



# 数据结构与算法 (A) -W04/字符串

北京大学 陈斌

2024.09.25



## 第四章 字符串

赵海燕 主讲

采用教材：《数据结构与算法》，张铭，王腾蛟，赵海燕 编写  
高等教育出版社，2008.6（“十二五”国家级规划教材）

<http://jpk.pku.edu.cn/course/sjjg/>

<https://www.icourse163.org/course/PKU-1002534001>



## 主要内容

- 字符串基本概念
- 字符串的存储结构
- 字符串运算的算法实现
- 字符串的模式匹配
  - 朴素算法
  - KMP算法



## 4.1 字符串基本概念

- 字符串，特殊的线性表，即元素为字符的线性表
- $n ( \geq 0 )$  个字符的有限序列，  
一般记作  $S : "c_0c_1c_2 \dots c_{n-1}"$ 
  - $S$  是**串名字**
  - $"c_0c_1c_2 \dots c_{n-1}"$  是**串值**
  - $c_i$  是串中的**字符**
  - **$n$  是串长**：一个字符串所包含的字符个数
    - **空串**：长度为零的串，它不包含任何字符内容



## 4.1 字符串基本概念

# 字符/符号

- **字符(char)**：组成字符串的基本单位

取值依赖于字符集 $\Sigma$ （同线性表，结点的有限集合）

- 二进制字符集： $\Sigma = \{0, 1\}$
- 生物信息中DNA字符集： $\Sigma = \{A, C, G, T\}$
- 英语语言： $\Sigma = \{26\text{个字符}, \text{标点符号}\}$
- .....



## 字符串的数据类型

- 因编程语言而不同
- 简单类型
  - 复合类型
- 字符串常数和变量
  - 字符串常数 (string literal)
    - 例如: `"\n"`, `"a"`, `"student"`...
  - 字符串变量



## 4.1 字符串基本概念

# 字符编码

- 单字节 (8 bits)
  - 采用 ASCII 码对 128 个符号进行编码
  - 在 C 和 C++ 中均采用
- 其他编码方式
  - GB
  - CJK
  - UNICODE



# 处理子串(Substring)的函数

- 子串定义：假设  $s_1, s_2$  是两个串：

$$s_1 = a_0a_1a_2\dots a_{n-1}$$

$$s_2 = b_0b_1b_2\dots b_{m-1}$$

其中  $0 \leq m \leq n$ , 若存在整数  $i$  ( $0 \leq i \leq n-m$ ),  
使得  $b_j = a_{i+j}$ ,  $j = 0, 1, \dots, m-1$  同时成立,  
则称串  $s_2$  是串  $s_1$  的子串, 或称  $s_1$  包含串  $s_2$

- 特殊子串

- 空串是任意串的子串
- 任意串  $S$  都是  $S$  本身的子串
- 真子串：非空且不为自身的子串





## 4.1 字符串基本概念

- 子串是字符串的连续片段
- 例如, software
  - 子串: 空串、software、soft、oft...
  - 不是子串: fare、sfw...
- “子串函数”
  - 提取子串
  - 插入子串
  - 寻找子串
  - 删除子串
  - ...



## 4.1 字符串基本概念

操作类别	方法	描述
子串	substr ()	返回一个串的子串
拷贝/交换	swap ()	交换两个串的内容
	copy ()	将一个串拷贝到另一个串中
赋值	assign ()	把一个串、一个字符、一个子串赋值给另一个串中
	=	把一个串或一个字符赋值给另一个串中
插入/追加	insert()	在给定位置插入一个字符、多个字符或串
	append () / +=	将一个或多个字符、或串追加在另一个串后
拼接	+	通过将一个串放置在另一个串后面来构建新串
查询	find ()	找到并返回一个子序列的开始位置
替换/清除	replace ()	替换一个指定字符或一个串的字串
	clear ()	清除串中的所有字符
统计	size () / length()	返回串中字符的数目
	max_size ()	返回串允许的最大长度



