

第四章 串

4.1 串的定义

串,即**字符串 (String)**,是由零个或多个字符组成的有限序列。一般记为 $S = \text{"a}_1 \text{a}_2 \cdots \text{a}_n\text{"}$ ($n \geq 0$)其中, S 是串名, 用单引号或双引号括起来的字符序列是串的值, a_i ($1 \leq i \leq n$)可以是字母、数字或其他字符。

串是一种特殊的线性表, 它的特殊性在于: 串中的每一个数据元素仅由一个字符组成。

例:

S="HelloWorld!"

T='HuaWei mate 4.0 pro'

串中字符的数目 n 称为串的长度，长度为0的串称为空串。

串中任意个连续的字符组成的子序列称为该串的子串，包含子串的串相应地称为主串。

通常把字符在序列中的序号称为该字符在串中的位置，子串在主串中的位置则以子串的第一个字符在主串中的位置来表示。

空串是任意串的子串，任意串是其自身的子串。

串值必须用一对单引号或双引号括起来，但引号本身不属于串。

空格串不是空串。

串 VS 线性表

串是一种特殊的线性表，数据元素之间呈线性关系



串的数据对象限定为字符集（如中文字符、英文字符、数字字符、标点字符等）

串的基本操作，如增删改查等通常以子串为操作对象。

串的基本操作

假设有串T="", S="HuaWei mate 4.0 pro?", W="Pro"

StrAssign(&T,chars): 赋值操作。把串T赋值为chars。

StrCopy(&T,S): 复制操作。由串S复制得到串T。

StrEmpty(S): 判空操作。若S为空串, 则返回TRUE, 否则返回FALSE。

StrLength(S): 求串长。返回串S的元素个数。

ClearString(&S): 清空操作。将S清为空串。

DestroyString(&S): 销毁串。将串S销毁(回收存储空间)。

Concat(&T,S1,S2): 串联接。用T返回由S1和S2联接而成的新串

SubString(&Sub,S,pos,len): 求子串。用Sub返回串S的第pos个字符起长度为len的子串。

Index(S,T): 定位操作。若主串S中存在与串T值相同的子串, 则返回它在主串S中第一次出现的位置; 否则函数值为0。

StrCompare(S,T): 比较操作。若 $S>T$, 则返回值 >0 ; 若 $S=T$, 则返回值 $=0$; 若 $S<T$, 则返回值 <0 。

串的比较操作

StrCompare(S,T): 比较操作。若 $S>T$, 则返回值 >0 ; 若 $S=T$, 则返回值 $=0$; 若 $S<T$, 则返回值 <0

A

abandon/ ə'bəndən/ vt. 丢弃; 放弃, 抛弃

aboard/ ə'bo:d/ ad. 在船(车)上; 上船

absolute/ 'æbsəlu:t/ a. 绝对的; 纯粹的

absolutely/ 'æbsəlu:tli/ ad. 完全地; 绝对地

absorb/ əb'sɔ:b/ vt. 吸收; 使专心

abstract/ 'æbstrækt/ n. 摘要

abundant/ ə'bʌndənt/ a. 丰富的; 大量的

abuse/ ə'bjʊ:z, ə'bjʊ:s/ vt. 滥用; 虐待 n. 滥用

academic/ ækə'demik/ a. 学院的; 学术的

accelerate/ æk'seləreit/ vt. (使)加快; 促进

字符集编码

ASCII 字符代码表 一

高四位 低四位		ASCII非打印控制字符										ASCII 打印字符															
		0000					0001					0010	0011	0100	0101	0110	0111										
		0					1					2	3	4	5	6	7										
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl			
0000	0	0	BLANK NULL	^@	NUL 空	16	▶	^P	DLE 数据链路转意	32		48	0	64	@	80	P	96	`	112	p						
0001	1	1	☺	^A	SOH 头标开始	17	◀	^Q	DC1 设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q						
0010	2	2	☹	^B	STX 正文开始	18	↕	^R	DC2 设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r						
0011	3	3	♥	^C	ETX 正文结束	19	!!	^S	DC3 设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s						
0100	4	4	♦	^D	EOT 传输结束	20	¶	^T	DC4 设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t						
0101	5	5	♣	^E	ENQ 查询	21	⌘	^U	NAK 反确认	37	%	53	5	69	E	85	U	101	e	117	u						
0110	6	6	♠	^F	ACK 确认	22	■	^V	SYN 同步空闲	38	&	54	6	70	F	86	V	102	f	118	v						
0111	7	7	●	^G	BEL 震铃	23	↑	^W	ETB 传输块结束	39	'	55	7	71	G	87	w	103	g	119	w						
1000	8	8	◻	^H	BS 退格	24	↑	^X	CAN 取消	40	(56	8	72	H	88	X	104	h	120	x						
1001	9	9	○	^I	TAB 水平制表符	25	↓	^Y	EM 媒体结束	41)	57	9	73	I	89	Y	105	i	121	y						
1010	A	10	◼	^J	LF 换行/新行	26	→	^Z	SUB 替换	42	*	58	:	74	J	90	Z	106	j	122	z						
1011	B	11	♂	^K	VT 垂直制表符	27	←	^[ESC 转意	43	+	59	;	75	K	91	[107	k	123	{						
1100	C	12	♀	^L	FF 换页/新页	28	└	^\ FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124							
1101	D	13	🎵	^M	CR 回车	29	↔	^] GS	组分分隔符	45	-	61	=	77	M	93]	109	m	125	}						
1110	E	14	🎵	^N	SO 移出	30	▲	^6 RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~						
1111	F	15	☼	^O	SI 移入	31	▼	^- US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ		Back space				

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键” 输入

任何数据存到计算机中一定是二进制数。

需要确定一个字符和二进制数的对应规则这就是“编码”

