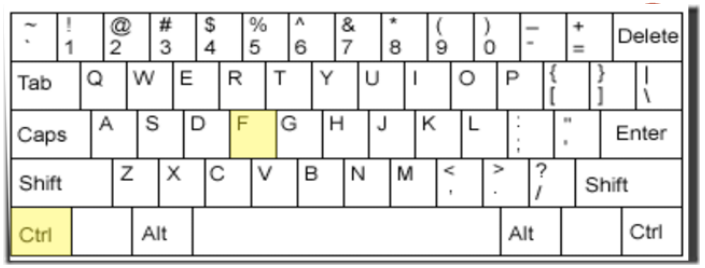
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

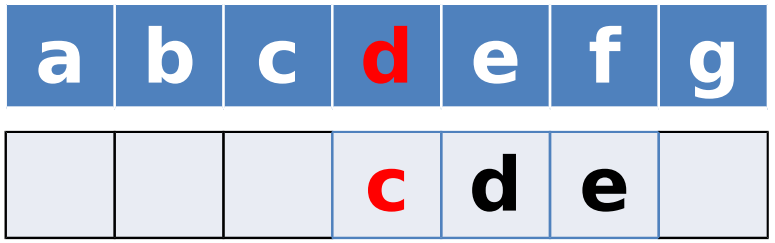
- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”



String Matching

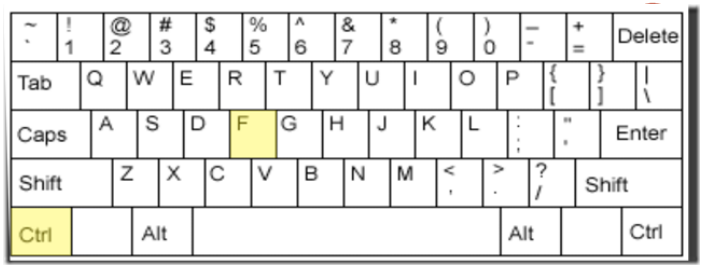


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”



made by Jingfei

String Matching

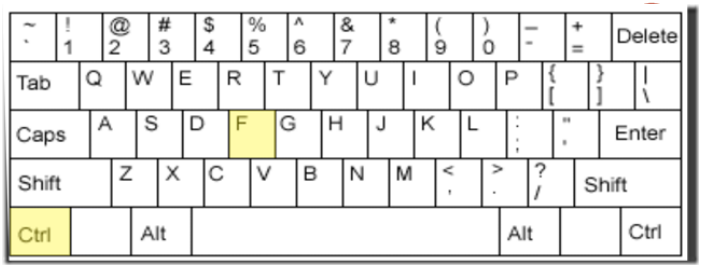


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “abcdefg”
  B = “cde”

a	b	c	d	e	f	g
				c	d	e

made by Jingfei

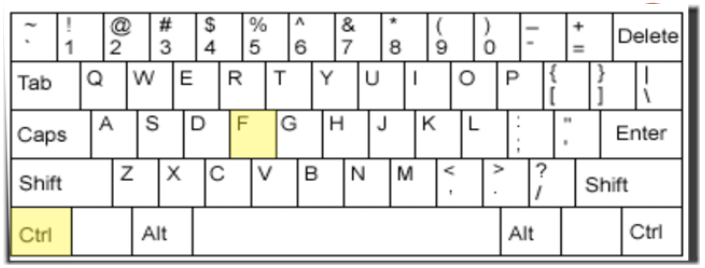
String Matching



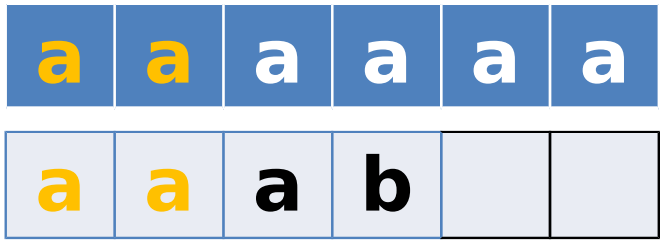
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

a	a	a	a	a	a
a	a	a	b		

String Matching

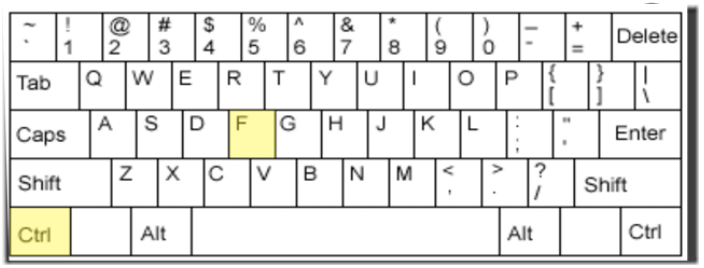


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



made by Jingfei

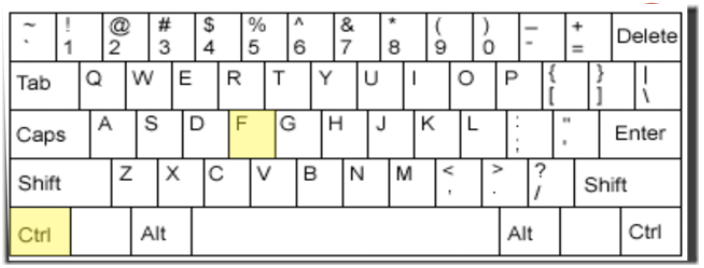
String Matching



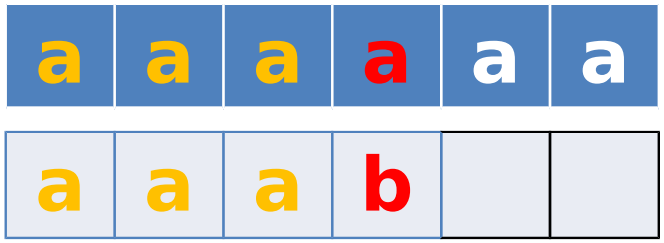
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

made by Jingfei

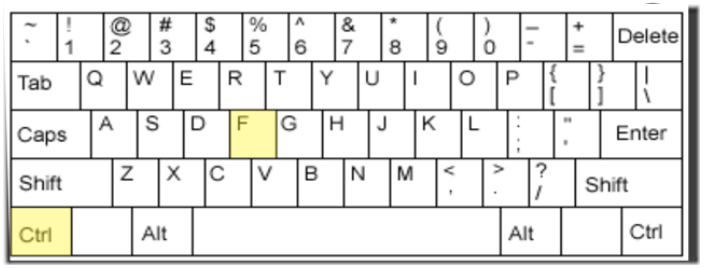
String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