## 字符串中的字符

- 重载下标运算符[ ]

```
char& string::operator [] (int n);
```

- 按字符定位下标

```
int string::find(char c, int start=0);
```

- 反向寻找，定位尾部出现的字符

```
int string::rfind(char c, int pos=0);
```



## 4.2 字符串的存储结构和实现

- 字符串的顺序存储
- 字符串类的存储结构
- 串的运算实现



## 字符串的顺序存储

- 对串长变化不大的字符串，有三种处理方案：
  - (1) 用  $S[0]$  作为记录串长的存储单元 (Pascal)
    - 缺点：限制了串的最大长度不能超过256
  - (2) 为存储串的长度，另辟一个存储的地方
    - 缺点：串的最大长度一般是静态给定的，不是动态申请数组空间
  - (3) 用一个特殊的末尾标记 `'\0'` (C/C++)
    - 例如：C 语言的 string 函数库 `<string.h>` 采用这一存储结构



## 补充：较早期串的存储

### • 顺序

- 字编址(压缩、非压缩)

Pascal 中一般采用压缩的字编址形式, packed array

- 字节编址

### • 索引

- 有较多子串的命名串常量
- 以串名为关键码组织符号表

### • 链接

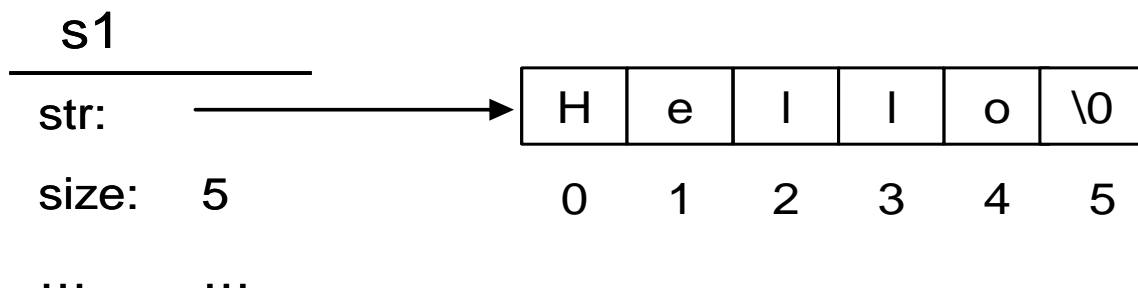
- 一般用单链(因为顺序处理)
- 一个结点中存储多个字符
- 插入、删除方便, 但存储密度小



# 字符串类的存储结构

```
private:    // 具体实现的字符串存储结构
char *str;  // 字符串的数据表示
int size;   // 串的当前长度
```

例如,  
String s1 = "Hello";





## 4.2 字符串的存储结构和实现

## 串运算的实现

## // 字符串的比较

```
int strcmp(const char *s1, const char *s2) {  
    int i = 0;  
    while (s2[i] != '\0' && s1[i] != '\0') {  
        if (s1[i] > s2[i])  
            return 1;  
        else if (s1[i] < s2[i])  
            return -1;  
        i++;  
    }  
    if (s1[i] == '\0' && s2[i] != '\0')  
        return -1;  
    else if s2[i] == '\0' && s1[i] != '\0')  
        return 1;  
    return 0;  
}
```





## 4.2 字符串的存储结构和实现

## 更简便的算法

```
int strcmp_1(char *d, char *s) {  
    int i;  
    for (i=0; d[i]==s[i]; ++i) {  
        if(d[i]=='\0' && s[i]=='\0')  
            return 0;           //两个字符串相等  
    }  
    //不等,比较第一个不同的字符  
    return (d[i]-s[i])/abs(d[i]-s[i]);  
}
```



## 4.3 字符串的模式匹配

- 模式匹配(pattern matching)
  - 一个目标对象 T (字符串)  
(pattern) P (字符串)
- 在目标T中寻找一个给定的模式P的过程
- 应用
  - 文本编辑时的特定词、句的查找 (UNIX/Linux: sed, awk, grep)
  - DNA 信息的提取
  - 确认是否具有某种结构
  - ...