“字符集”：

英文字符——ASCII字符集

中英文——Unicode字符集

拓展：乱码问题

¥P■u■儀■ u■儀θ u !??3离¥烂?朕@ 嬋端 姝 ??旅熾S娼嬋鐵乇 婢 姝■[脣總u嬌?嬌瑁üü!
■t3江■t2???????姝■空t■空 t■???糞D? u■????儀■ u-?婢^[朕@ SUWQ嫫場 甞■? \$峯■?
[脣卸@ 姝■脰SU娣 婢■伊t]儻■ uW儻θ u9客■RjTP鑄?ü呦■嫩尅■ (娼 塤 娼■塤■F井■f
鑄?ü脰姝 儀■ u■娼■塤■胚u娼■{■ t■{■ u3婧 婢■伊t■鑄?ü婢■P铨?ü3線F■婢璽üüü婢璧!
@ dÿ0d?娼鳴x■娼鳴p■娼■娼烏饋z■? 娼娼馱x, t■娼烏@娼鳴B■3繼YYd??環?ü娼 間!
??ü3繼YYd?h■福 h?B 桡?ü瞄■?ü臍娼 @■ ^[YY]? 娼 SUW甦▼趨饗蛙■?ü: ?劫 娼 ?■
;G■劍 嬌鐙?ü嬌桡?ü昭■P嫫■娼■嬌瑾?ü護媼■伊t¥G■uc嬌??üUW喊■近\$?
?ü■<\$ t¥湾■? 娼\$B娼嫩嬌??ü甦x_^[脣總uW甦珎鵲饗^ !;< t/嬌机?üUW覓■近\$■? 娼_^壺
乇 婢,fs■?嬌?üQ■軋 岨?珊T ?■üü婢 u猛■晤4■ 娼^拖 3霸U磺E? 娼珎u脣U痞U
<j 鑄?ü娼S鑿?ü堉 E?Sj ??ü?苔? 苔 苔豚■ 苔? 3繳h■墨 dÿ0d?岨 嬌?ü嫩
ü朕^[嬌]朕@ SUW甦珎鵲饗^ !;<■t/嬌鐙?üUW覓■近\$■? 娼_^墊\$壺\$■T耍■娼■嬌鐙?ü甦
娼PR由■ 娼\$■杓?ü娼\$■尅■也\$ 岨\$■3珊■ 鋁}üüf莖\$■BM嬌杪?ü?■侯 钲?ü婢■P婢■姝■
?ü婢P伊u■;鴉 雲?u3儻■ t-■{1 u' 娼\$■? 婢■恍?ü雲f儻>■u■妻拎一也\$!也\$■娼\$ 娼\$■娼:
娼P伊t !;鴉 珊P娼\$■? 娼璿?ü客θ? 娼瑁?üf儻>■u■鯨@ t■客x? 娼鐙?ü3禧■\$博 ■
桡?ü娼jZS鎗?üX■B sj 鑽?ü?博 ? 枋?üP ■B [朕@ u變3繳h■琳 dÿ0d?ü¥B u3?博 杞■!
%琳 瞄¥ü膝]脰??B fs 鐙üüüh?B 璦?üh ■B 铯?üj■璠?ü\■B j\钲?ü \■B j■鑿?üd■B ?
嬌3蔣予■{üü嬌^[朕@ SUWU嫫雲雲鑑?ü嬌? ?C;飢■||ü u?詠■娼槩?ü?N■|ü u廬嬌+蔣嬌
u嫫娼嬌栞?üP婢栞?üP璧?ü湮■繞^[胚u嫫娼嬌娼鑑?ü^[脣?膏■ 纒■髻? ?AN?雲■
u鸞■朕爰波■1直1詣?üü夠x??ü甦■^朕@ 裸 u-1豫爰波 Q? 鏽üüü夠x桡?ü甦 ^朕@ u變世:
琳叁üüP璦?ü岨麗岨厂岨鬚鑿?ü伊u■苔?üüü娼藕嬌]胚娼婢鑿üüü@■璋[朕變3繳h■機@ dÿ0d?!

在你的文件中，原本采用某一套编码规则 $y=f(x)$ ，如：‘码’ \leftrightarrow 0001010100010101010010

打开文件时，你的软件以为你采用的是另一套编码规则 $y=g(x)$ ，如：0001010100010101010010 \leftrightarrow



串的顺序存储

```
#define MAXLEN 255

typedef struct{
    char ch[MAXLEN];//每个分量存储一个字符
    int length;//串的实际长度
}SString;

typedef struct{
    char*ch;//按串长分配存储区，ch指向串的基地址
    int length;//串的长度
}HString;

HString s;

S.ch=(char *)malloc(MAXLEN *sizeof(char);

S.length=0;
```

串的顺序存储

方案一：



方案二：



方案三：



方案四：



串的链式存储

```
typedef struct StringNode{  
    char ch;//每个结点存1个字符  
    struct StringNode * next;  
}StringNode,* String;
```



缺点：存储密度低。

4.2 基本操作的实现

```
#define MAXLEN 255  
typedef struct {  
    char ch[MAXLEN]; // 每个分量存储一个字符  
    int length; // 串的实际长度  
} SString;
```

StrAssign(&T,chars): 赋值操作。把串T赋值为chars。

StrCopy(&T,S): 复制操作。由串S复制得到串T。

StrEmpty(S): 判空操作。若S为空串，则返回TRUE，否则返回FALSE。

StrLength(S): 求串长。返回串S的元素个数。

ClearString(&S): 清空操作。将S清为空串。

DestroyString(&S): 销毁串。将串S销毁（回收存储空间）。

Concat(&T,S1,S2): 串联接。用T返回由S1和S2联接而成的新串

基本操作的实现

```
bool SubString(SString &Sub, SString S, int pos, int len){  
    // 用Sub返回串S的第pos个字符起长度为len的子串。  
    int i;  
    if (pos< 1 || pos >S.length || len<0 || len>S.length-pos+1)  
        return false;  
    for(i=1; i<=len; i++)  
        Sub[i] = S[pos+i-1];  
    Sub.length= len;  
    return true;  
} // SubString
```

基本操作的实现

StrCompare(S,T): 比较操作。若 $S>T$ ，则返回值 >0 ；若 $S=T$ ，则返回值 $=0$ ；若 $S<T$ ，则返回值 <0 。

```
int StrCompare(SString S,SString T){
    for(int i=1;i<=S.length && i<=T.length;i++){
        if(S.ch[i]!=T.ch[i])
            return S.ch[i]-T.ch[i];
    }
    //扫描过的所有字符都相同，则长度长的串更大
    return S.length-T.length;
}
```