String Matching



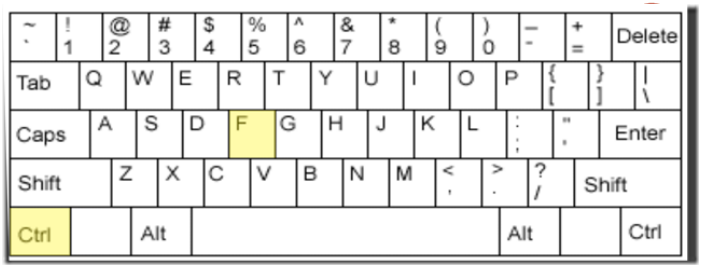
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

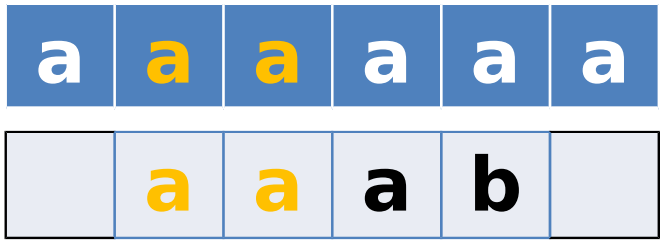
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

made by Jingfei

String Matching

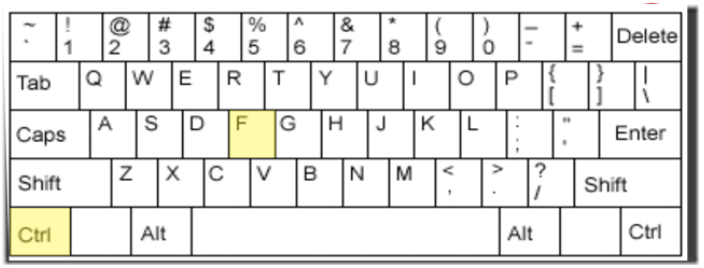


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



made by Jingfei

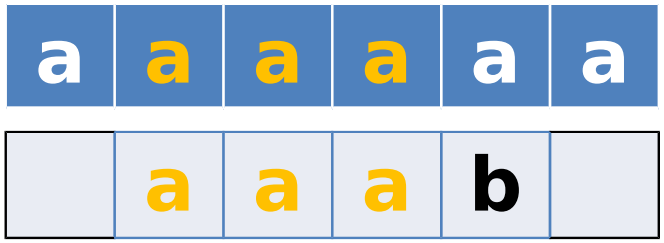
String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

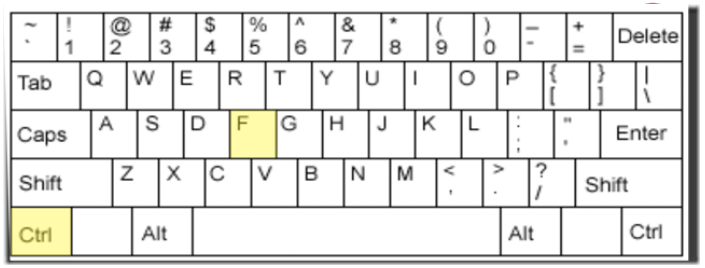
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

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



made by Jingfei

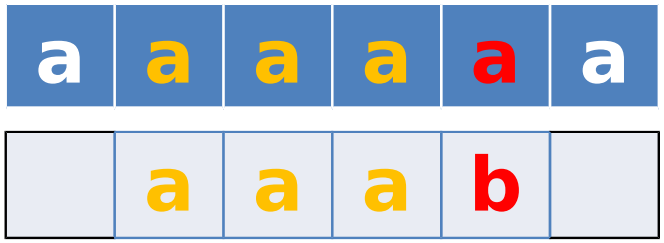
String Matching



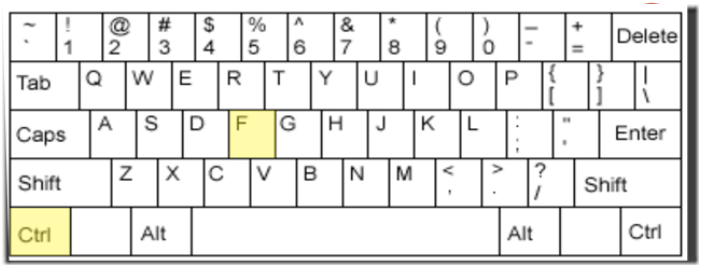
- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

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



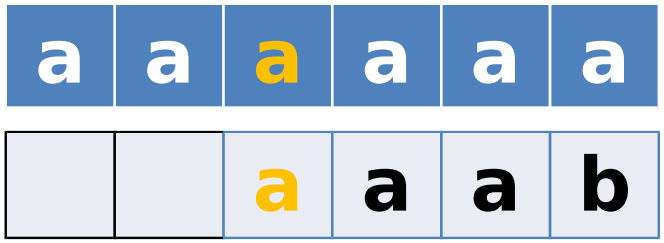
String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

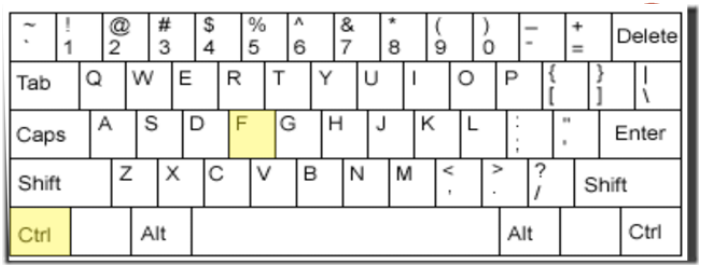
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

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

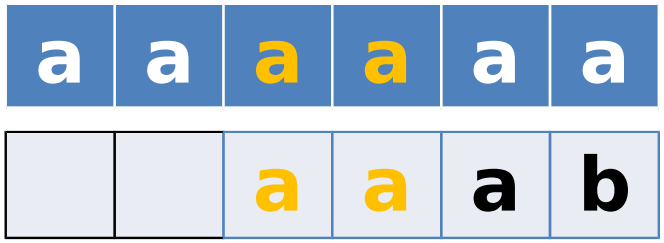


made by Jingfei

String Matching

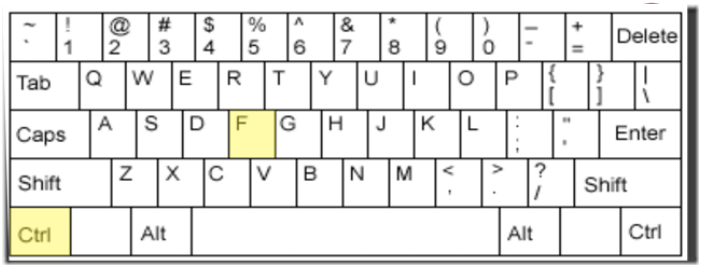


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”