## 4.3 字符串的模式匹配

## 字符串的模式匹配

- 用给定的模式  $P$ ，在目标字符串  $T$  中搜索与模式  $P$  全同的一个子串，并求出  $T$  中第一个和  $P$  全同匹配的子串（简称为“**配串**”），返回其首字符位置

$$\begin{array}{ccccccccccc}
 T & t_0 & t_1 & \cdots & t_i & t_{i+1} & t_{i+2} & \cdots & t_{i+m-2} & t_{i+m-1} & \cdots & t_{n-1} \\
 & & & & \parallel & \parallel & \parallel & & \parallel & \parallel & & \\
 P & & & & p_0 & p_1 & p_2 & \cdots & p_{m-2} & p_{m-1} & & 
 \end{array}$$

为使模式  $P$  与目标  $T$  匹配，必须满足

$$p_0 p_1 p_2 \cdots p_{m-1} = t_i t_{i+1} t_{i+2} \cdots t_{i+m-1}$$



## 模式匹配的目标

- 在大文本（诸如，句子、段落，或书本）中定位（查找）特定的模式
- 解决模式匹配问题的算法
  - 朴素（称为 “Brute Force”，也称 “Naive”）
  - Knuth-Morris-Pratt ( KMP 算法 )
  - .....



## 朴素模式匹配（穷举法）

设  $T = t_0 t_1 t_2 \dots t_{n-1}$ ,  $P = p_0 p_1 \dots p_{m-1}$

$i$  为  $T$  中字符的下标,  $j$  为  $P$  中字符的下标

匹配成功 (  $p_0 = t_i, p_1 = t_{i+1}, \dots, p_{m-1} = t_{i+m-1}$  )