基本操作的实现

Index(S,T): 定位操作。若主串S中存在与串T值相同的子串，则返回它的主串S中第一次出现的位置；否则函数值为0。

S: HuaWei mate40 pro

T: Wei

基本操作的实现

Index(S,T): 定位操作。若主串S中存在与串T值相同的子串, 则返回它的主串S中第一次出现的位置; 否则函数值为0。

```
bool SubString(SString &Sub, SString S, int pos, int len)
```

```
int StrCompare(SString S,SString T)
```

```
int Index(SString S, SString T){
```

```
    int i=1, n=StrLength(S),m=StrLength(T);
```

```
    SString sub; //用于暂存子串
```

```
    while(i<=n-m+1){
```

```
        SubString(sub, S,i,m);
```

```
        if(StrCompare(sub,T)!=0) ++i;
```

```
        else return i;//返回子串在主串中的位置
```

```
    }
```

```
    return 0;//S中不存在与T相等的子串
```

```
}
```


**4.3 模式匹配

index的算法的实现

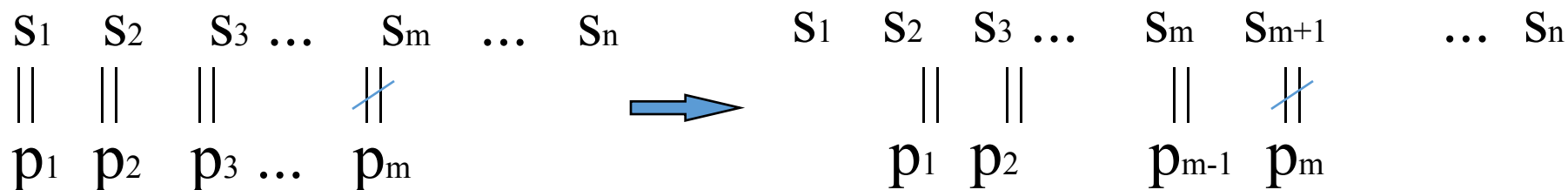
设 $S = s_1 s_2 \dots s_n$ 目标

$P = p_1 p_2 \dots p_m$ 模式

其中 $0 < m \leq n$ 。

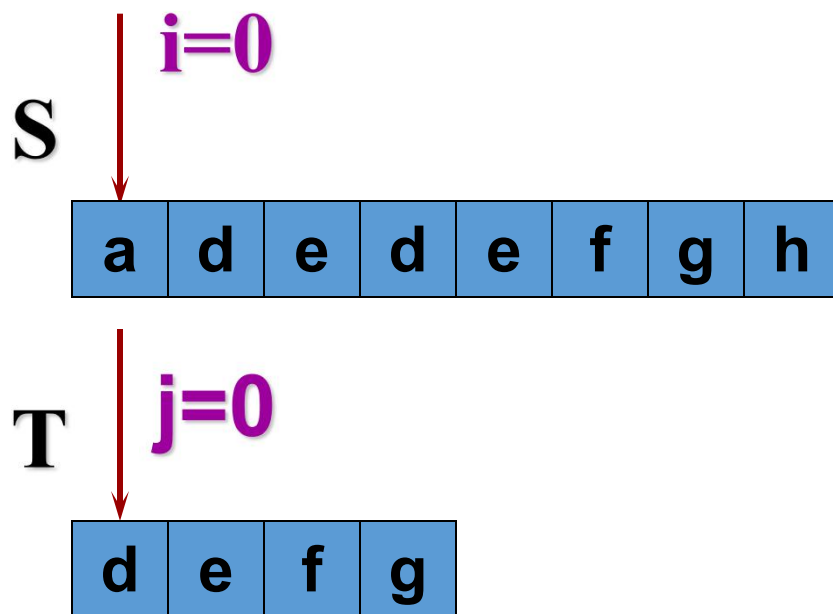
- 在S中找到一个与P相同的子串，是index的功能，称求子串序号
- 在目标S中查找模式为P的子串的过程成为模式匹配。