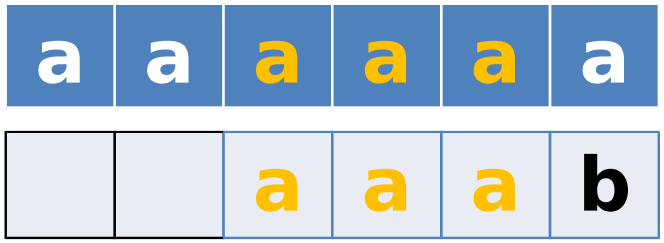


made by Jingfei

String Matching

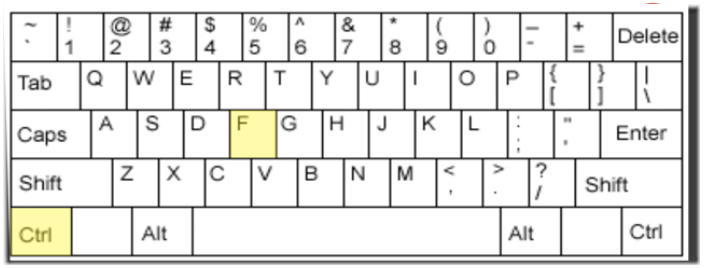


- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```
- 複雜度：$\mathcal{O}(|A|)$
- A = “aaaaaaa…aaa”
  B = “aaaaaaa…aab”



made by Jingfei

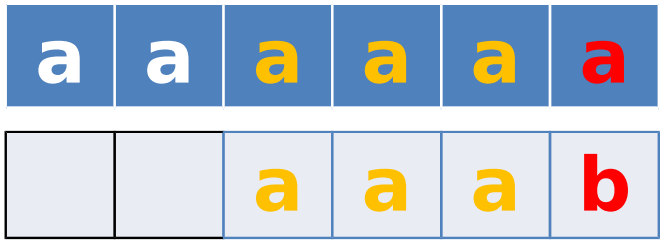
String Matching



- 給兩個字串 A, B 找出所有 B 出現在 A 中的位置
-

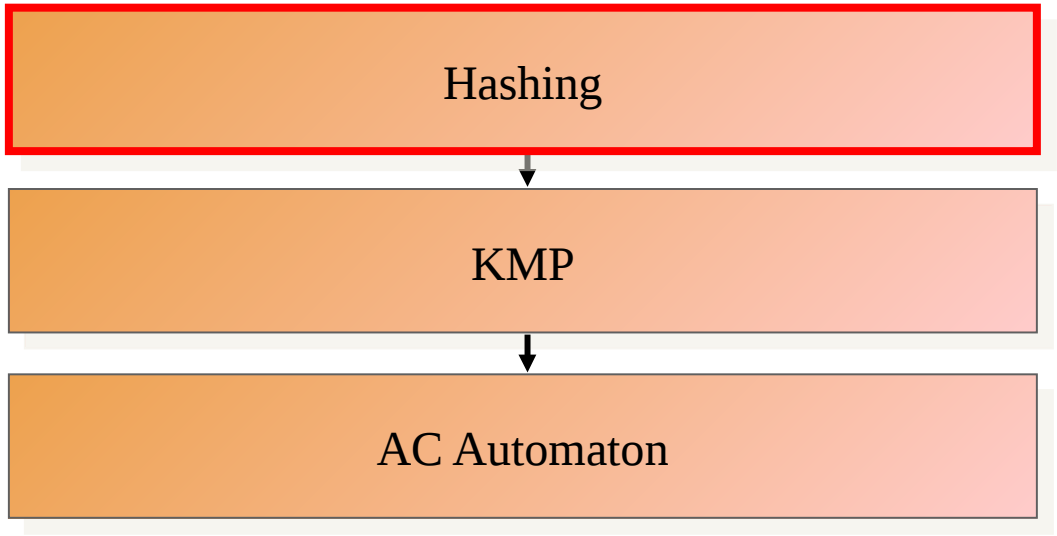
```
1 for(int i=0; i+lenB<=lenA; ++i){
2       int mat=0;
3       while(mat<lenB && A[i+mat]==B[mat]) ++mat;
4       if(mat == lenB) print(i);
5 }
```

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



made by Jingfei

Outline



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
  - 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

made by Jingfei

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 hashvalue，$\mathcal{O}(|A|)$
  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 hashvalue，$\mathcal{O}(1)$
  3. 枚舉 A 長度為 |B| 的子字串，比較 hashvalue
     ⇒ $\mathcal{O}(N)$

made by Jingfei

Hashing

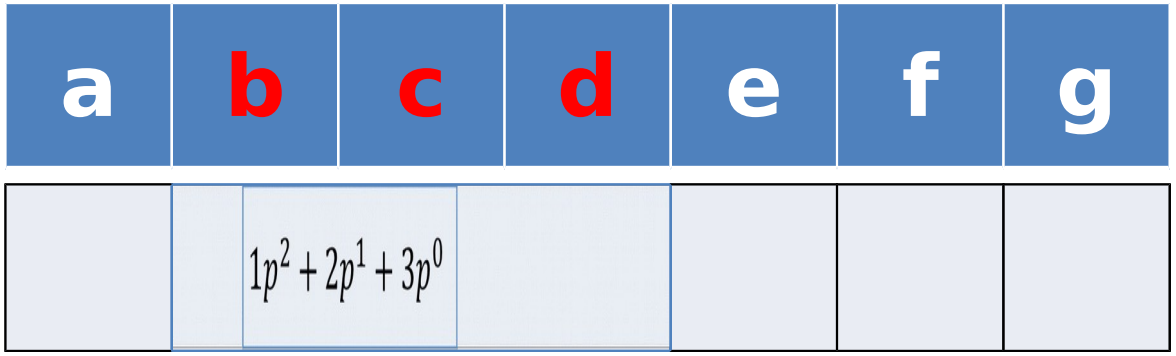
- A = "abcdefg"

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

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

Hashing

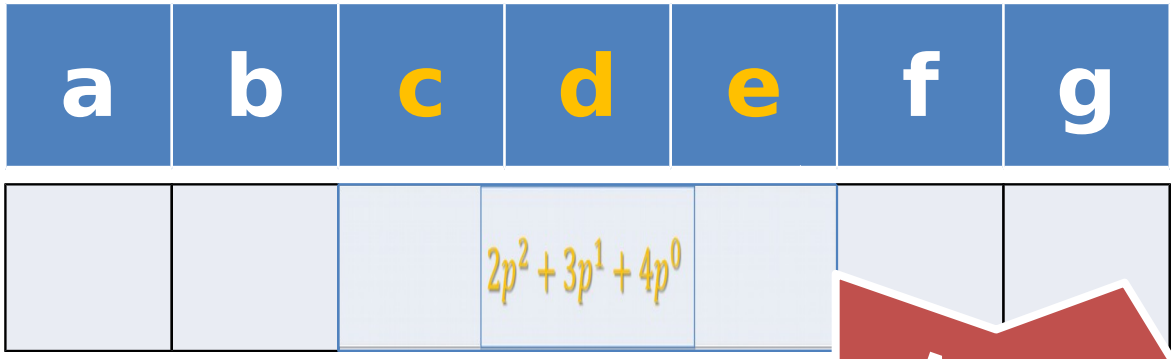
- A = "abcdefg"