即,  $T.substr(i, m) == P.substr(0, m)$

匹配失败 (  $p_j \neq t_i$  )时,

将  $P$  右移再行比较

尝试所有的可能情况



## 4.3 字符串的模式匹配

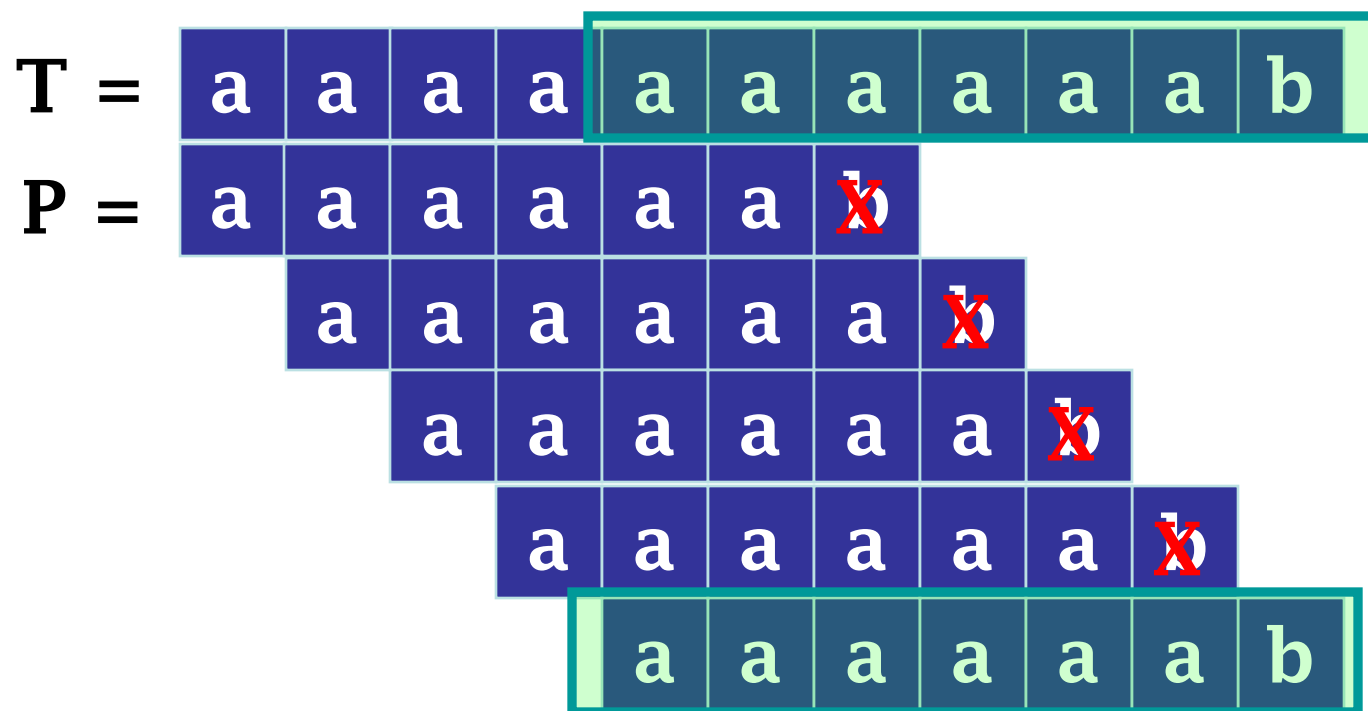
## 朴素模式匹配例1

T=	a	b	a	b	a	b	a	b	a	b	a	b	b
P=	a	b	a	b	a	b	×						
		×	b	a	b	a	b	b					
			a	b	a	b	a	b	×				
				×	b	a	b	a	b	b			
					a	b	a	b	a	b	×		
						×	b	a	b	a	b	b	
							a	b	a	b	a	b	b



## 4.3 字符串的模式匹配

## 朴素匹配例2





## 4.3 字符串的模式匹配

## 朴素匹配例3

T = 

a	b	c	d	e	f	a	b	c	d	e	f	f
---	---	---	---	---	---	---	---	---	---	---	---	---

P = 

a	b	c	d	e	f	<del>f</del>
---	---	---	---	---	---	--------------

a	a	a	a	a	a	a	b	c	d	e	f	f
---	---	---	---	---	---	---	---	---	---	---	---	---

 ✓





# 朴素模式匹配算法：其一

```
int FindPat_1(string S, string P, int startindex) {  
    // 从S末尾倒数一个模式长度位置  
    int LastIndex = S.length() - P.length();  
    int count = P.length();  
    // 开始匹配位置startindex的值过大, 匹配无法成功  
    if (LastIndex < startindex)  
        return (-1);  
    // g为S的游标, 用模式P和S第g位置子串比较, 若失败则继续循环  
    for (int g = startindex; g <= LastIndex; g++) {  
        if (P == S.substr(g, count))  
            return g;  
    }  
    // 若for循环结束, 则整个匹配失败, 返回值为负,  
    return (-1);  
}
```



## 4.3 字符串的模式匹配

## 朴素模式匹配算法：其二

```
int FindPat_2(string T, string P, int startindex) {  
    // 从T末尾倒数一个模板长度位置  
    int LastIndex = T.length() - P.length();  
    // 开始匹配位置startindex的值过大，匹配无法成功  
    if (LastIndex < startindex) return (-1);  
    // i 是指向T内部字符的游标，j 是指向P内部字符的游标  
    int i = startindex, j = 0;  
    while (i < T.length() && j < P.length()) // “<=”呢？  
        if (P[j] == T[i])  
            { i++; j++; }  
        else  
            { i = i - j + 1; j = 0; }  
    // 若匹配成功，则返回该T子串的开始位置；若失败，函数返回值为负  
    if (j >= P.length()) // “>” 可以吗？  
        return (i - j);  
    else return -1;  
}
```



## 朴素模式匹配代码（简洁）

```
int FindPat_3(string T, string P, int startindex) {  
    //g为T的游标，用模板P和T第g位置子串比较，  
    //若失败则继续循环  
    for (int g= startindex; g <= T.length() - P.length(); g++) {  
        for (int j=0; ((j<P.length()) && (T[g+j]==P[j])) ; j++)  
            ;  
        if (j == P.length())  
            return g;  
    }  
    return(-1);    // for结束，或startindex值过大，则匹配失败  
}
```



## 模式匹配原始算法：效率分析

- 假定目标  $T$  的长度为  $n$ ，模式  $P$  长度为  $m$ ， $m \leq n$ 
  - 在最坏的情况下，每一次循环都不成功，则一共要进行比较  $(n-m+1)$  次
  - 每一次“相同匹配”比较所耗费的时间，是  $P$  和  $T$  逐个字符比较的时间，最坏情况下，共  $m$  次
  - 因此，整个算法的最坏时间开销估计为