4.3.1.朴素的模式匹配算法

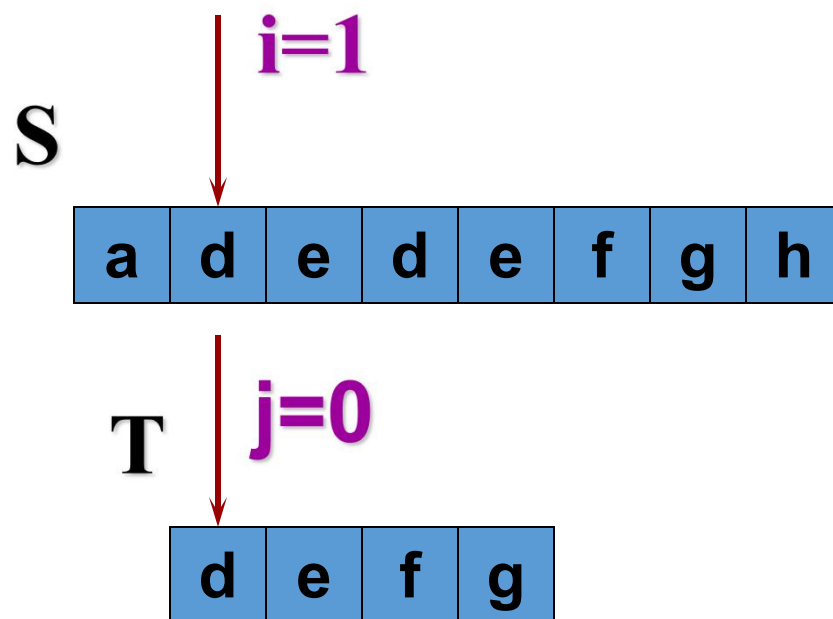


直到 某一步 $p_1 = s_i, p_2 = s_{i+1}, \dots, p_m = s_{i+m-1}$ 匹配成功。

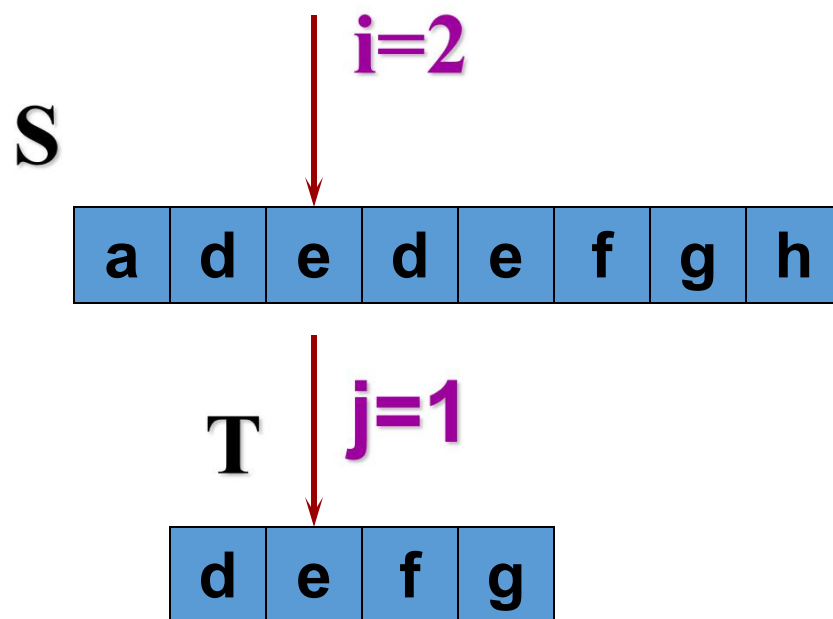
Brute-Force (BF蛮力)模式匹配的过程:



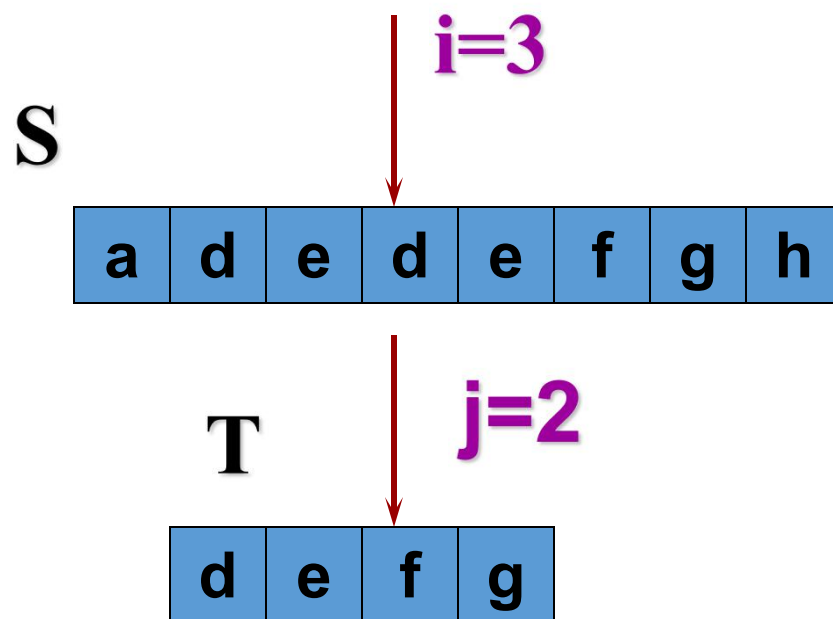
Brute-Force (BF蛮力)模式匹配的过程:



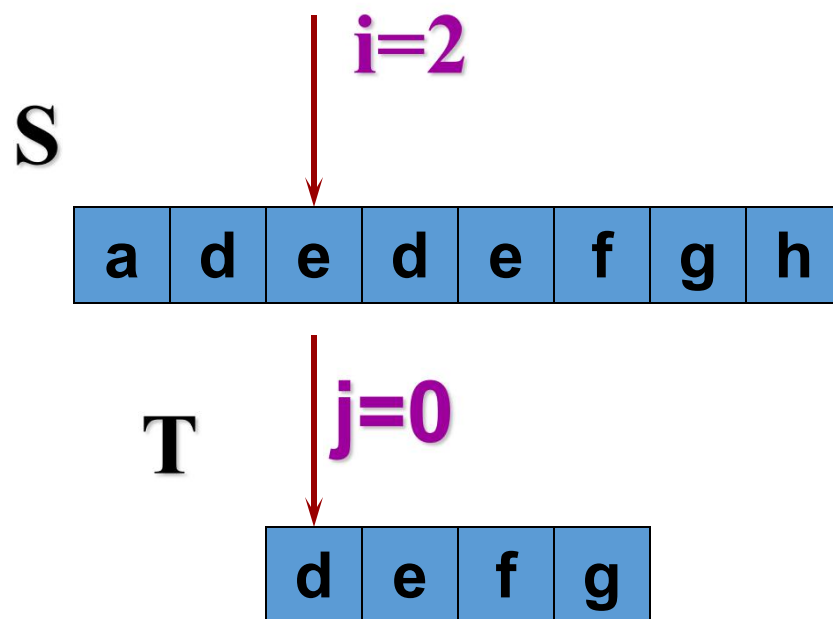
Brute-Force (BF蛮力)模式匹配的过程:



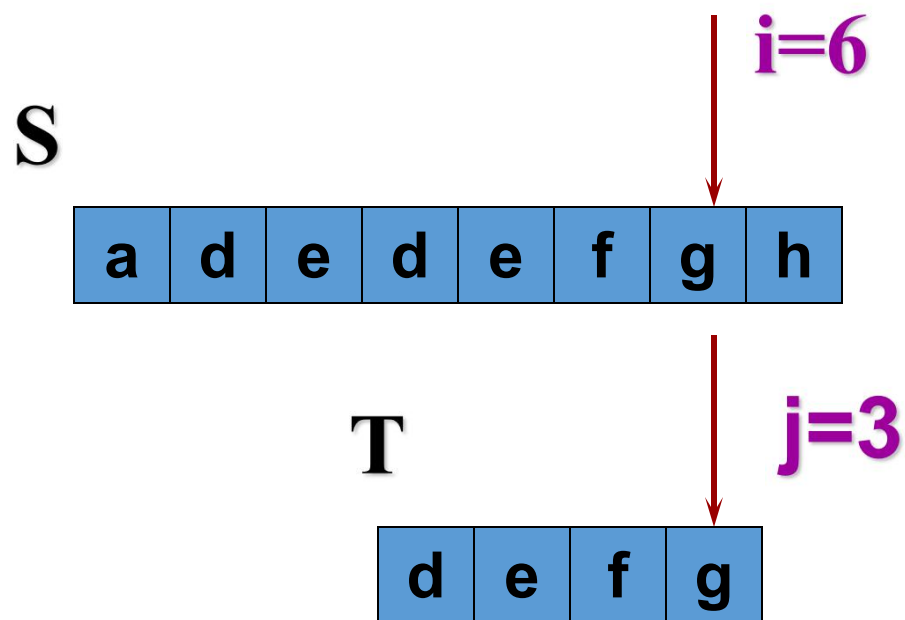
Brute-Force (BF蛮力)模式匹配的过程:



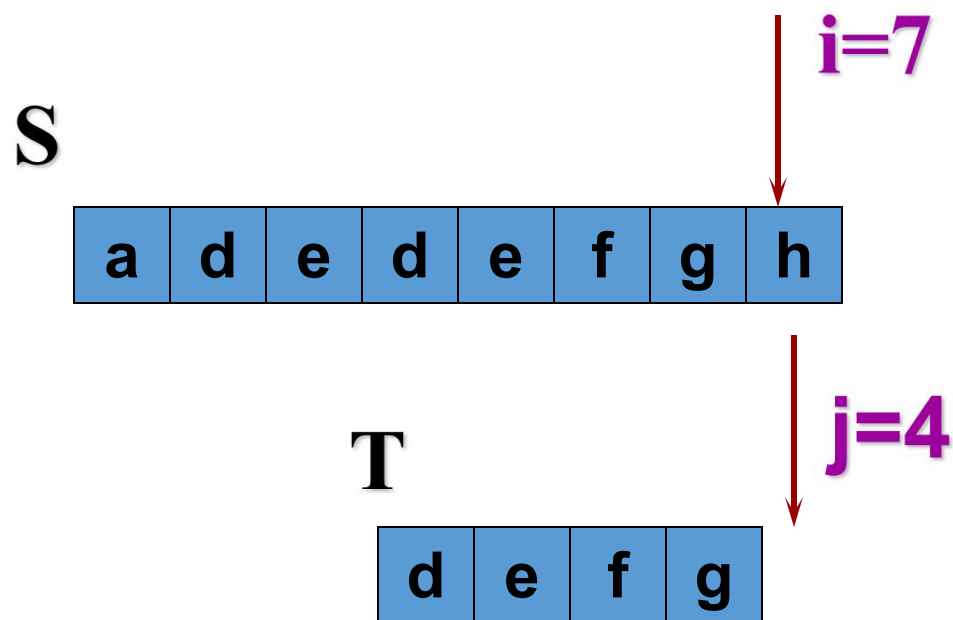
Brute-Force (BF蛮力)模式匹配的过程:



Brute-Force (BF蛮力)模式匹配的过程:



Brute-Force (BF 蛮力) 模式匹配的过程:




```
int Index(SString S, SString T, int pos) {  
    // 返回子串T在主串S中第pos个字符之后的位置。  
    // 若不存在，则函数值为0。  
    // 其中，T非空， $1 \leq \text{pos} \leq \text{StrLength}(S)$ 。  
    int i = pos; int j = 1;  
    while (i <= S.length && j <= T.length {  
        if (S[i] == T[j]) {++i; ++j;}           // 继续比较后继字符  
        else { // 指针后退重新开始匹配  
            i = i-j+2;    j = 1; }  
        }  
    if (j > T.length) return i-T.length;  
    else return 0;  
} // Index
```

```

int Index(SString S, SString T, int pos) {
    // 返回子串T在主串S中第pos个字符之后的位置。
    // 若不存在，则函数值为0。
    // 其中，T非空， $1 \leq \text{pos} \leq \text{StrLength}(S)$ 。
    int i = pos; int j = 1;
    while (i <= S.length && j <= T.length {
        if (S[i] == T[j]) {++i; ++j;}           // 继续比较后继字符
        else { // 指针后退重新开始匹配
            i = i-j+2;      j = 1; }
    }
    if (j > T.length) return i-T.length;
    else return 0;
} // Index

```

$i=i-j+2$ 公式的由来:

j-1 是比较相等的个数

则 $i-(j-1)$ 是回到本次比较开始时i的原位,

所以 $i-(j-1)+1$ 是新的 i

BBC ABCDAB ABCDABCDABDE
ABCDABD

BBC ABCDAB ABCDABCDABDE
ABCDABD

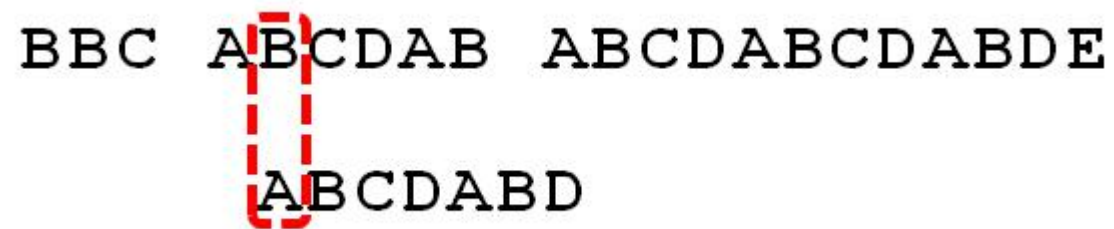
BBC ABCDAB ABCDABCDABDE
ABCDABD

BBC ABCDAB ABCDABCDABDE
ABCDABD

BBC ABCDAB ABCDABCDABDE
ABCDABD



BBC ABCDAB ABCDABCDABDE
ABCDABD



BBC ABCDAB ABCDABCDABDE
ABCDABD



BBC ABCDAB ABCDABCDABDE
ABCDABD



4.3.2 模式匹配的一种改进算法

假设现在文本串 S 匹配到 i 位置，模式串 P 匹配到 j 位置

如果 $j = -1$ ，或者当前字符匹配成功（即 $S[i] == P[j]$ ），都令 $i++$ ， $j++$ ，继续匹配下一个字符；

如果 $j \neq -1$ ，且当前字符匹配失败（即 $S[i] \neq P[j]$ ），则令 i 不变， $j = \text{next}[j]$ 。此举意味着失配时，模式串 P 相对于文本串 S 向右移动了 $j - \text{next}[j]$ 位。

4.3.2 模式匹配的一种改进算法

Int Index_KMP(**char*** s, **char*** p)

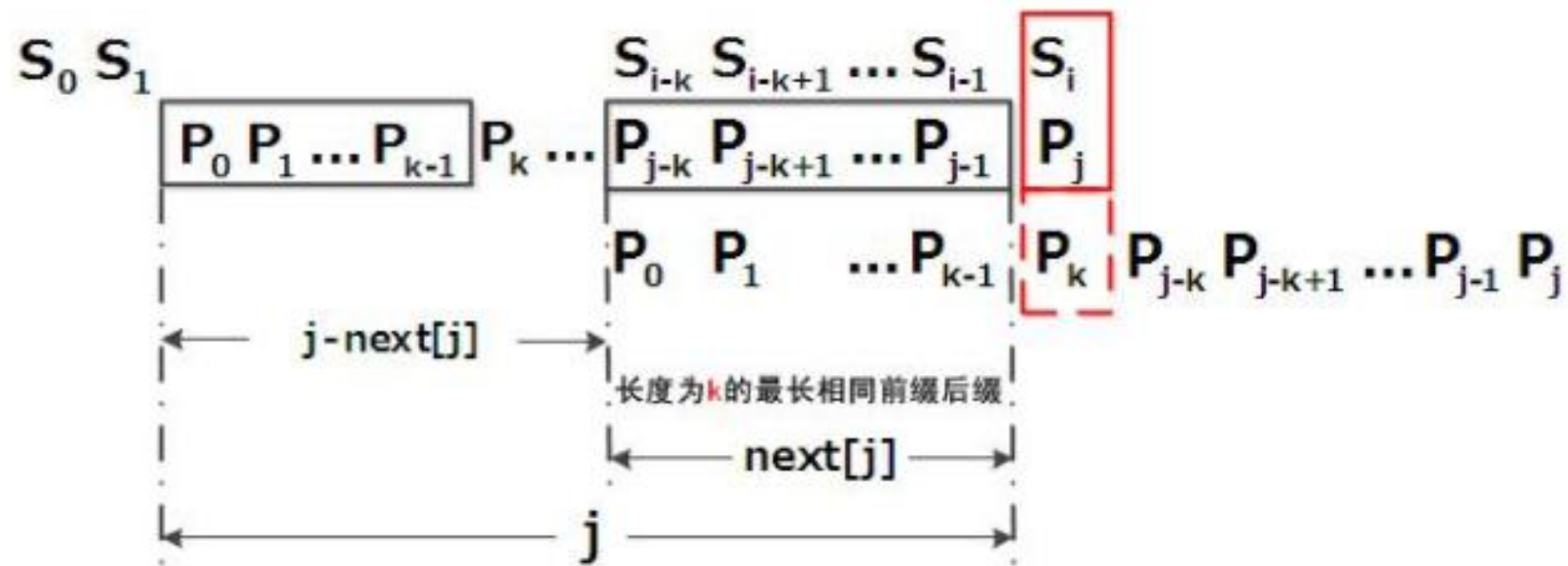
```
{ int i = 0;    int j = 0;    int sLen = strlen(s);    int pLen = strlen(p);  
    while (i < sLen && j < pLen)  
    { //①如果j = -1, 或者当前字符匹配成功 (即S[i] == P[j]) , 都令i++, j++  
        if (j == -1 || s[i] == p[j])  
        {  
            i++;  
            j++;  
        }  
        else  
        {  
            //②如果j != -1, 且当前字符匹配失败 (即S[i] != P[j]) , 则令 i 不变, j = next[j]  
            j = next[j];  
        }  
    }  
    if (j == pLen)  
        return i - j;  
    else  
        return -1;
```

BBC ABCDAB ABCDABCDABDE
ABCDABD

A red dashed rectangular box highlights the overlapping portion of the second and third strings. The box starts at the beginning of the second string 'ABCDAB' and extends to the end of the third string 'ABCDABD', specifically enclosing the characters 'ABCDAB'.

BBC ABCDAB ABCDABCDABDE
ABCDABD

A red dashed rectangular box highlights the overlapping portion of the second and third strings. The box starts at the beginning of the second string 'ABCDAB' and extends to the end of the third string 'ABCDABD', specifically enclosing the characters 'ABCDAB'.