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

Hashing

- A = "abcdefg"



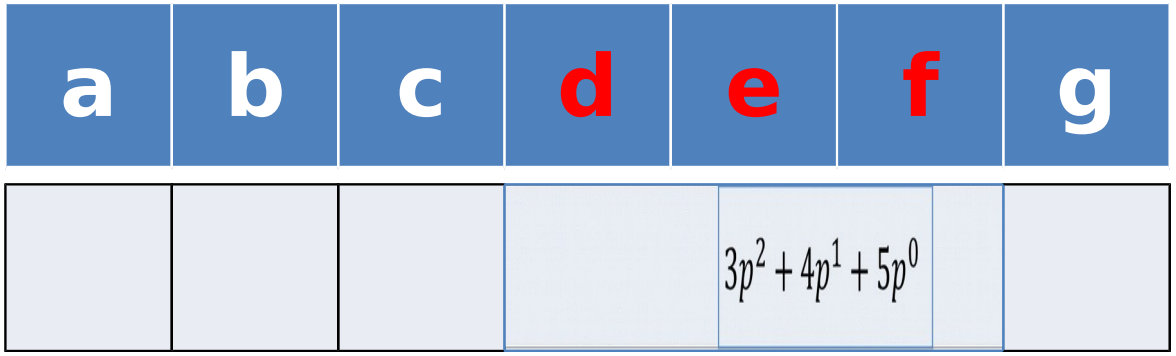
- B = "cde" = $2p^2 + 3p^1 + 4p^0$

Matching!!

made by Jingfei

Hashing

- A = "abcdefg"



- 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

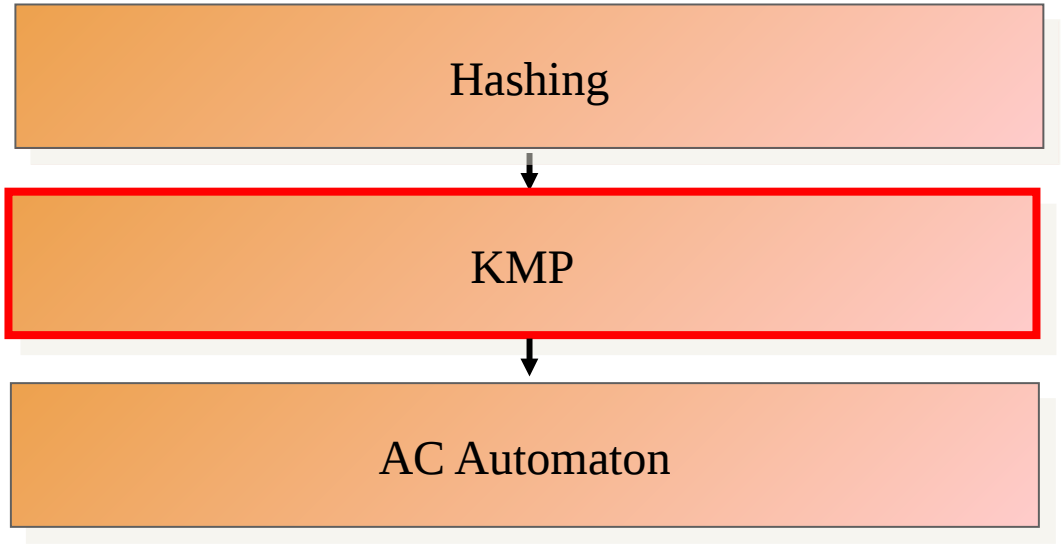
- 回到剛剛
  - $A \neq B \Rightarrow f(A) \neq f(B)$ → 不一定
  - 相等時重新檢查一次？
  - A = “aaaaaaa…aaa”
    B = “aaaaaaa…aab”
- q 取大一點 (long long 質數)
  → 碰撞機率小
- ex. $q \in 10^{15} \Rightarrow probability: 10^{-15}$
- ex. 2147483647

Hashing – 參考

```
1  #define MAXN 1000000
2  #define prime_mod 1073676287
3  /*prime_mod 必須要是質數*/
4  typedef long long T;
5  char s[MAXN+5];
6  T h[MAXN+5];/*hash陣列*/
7  T h_base[MAXN+5];/*h_base[n]=(prime^n)%prime_mod*/
8  inline void hash_init(int len,T prime=0xdefaced){
9          h_base[0]=1;
10         for(int i=1;i<=len;++i){
11                 h[i]=(h[i-1]*prime+s[i-1])%prime_mod;
12                 h_base[i]=(h_base[i-1]*prime)%prime_mod;
13         }
14 }
15 inline T get_hash(int l,int r){/*閉區間寫法，設編號為0 ~ len-1*/
16         return (h[r+1]-(h[l]*h_base[r-l+1])%prime_mod+prime_mod)%prime_mod;
17 }
```