$$O(m \cdot n)$$



## 4.3 字符串的模式匹配

# 思考

- 若字符串  $S = \text{"software"}$ ，则其子串的数目是多少？
- 请分析朴素模式匹配效率低下的原因



## 主要内容

- 字符串基本概念
- 字符串的存储结构
- 字符串运算的算法实现
- 字符串的模式匹配
  - 朴素算法
  - **KMP算法**



## 4.3 字符串的模式匹配

## 无回溯匹配

- 匹配过程中, 一旦  $p_j$  和  $t_i$  比较不等时, 即
$$P.substr(1, j-1) == T.substr(i-j+1, j-1)$$
但  $p_j \neq t_i$ 
  - 该用  $P$  中的哪个字符  $p_k$  和  $t_i$  进行比较?
  - 确定右移的位数
  - 显然有  $k < j$ , 且不同的  $j$ , 其  $k$  值不同
- Knuth-Morrit-Pratt (KMP)算法
  - $k$  值仅仅依赖于模式  $P$  本身, 而与目标对象  $T$  无关



## 4.3 字符串的模式匹配

## KMP算法思想

 $T = \text{a b c d e f a b c d e f f}$ 
 $P = \text{a b c d e f f}$ 

$$T \quad t_0 \quad t_1 \quad \dots \quad t_{i-j-1} \quad t_{i-j} \quad t_{i-j+1} \quad t_{i-j+2} \quad \dots \quad t_{i-2} \quad t_{i-1} \quad t_i \quad \dots \quad t_{n-1}$$

$$\quad \quad \quad \parallel \quad \parallel \quad \quad \parallel \quad \parallel \quad \quad \quad \times$$
 $P \quad \quad \quad p_0 \quad p_1 \quad p_2 \quad \dots \quad p_{j-2} \quad p_{j-1} \quad p_j$ 

则有  $t_{i-j} \quad t_{i-j+1} \quad t_{i-j+2} \quad \dots \quad t_{i-1} = p_0 \quad p_1 \quad p_2 \quad \dots \quad p_{j-1}$  (1)

**朴素下一趟**  $p_0 \quad p_1 \quad \dots \quad p_{j-2} \quad p_{j-1}$

如果  $p_0 \quad p_1 \quad \dots \quad p_{j-2} \neq p_1 \quad p_2 \quad \dots \quad p_{j-1}$  (2)

则立刻可以断定

$$p_0 \quad p_1 \quad \dots \quad p_{j-2} \neq t_{i-j+1} \quad t_{i-j+2} \quad \dots \quad t_{i-1}$$

(朴素匹配的)下一趟一定不匹配, 可以跳过去

$$p_0 \quad p_1 \quad \dots \quad p_{j-2} \quad p_{j-1}$$





## 4.3 字符串的模式匹配

$T =$ 

a	b	c	d	e	f	a	b	c	d	e	f	f
---	---	---	---	---	---	---	---	---	---	---	---	---

$P =$ 

a	b	c	d	e	f	f		
		a	b	c	d	e	f	f

同样, 若  $p_0 p_1 \dots p_{j-3} \neq p_2 p_3 \dots p_{j-1}$   
则再下一趟也不匹配, 因为有

$$p_0 p_1 \dots p_{j-3} \neq t_{i-j+2} t_{i-j+3} \dots t_{i-1}$$

直到对于某一个 “ $k$ ” 值 (首尾串长度), 使得

$$p_0 p_1 \dots p_k \neq p_{j-k-1} p_{j-k} \dots p_{j-1}$$

且

$$p_0 p_1 \dots p_{k-1} = p_{j-k} p_{j-k+1} \dots p_{j-1}$$

模式右滑  $j-k$  位

$$\begin{array}{ccccccc} t_{i-k} & t_{i-k+1} & \dots & t_{i-1} & t_i & & \\ \parallel & \parallel & & \parallel & \times & & \\ p_{j-k} & p_{j-k+1} & \dots & p_{j-1} & p_j & & \\ \parallel & \parallel & & \parallel & ? & & \\ p_0 & p_1 & \dots & p_{k-1} & p_k & & \end{array}$$