①寻找前缀后缀最长公共元素长度

对于 $P = p_0 p_1 \dots p_{j-1} p_j$ ，寻找模式串 P 中长度最大且相等的前缀和后缀。如果存在 $p_0 p_1 \dots p_{k-1} p_k = p_{j-k} p_{j-k+1} \dots p_{j-1} p_j$ ，那么在包含 p_j 的模式串中有最大长度为 $k+1$ 的相同前缀后缀。举个例子，如果给定的模式串为“abab”，那么它的各个子串的前缀后缀的公共元素的最大长度如下表格所示：

模式串	a	b	a	b
最大前缀后缀公共元素长度	0	0	1	2

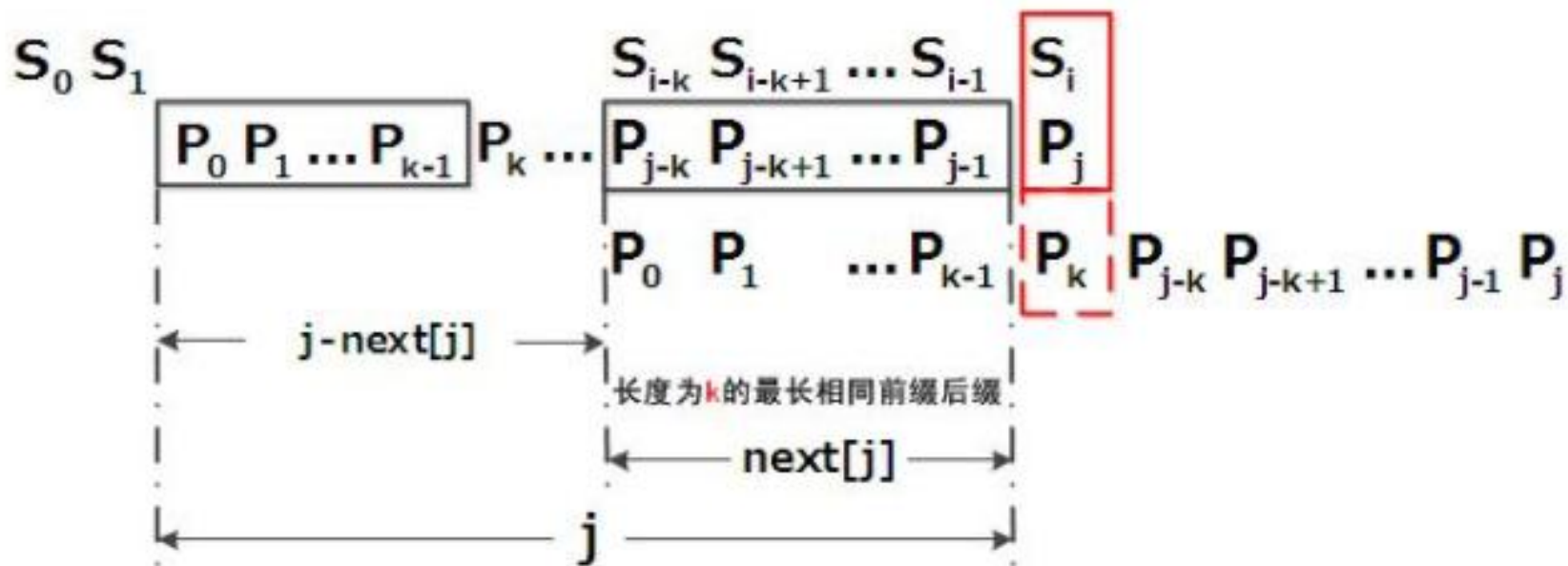
②求next数组

next 数组考虑的是除当前字符外的最长相同前缀后缀，所以通过第①步骤求得各个前缀后缀的公共元素的最大长度后，只要稍作变形即可：将第①步骤中求得的值整体右移一位，然后初值赋为-1，如下表格所示：

模式串	a	b	a	b
next数组	-1	0	0	1

③根据next数组进行匹配

匹配失败, $j = \text{next}[j]$, 模式串向右移动的位数为: $j - \text{next}[j]$ 。换言之, 当模式串的后缀 $p_{j-k} p_{j-k+1}, \dots, p_{j-1}$ 跟文本串 $s_{i-k} s_{i-k+1}, \dots, s_{i-1}$ 匹配成功, 但 p_j 跟 s_i 匹配失败时, 因为 $\text{next}[j] = k$, 相当于在不包含 p_j 的模式串中有最大长度为 k 的相同前缀后缀, 即 $p_0 p_1 \dots p_{k-1} = p_{j-k} p_{j-k+1} \dots p_{j-1}$, 故令 $j = \text{next}[j]$, 从而让模式串右移 $j - \text{next}[j]$ 位, 使得模式串的前缀 $p_0 p_1, \dots, p_{k-1}$ 对应着文本串 $s_{i-k} s_{i-k+1}, \dots, s_{i-1}$, 而后让 p_k 跟 s_i 继续匹配。如下图所示:



通过代码递推计算next 数组

计算next 数组的方法可以采用递推:

1. 如果对于值 k , 已有 $p_0 p_1, \dots, p_{k-1} = p_{j-k} p_{j-k+1}, \dots, p_{j-1}$, 相当于 $\text{next}[j] = k$ 。

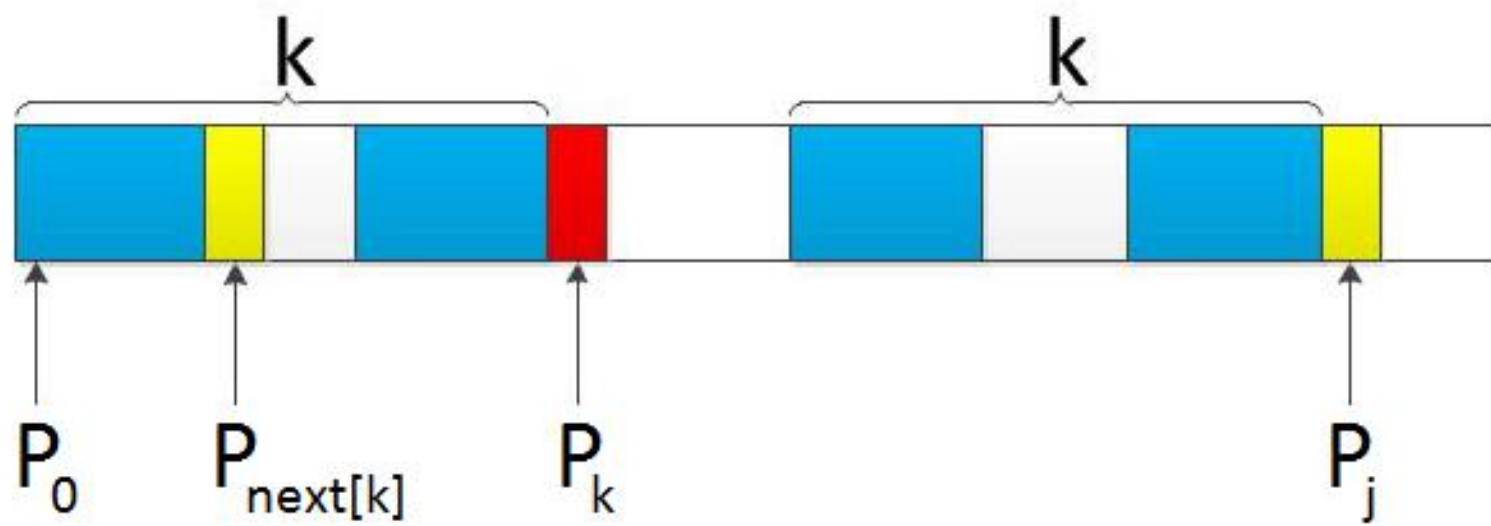
此意味着什么呢? 究其本质, $\text{next}[j] = k$ 代表 $p[j]$ 之前的模式串子串中, 有长度为 k 的相同前缀和后缀。有了这个next 数组, 在KMP匹配中, 当模式串中 j 处的字符失配时, 下一步用 $\text{next}[j]$ 处的字符继续跟文本串匹配, 相当于模式串向右移动 $j - \text{next}[j]$ 位。