Source: 日月卦長的模板庫
→ [Rabin-Karp rolling hash] Rabin-Karp 字串hash演算法

Outline



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**			

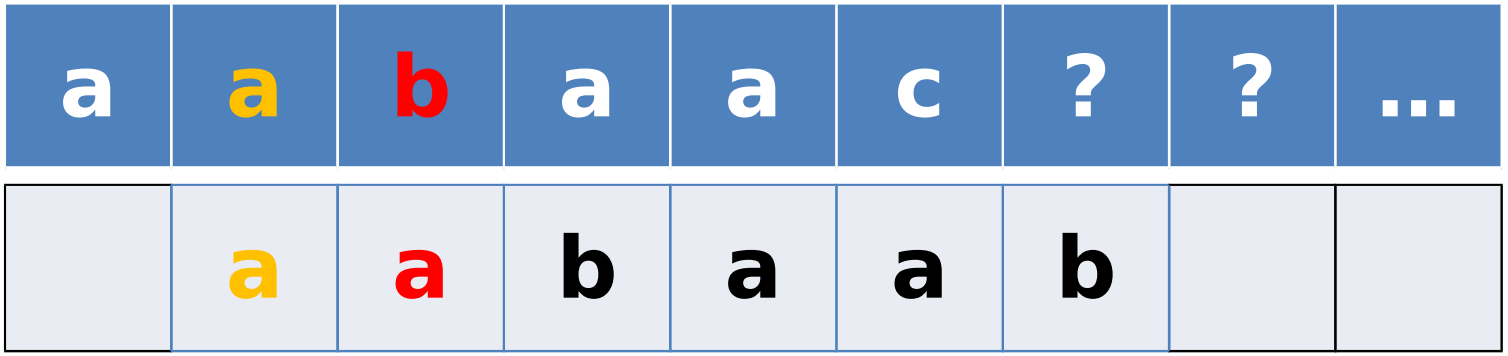
KMP

- 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			

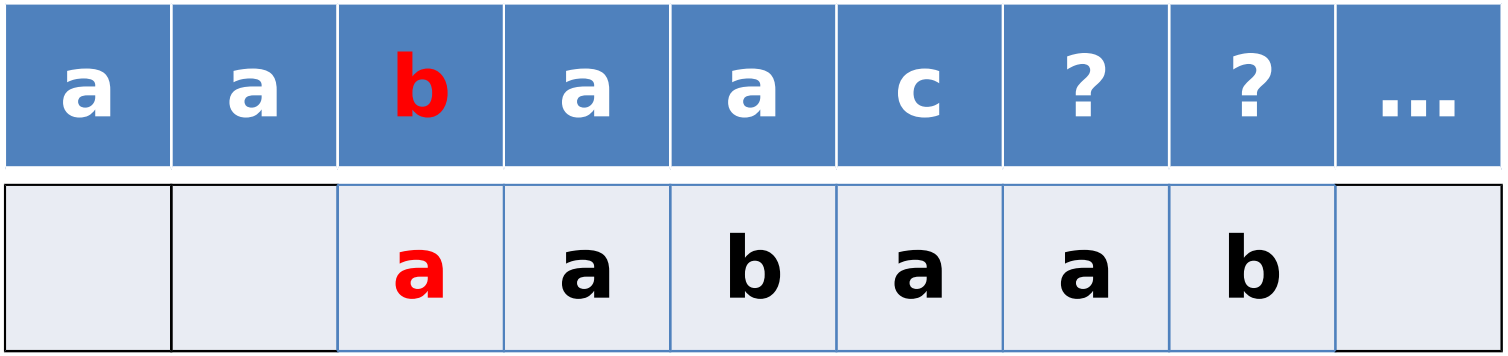
KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子
  a = “aabaac…”
  b = “aabaab”



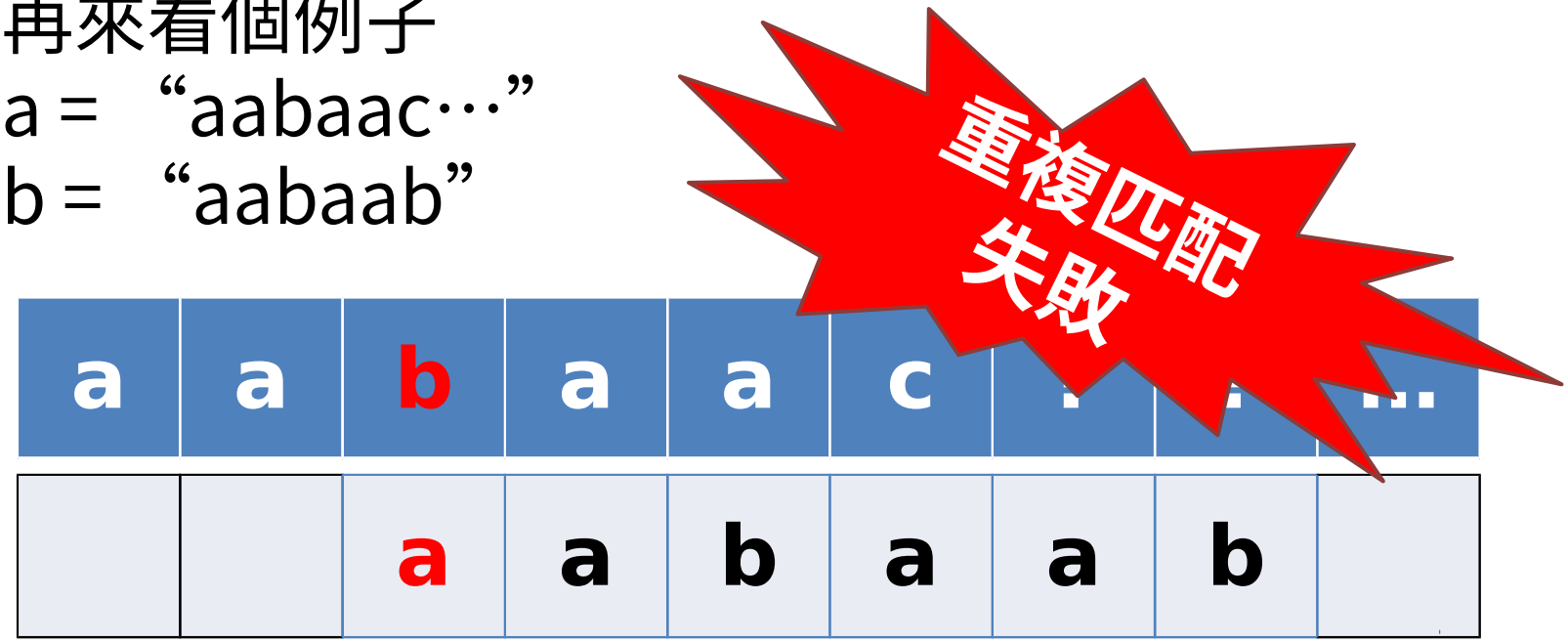
KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子
  a = “aabaac…”
  b = “aabaab”



KMP

- Knuth-Morris-Pratt algorithm
- 再來看個例子
  a = “aabaac…”
  b = “aabaab”