则  $p_0 p_1 \dots p_{k-1} = t_{i-k} t_{i-k+1} \dots t_{i-1}$



## 字符串的特征向量N

设模式  $P$  由  $m$  个字符组成, 记为

$$P = p_0 p_1 p_2 p_3 \dots p_{m-1}$$

令 **特征向量**  $N$  用来表示模式  $P$  的字符分布特征, 简称  **$N$  向量** 由  $m$  个特征数  $n_0 \dots n_{m-1}$  整数组成, 记为

$$N = n_0 n_1 n_2 n_3 \dots n_{m-1}$$

$N$  在很多文献中也称为 next 数组, 每个  $n_j$  对应 next 数组中的元素  $\text{next}[j]$



## 字符串的特征向量N：构造方法

- P 第  $j$  个位置的特征数  $n_j$ , 首尾串最长的  $k$ 
  - 首串:  $p_0 \ p_1 \ \dots \ p_{k-2} \ p_{k-1}$
  - 尾串:  $p_{j-k} \ p_{j-k+1} \ \dots \ p_{j-2} \ p_{j-1}$

$$\text{next}[j] = \begin{cases} -1, & 0 \\ \max\{k: 0 < k < j \wedge P[0..k-1] = p[j-k..j-1]\}, & \text{存在最长首尾配串 } k \\ 0, & \text{otherwise} \end{cases}$$



## 4.3 字符串的模式匹配

如果不是“最长”首尾配串？

P =

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

N =

-1	0	1	2	1	0	1	2	3	4
----	---	---	---	---	---	---	---	---	---

X (应为3)

T =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
a	a	b	a	a	a	a	a	a	b	a	a	a	a	c	b

P =

a	a	<del>x</del>	a	b	a	a	a	a	c
---	---	--------------	---	---	---	---	---	---	---

i=2, j=1, N[j]=0

	a	a	a	a	<del>x</del>	a	a	a	a	c
--	---	---	---	---	--------------	---	---	---	---	---

i=7, j=4, N[4]=~~1~~  
X

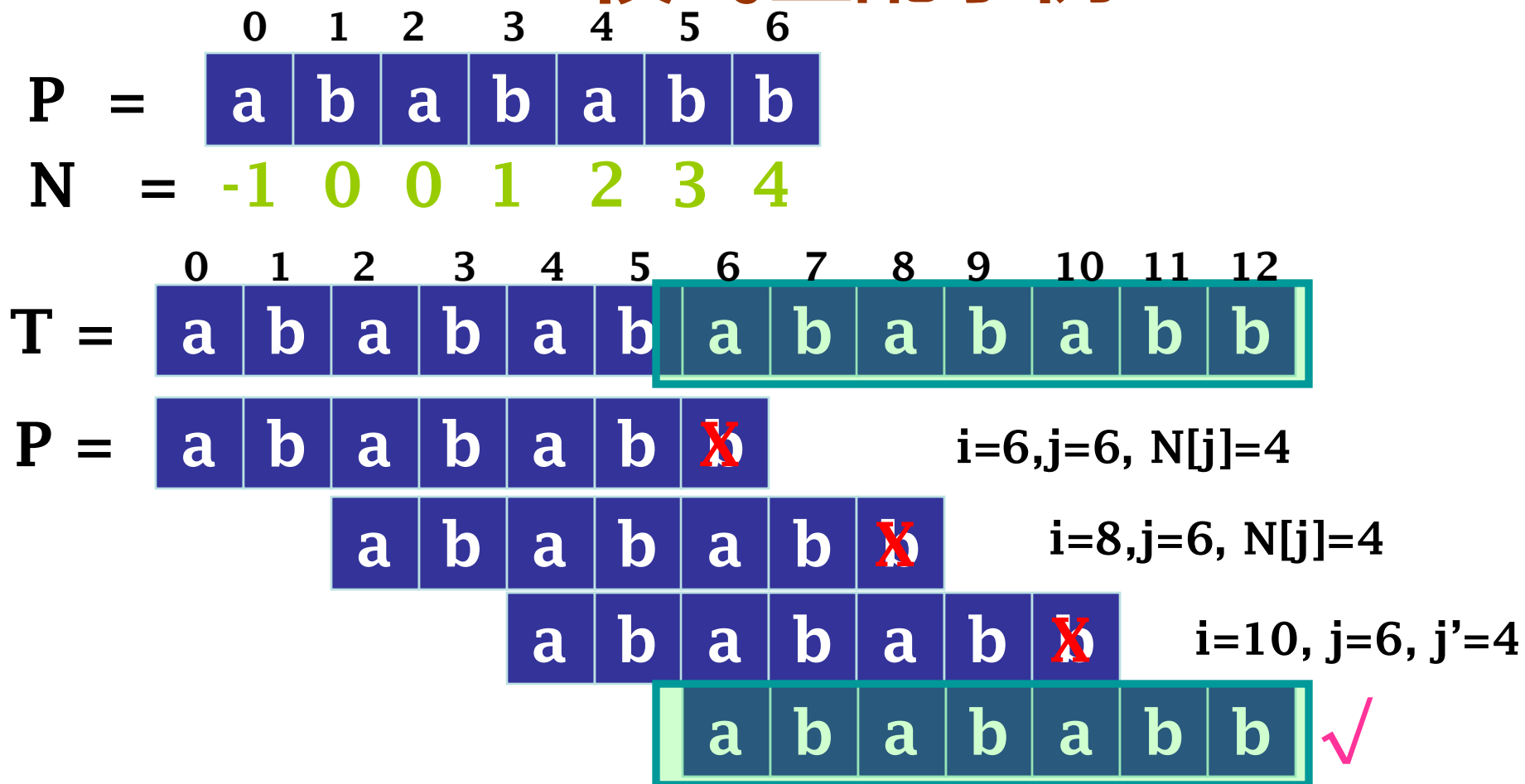
错过了!