2. 下面的问题是: 已知 $\text{next}[0, \dots, j]$, 如何求出 $\text{next}[j+1]$ 呢?

对于P的前 $j+1$ 个序列字符:

若 $p[k] == p[j]$, 则 $\text{next}[j+1] = \text{next}[j] + 1 = k + 1$;

若 $p[k] \neq p[j]$, 如果此时 $p[\text{next}[k]] == p[j]$, 则 $\text{next}[j+1] = \text{next}[k] + 1$, 否则继续递归前缀索引 $k = \text{next}[k]$, 而后重复此过程。相当于在字符 $p[j+1]$ 之前不存在长度为 $k+1$ 的前缀" $p_0 p_1, \dots, p_{k-1} p_k$ "跟后缀" $p_{j-k} p_{j-k+1}, \dots, p_{j-1} p_j$ "相等, 那么是否可能存在另一个值 $t+1 < k+1$, 使得长度更小的前缀" $p_0 p_1, \dots, p_{t-1} p_t$ "等于长度更小的后缀" $p_{j-t} p_{j-t+1}, \dots, p_{j-1} p_j$ "呢? 如果存在, 那么这个 $t+1$ 便是 $\text{next}[j+1]$ 的值, 此相当于利用已经求得的next 数组 ($\text{next}[0, \dots, k, \dots, j]$) 进行P串前缀跟P串后缀的匹配。



```
void get_next(char* p,int next[ ])
{
    int pLen = strlen(p);    next[0] = -1;
    int k = -1;    int j = 0;
    while (j < pLen - 1)
    {
        //p[k]表示前缀, p[j]表示后缀
        if (k == -1 || p[j] == p[k])
        {
            ++k;
            ++j;
            next[j] = k;
        }
        else
        {
            k = next[k];
        }
    }
}
```

通过代码递推计算next 数组

假定给定模式串ABCDABCE，且已知 $\text{next}[j] = k$ （相当于“ $p_0 p_{k-1}$ ” = “ $p_{j-k} p_{j-1}$ ” = AB，可以看出k为2），现要求 $\text{next}[j+1]$ 等于多少？因为 $p_k = p_j = C$ ，所以 $\text{next}[j+1] = \text{next}[j] + 1 = k + 1$ （可以看出 $\text{next}[j+1] = 3$ ）。代表字符E前的模式串中，有长度 $k+1$ 的相同前缀后缀。

模式串	A	B	C	D	A	B	C	E
前后缀 相同长度	0	0	0	0	1	2	3	0
next 值	-1	0	0	0	0	1	2	?
索引	p_0	p_{k-1}	p_k	p_{k+1}	p_{j-k}	p_{j-1}	p_j	p_{j+1}



通过代码递推计算next 数组

但如果 $p_k \neq p_j$ 呢？说明 “ $p_0 p_{k-1} p_k$ ” \neq “ $p_{j-k} p_{j-1} p_j$ ”。换言之，当 $p_k \neq p_j$ 后，字符E前有多大长度的相同前缀后缀呢？很明显，因为C不同于D，所以ABC跟 ABD不相同，即字符E前的模式串没有长度为 $k+1$ 的相同前缀后缀，也就不能再简单的令： $next[j + 1] = next[j] + 1$ 。所以，只能去寻找长度更短一点的相同前缀后缀。

模式串	A	B	<u>C</u>	D	A	B	<u>D</u>	E
前后缀相同长度	0	0	0	0	1	2	0	0
next 值	-1	0	0	0	0	1	2	?
索引	p_0	p_{k-1}	p_k	p_{k+1}	p_{j-k}	p_{j-1}	p_j	p_{j+1}

若能在前缀 “ $p_0 p_{k-1} p_k$ ” 中不断的递归前缀索引 $k = next[k]$ ，找到一个字符 $p_{k'}$ 也为D，代表 $p_{k'} = p_j$ ，且满足 $p_0 p_{k'-1} p_{k'} = p_{j-k'} p_{j-1} p_j$ ，则最大相同的前缀后缀长度为 $k' + 1$ ，从而 $next[j + 1] = k' + 1 = next[k'] + 1$ 。否则前缀中没有D，则代表没有相同的前缀后缀， $next[j + 1] = 0$ 。

模式串	<u>D</u>	A	B	C	D	A	B	<u>D</u>	E
最长相同 前缀后缀	0	0	0	0	1	2	3	?	
next 值	-1	0	0	0	0	1	2	3	?
索引	p_0	p_1	p_{k-1}	p_k	p_{j-k}	p_{j-2}	p_{j-1}	p_j	p_{j+1}

模式串	A	B	C	D	A	B	D
k	-1	0	-1,0	-1,0	-1,0	1	2
j	0	1	2	3	4	5	6
next 数组	-1	0	0	0	0	1	2