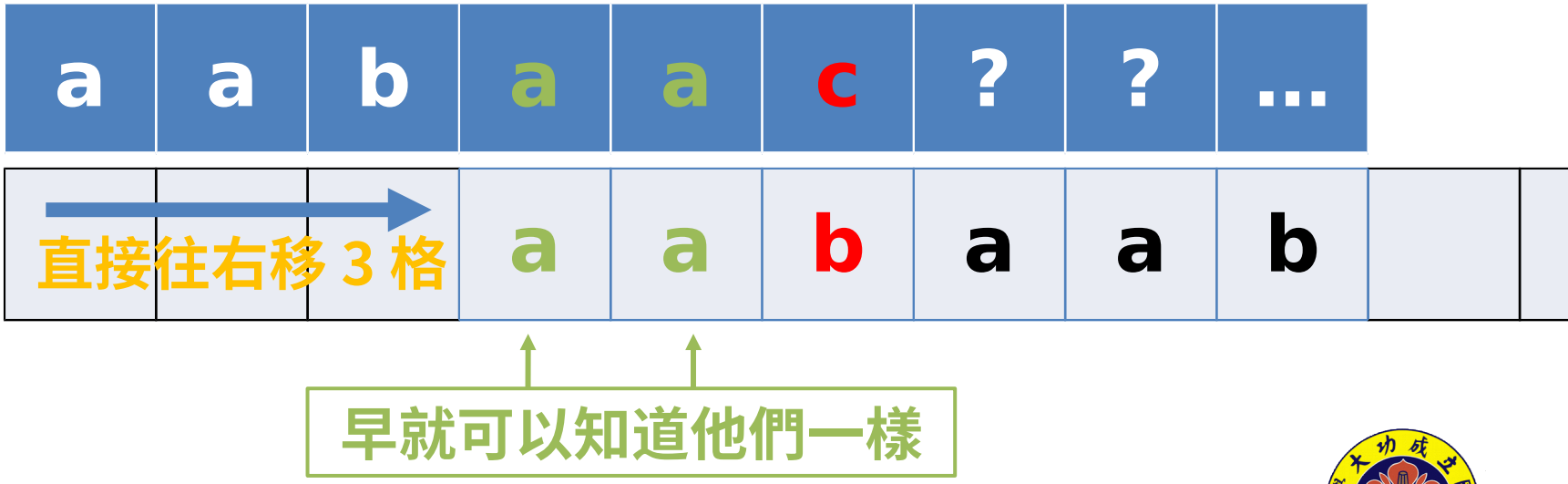
KMP

- 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			

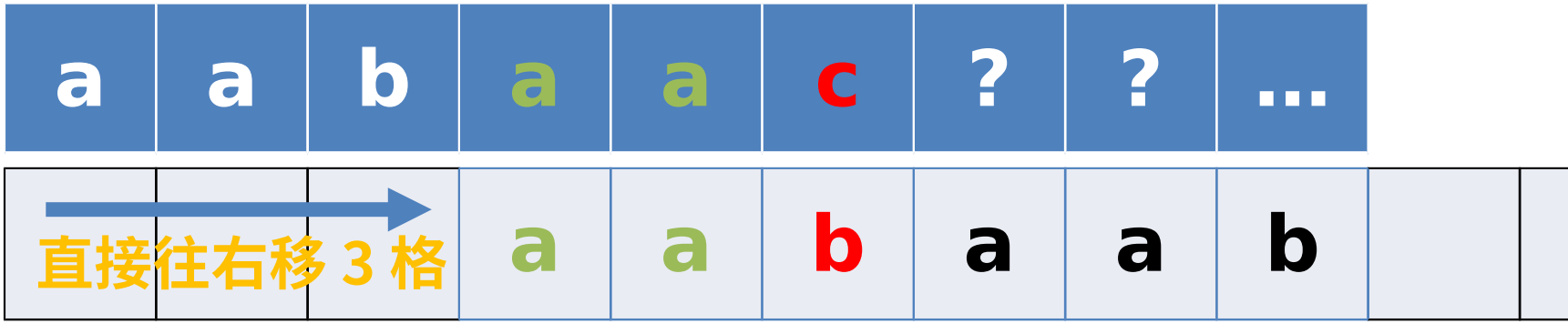
KMP

- 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”

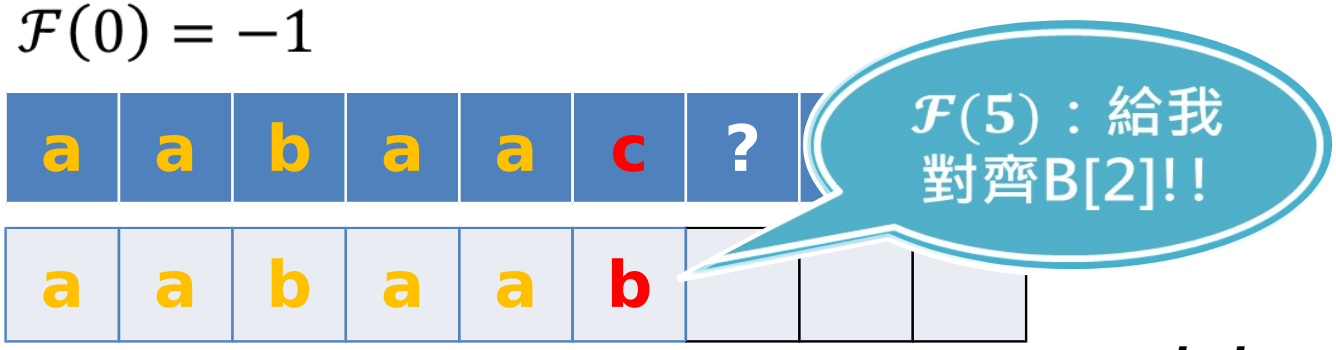


問題出在 B 有重複子字串

KMP

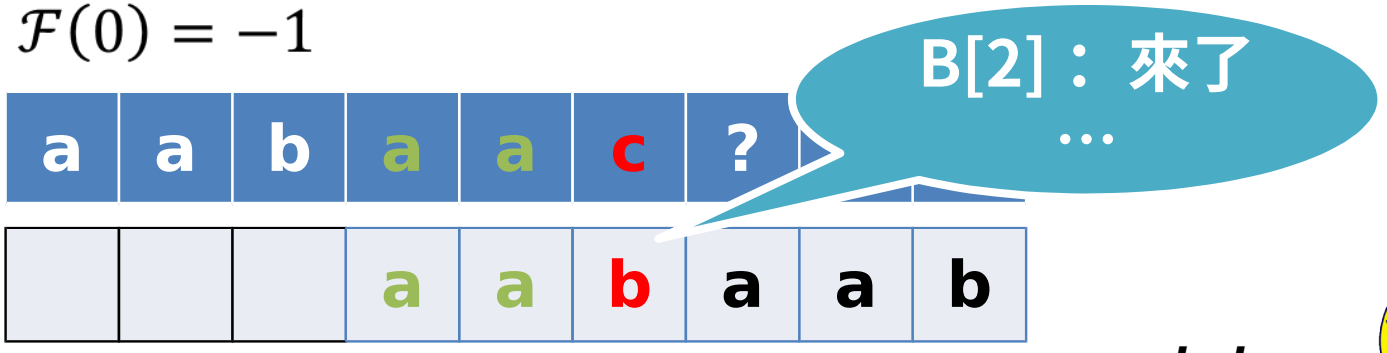
- Knuth-Morris-Pratt algorithm
- 怎麼處理 B ??
- 定義 **Fail function**（失敗函數）
  - 期望：$\mathcal{F}(i)$ 能知道匹配失敗時，B 要對齊哪裡繼續匹配
  - $\mathcal{F}_B(i) =$

$$\begin{cases} \max\{k:\ P_B(k) = B[0,k] = B[i-k,i]\}, if\ i \neq 0\ and\ at\ least\ a\ \boldsymbol{k}\ exists \\ -1,\ else \end{cases}$$

  - $\mathcal{F}(0) = -1$
  - 

made by Jingfei

KMP

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

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								

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

```
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						

```
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**					

```
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**				

```
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**			

```
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
```