a	a	a	a	b	a	a	a	a	c
---	---	---	---	---	---	---	---	---	---



## 4.3 字符串的模式匹配

## KMP模式匹配示例





## 4.3 字符串的模式匹配

## KMP模式匹配算法

```
int KMPStrMatching(string T, string P, int *N, int start) {  
    int j= 0;                // 模式的下标变量  
    int i = start;           // 目标的下标变量  
    int pLen = P.length( );  // 模式的长度  
    int tLen = T.length( );  // 目标的长度  
    if (tLen - start < pLen)  // 若目标比模式短, 匹配无法成功  
        return (-1);  
    while ( j < pLen && i < tLen) { // 反复比较, 进行匹配  
        if ( j == -1 || T[i] == P[j])  
            i++, j++;  
        else j = N[j];  
    }  
    if (j >= pLen)  
        return (i-pLen);      // 注意仔细算下标  
    else return (-1);  
}
```



## 对应的求特征向量算法框架

- 特征数  $n_j$  ( $j > 0, 0 \leq n_{j+1} \leq j$ ) 是递归定义的, 定义如下:
  - $n_0 = -1$ , 对于  $j > 0$  的  $n_{j+1}$ , 假定已知前一位置的特征数  $n_j$ , 令  $k = n_j$ ;
  - 当  $k \geq 0$  且  $p_j \neq p_k$  时, 则令  $k = n_k$ ; 让步骤2循环直到条件不满足
  - $n_{j+1} = k + 1$ ; // 此时,  $k == -1$  或  $p_j == p_k$



# 字符串的特征向量N ——非优化版

```
int findNext(string P) {  
    int j, k;  
    int m = P.length( );  
    assert( m > 0);  
    int *next = new int[m];  
    assert( next != 0);  
    next[0] = -1;  
    j = 0; k = -1;  
    while (j < m-1) {  
        while (k >= 0 && P[k] != P[j]) // 不等则采用 KMP 自找首尾子串  
            k = next[k];                // k 递归地向前找  
        j++; k++; next[j] = k;  
    }  
    return next;  
}
```





## 4.3 字符串的模式匹配

## 求特征向量N

$N =$ 

-1	0	1	2	3	0	1	2	3	4
----	---	---	---	---	---	---	---	---	---

0    1    2    3    4    5    6    7    8    9

$P =$ 

a	a	a	a	b	a	a	a	a	c
---	---	---	---	---	---	---	---	---	---

$j =$ 

9
---

 $k =$ 

0
---

首串→ 

a		
a	a	
a	a	a

首串→ 


a			
a	a		
a	a	a	
a	a	a	a



## 4.3 字符串的模式匹配

## 模式右滑j-k位

$t_{i-j}$	$t_{i-j+1}$	$t_{i-j+2}$	$\dots$	$t_{i-k}$	$t_{i-k+1}$	$\dots$	$t_{i-1}$	$t_i$
								×
$p_0$	$p_1$	$p_2$	$\dots$	$p_{j-k}$	$p_{j-k+1}$	$\dots$	$p_{j-1}$	$p_j$
								?
				$p_0$	$p_1$	$\dots$	$p_{k-1}$	$p_k$



$$p_0 p_1 \dots p_{k-1} = t_{i-k} t_{i-k+1} \dots t_{i-1}$$

$$t_i \neq p_j, \quad p_j == p_k?$$



## 4.3 字符串的模式匹配

## KMP匹配

j	0	1	2	3	4	5	6	7	8
P	a	b	c	a	a	b	a	b	c
K		0	0	0	1	1	2	1	2

目标

*a a b c b a b c a a b c a a b a b c*

*a ~~b~~ c a a b a b c*

*a b c ~~a~~ b a b c*

这行冗余

*~~a~~ b c a a b a b c*

*a b c a a b ~~a~~ b c*

*a b c a a b a b c*

 $N[1] = 0$ 
 $N[3] = 0$ 
 $N[0] = -1$ 
 $N[6] = 2$ 


上面  $P[3] == P[0]$ ,  $P[3] \neq T[4]$ , 再比冗余



# 字符串的特征向量N ——优化版

```
int findNext(string P) {  
    int j, k;  
    int m = P.length( );  
    int *next = new int[m];  
    next[0] = -1;  
    j = 0; k = -1;  
    while (j < m-1) {  
        while (k >= 0 && P[k] != P[j])  
            k = next[k];  
        j++; k++;  
        if (P[k] == P[j])  
            next[j] = next[k];  
        else next[j] = k;  
    }  
    return next;  
}
```

// m为模式P的长度  
// 动态存储区开辟整数数组  
// 若写成  $j < m$  会越界  
// 若不等, 采用 KMP 找首尾子串  
// k 递归地向前找  
// 前面找 k 值, 没有受优化的影响  
// 取消if判断, 则不优化



## 4.3 字符串的模式匹配

## next数组对比

序号j	0	1	2	3	4	5	6	7	8	
P	a	b	c	a	a	b	a	b	c	
k		0	0	0	1	1	2	1	2	非优化版
$p_k == p_j?$		$\neq$	$\neq$	$==$	$\neq$	$==$	$\neq$	$==$	$==$	
next[j]	-1	0	0	-1	1	0	2	0	0	优化版



## KMP算法的效率分析

- 循环体中 “ $j = N[j];$ ” 语句的执行次数不能超过  $n$  次。否则，
  - 由于 “ $j = N[j];$ ” 每执行一次必然使得  $j$  减少(至少减1)
  - 而使得  $j$  增加的操作只有 “ $j++$ ”
  - 那么，如果 “ $j = N[j];$ ” 的执行次数超过  $n$  次，最终的结果必然使得  $j$  为 **比-1小很多的负数**。这是不可能的 ( $j$  有时为 -1, 但是很快 +1 回到 0)。
- 同理可以分析出求  $N$  数组的时间为  $O(m)$   
故，KMP算法的时间为  $O(n+m)$



## 4.3 字符串的模式匹配

## 总结：单模式的匹配算法

算法	预处理时间效率	匹配时间效率
朴素匹配算法	$O$ (无需预处理)	$\Theta(n \cdot m)$
KMP算法	$\Theta(m)$	$\Theta(n)$
BM算法	$\Theta(m)$	最优 $(n/m)$ , 最差 $\Theta(nm)$
位运算算法 ( <i>shift-or</i> , <i>shift-and</i> )	$\Theta(m +  \Sigma )$	$\Theta(n)$
Rabin-Karp算法	$\Theta(m)$	平均 $(n+m)$ , 最差 $\Theta(nm)$
有限状态自动机	$\Theta(m \cdot  \Sigma )$	$\Theta(n)$



## 参考资料

- **Pattern Matching Pointer**
  - <http://www.cs.ucr.edu/~stelo/pattern.html>
- **EXACT STRING MATCHING ALGORITHMS**
  - <http://www-igm.univ-mlv.fr/~lecroq/string/>
  - 字符串匹配算法的描述、复杂度分析和C源代码





# 数据结构与算法

感谢倾听

国家精品课 “数据结构与算法”

<http://jpk.pku.edu.cn/course/sjjg/>  
<https://www.icourse163.org/course/PKU-1002534001>

张铭, 王腾蛟, 赵海燕

高等教育出版社, 2008. 6. “十一五” 国家级规划教材