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
```

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**

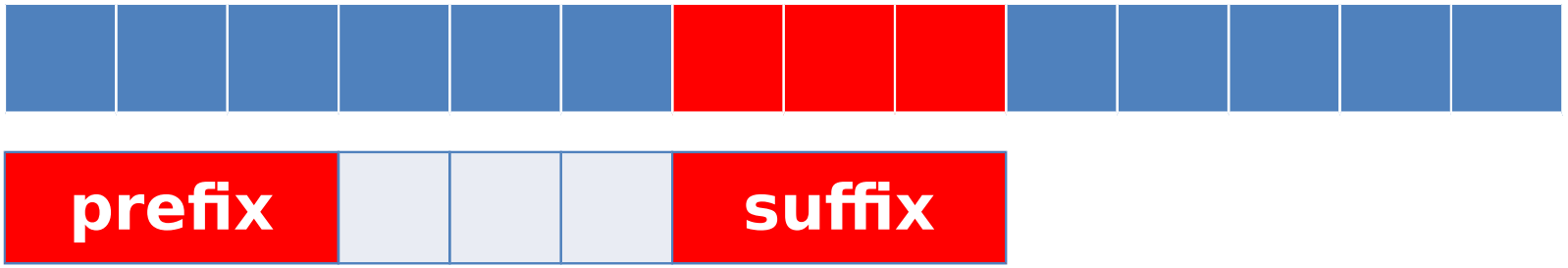
```
B[cur_pos+1]!=B[i]
pi[i]=pi[cur_pos]
```

KMP

- Fail function

```
inline void fail(char *B,int *pi){
   int len = strlen(B);
   pi[0] = -1;
   for(int i=1,cur_pos=-1;i<len;++i){
      while(~cur_pos && 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

A[i]==B[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

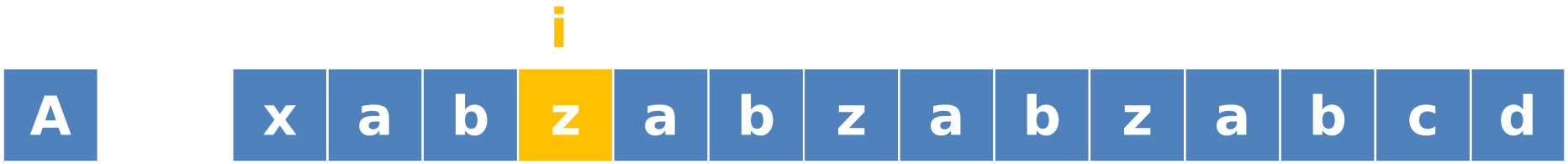
cur_pos

		-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++]
++cur_pos

KMP

- Matching



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

KMP

- Matching

i

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

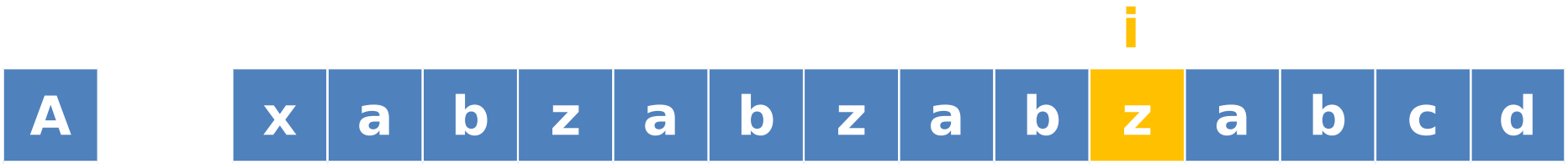
A[i]==B[cur_pos++]

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



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=pi[cur_po

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++]
++cur_pos

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	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++]

									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+

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

made by Jingfei

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+

++cur_pos

cur_pos+1==len(B

Match!!!

		-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+

++cur_pos

cur_pos+1==len(

cur_pos=pi[cur_po

KMP

- Matching

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

i

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

```
inline void match(char *A,char *B,int *pi){
   int lenA = strlen(A);
   int lenB = strlen(B);
   for(int i=1,cur_pos=-1;i<lenA;++i){
      while(~cur_pos && A[i]!=B[cur_pos+1])
         cur_pos = pi[cur_pos];
      if(A[i]==B[cur_pos+1])++cur_pos;
      if(cur_pos+1==lenB){
       /*Match!!*/
       cur_pos = pi[cur_pos];
      }
   }
}
```

made by Jingfei

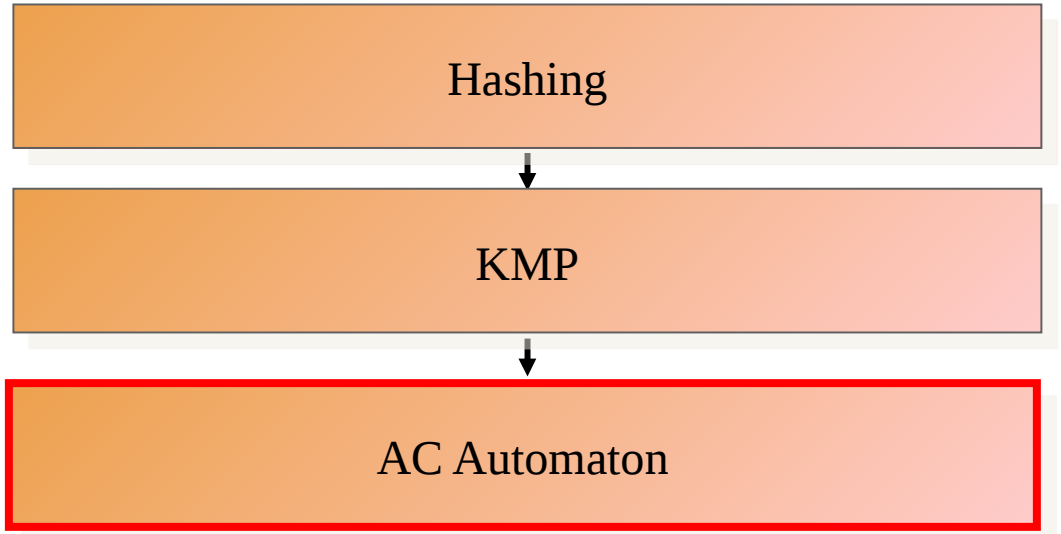
KMP

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

Example

- POJ 3461
- UVA 455

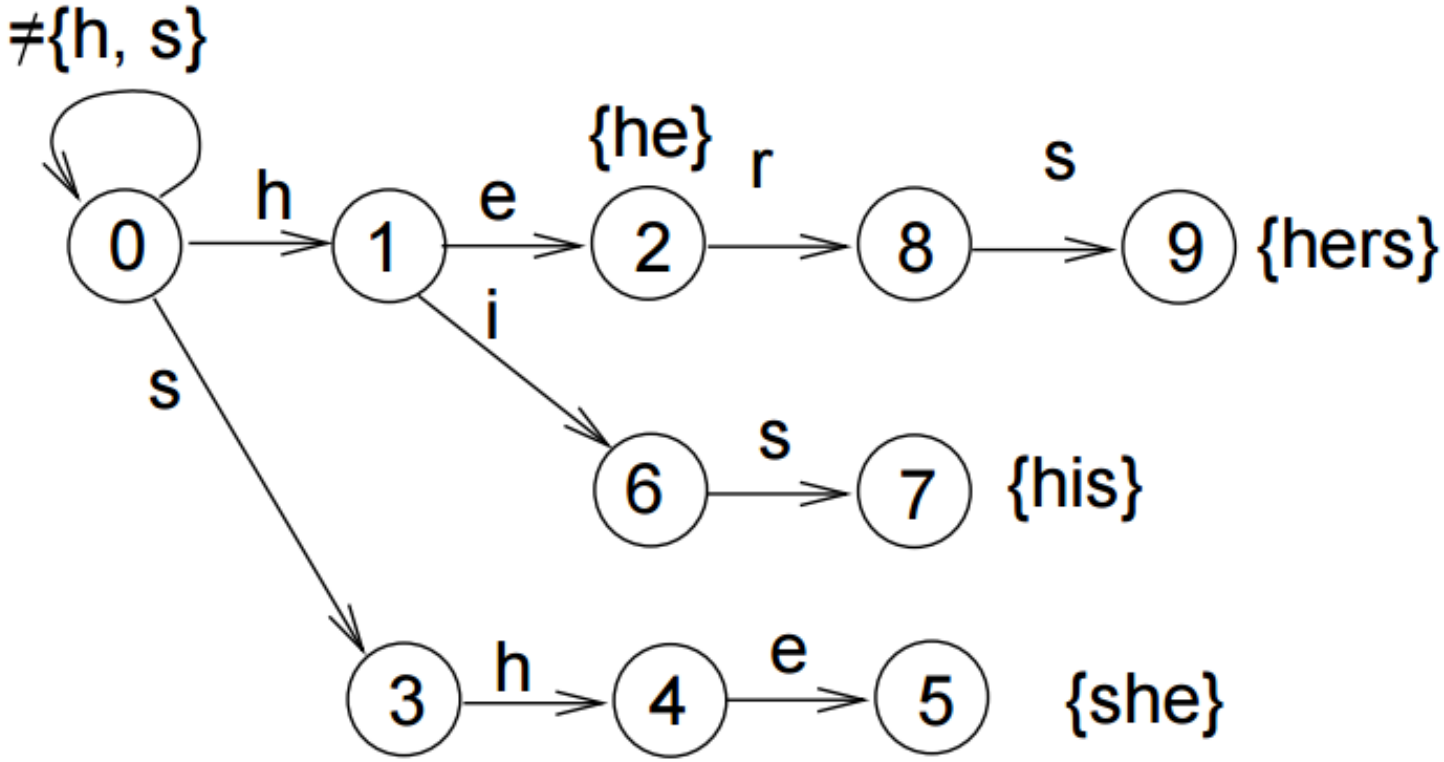
Outline



AC Automaton

- KMP 複雜度 $\mathcal{O}(|A| + |B|)$
- 多字串匹配
  1. 一個字串 B 匹配很多字串 A_i
     → $\mathcal{O}(\sum|A_i| + |B|)$
     → 線性
  2. 很多字串 B_i 匹配一個字串 A
     → $\mathcal{O}(n|A| + \sum|B_i|)$
     → 弱弱的
- Trie：儲存多個字串
- AC 自動機 = KMP + Trie

AC Automaton

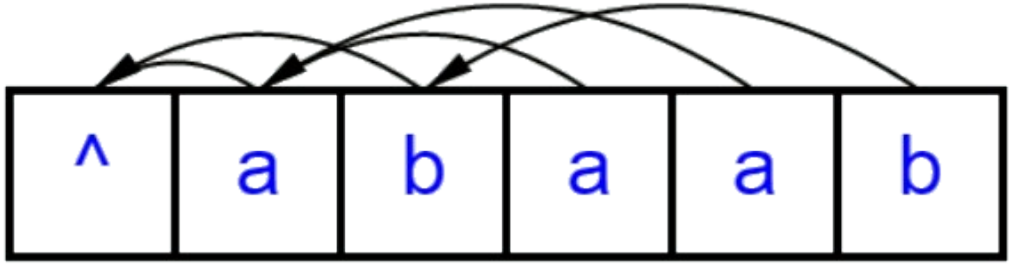


- Trie：儲存多個字串
- AC 自動機 = KMP + Trie

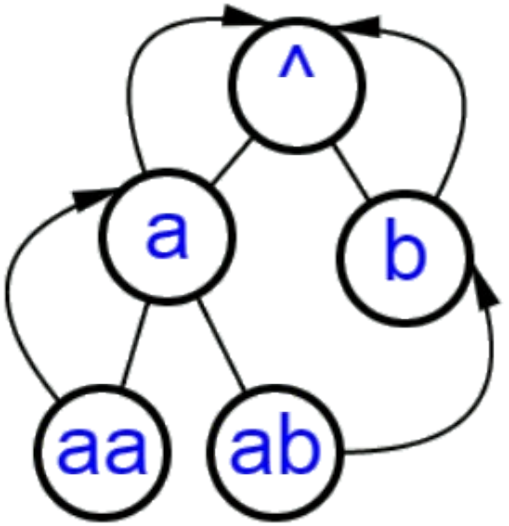
AC Automaton

- 比較 Fail function (圖)

KMP:



AC自動機:



AC Automaton

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

AC Automaton

- 比較 Fail function (匹配失敗)
- KMP
  - 沿著 $\mathcal{F}(i), \mathcal{F}^2(i), \ldots$ 嘗試，直到 $\mathcal{F}^t(i) = -1$
- AC Automaton
  - 沿著 $\mathfrak{F}(v), \mathfrak{F}^2(v), \ldots$ 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 $g(u)$ 求出 $g(v)$ ，u 為 v 的父節點
  - use BFS

Automaton – 範例 code

- 用計長的模板庫
  - [Aho-Corasick Automaton] AC自動機
- AC自动机算法详解
- AC自动机基础入门(PPT)，by李翔

- 用計長的模板庫
  - [Aho-Corasick Automaton] AC自動機
- AC自動机算法詳解
- AC自动机基础入门(PPT)，by李翔

Automaton – 例題

- AC Automaton
  - SPOJ NSUBSTR 題解
  - SPOJ SUBLEX 題解
  - Codeforces 235C 題解
  - 這些都是作法很多（Suffix Array, Suffix Tree...），非常經典的問題，可以從中理解 SAM 的精妙之處

- UVa 10679, 1449

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