

Lecture 13: 串

13.1 串的基本概念

13.1.1 基本概念

串的定义

- 概念：
 - 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 $S = a_1a_2 \dots a_n$
 - 其中，S 是串名，其整体表示串的值
 - a_i 可以是字母，数字或其他字符串，串中字符的个数称为串的长度；
 - $n = 0$ 时的串称为**空串**(用 $\emptyset$ 表示)
- 举例： `S="HelloWorld"`
 - 注意：有的地方用双引号 (如 Java, C)，有的地方用单引号 (Python)
 - 注意：空格也是一种字符
- 其它概念
 - **子串**：串中任意个连续的字符组成的子序列。
 - **主串**：包含子串的串。
 - **字符在主串中的位置**：字符在串中的序号。(字符第一次出现)
 - **子串在主串中的位置**：子串的第一个字符在主串中的位置。
- 注意：
 - 1. 位序：从 1 开始而不是从 0 开始；
 - 2. 子串在主串中的位置：子串的**第一个字符**在主串中的位置；

串的结构

- 串是一种**特殊的线性表**，数据元素之间呈线性关系；
- 区别
 - 1. 串的数据结构当中的数据对象**限定为字符集**(如中文字符，英文字符，数字字符，标点字符等)；
 - 2. 串的基本操作(如增删改查等)通常**以子串为操作对象**；

1.3.2 串的基本操作定义

假设：当前串为 `T="", S=iPhone 11 Pro Max?", W="Pro"`

串的基本操作

- `StrAssign(&T, chars)`：**赋值**操作。把串赋值为 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)** : **串链接**。用 T 返回由 S1 和 S2 链接而成的新串。
- **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。
 - 1. 比较规则: 从第一个字符开始往后依次对比, 先出现更大字符的串就更大;
 - 比如"ab"和"ac"相比, 因为"c"比"b"更大, 所以"ac">"ab"
 - 长串的前缀与短串相同时, 长串更大
 - 只有两个串完全相同时, 才相等
 - 2. 若 S>T, 则返回值>0。若 S=T, 则返回值=0。若 S<T, 则返回值<0。

字符集

- **英文字符**: ASCII 字符集

- 图示:

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				
十进制	字符	ctrl	代码	字符解释	十进制	字符	ctrl	代码	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	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 EOF	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	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}
1110	E	14	🎵	^N SO	移出	30	▲	^_ 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

- **中英文**: Unicode 字符集

- 注意:

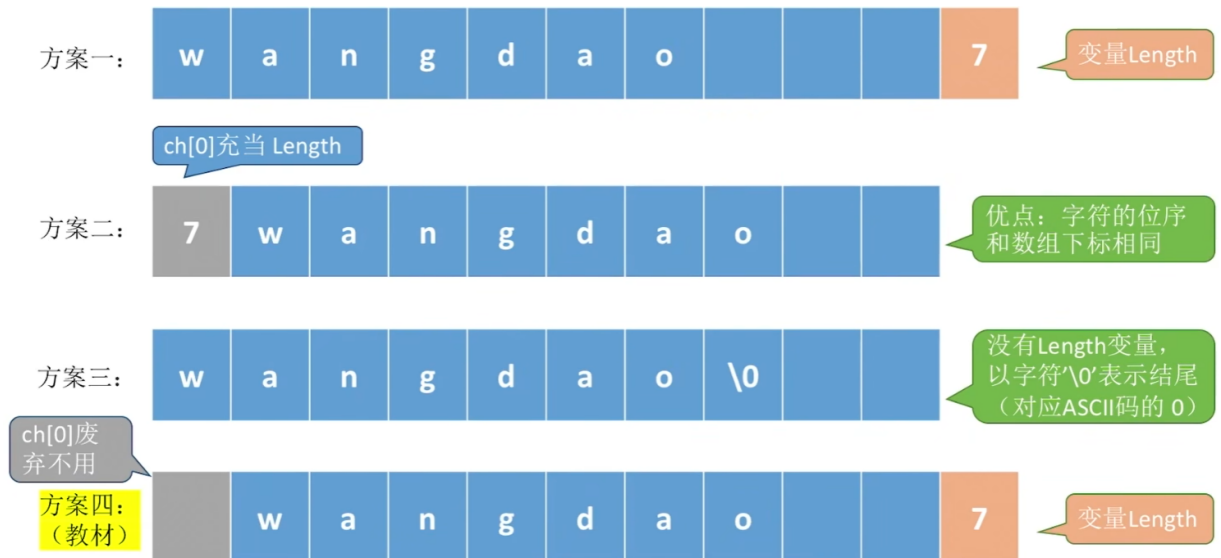
- 1. 采用不同的编码方式, 每个字符所占空间不同, 考研中只需默认每个字符占 1 B 即可;
- 2. 基于同一个字符集可以有多种编码方案, 如: UTF-8, UTF-16

13.2 串的存储结构

13.2.1 串的顺序存储

基本概念

- 静态数组实现：
 - 大小：每个 char 字符占 1B 大小，然后还需要一个位置来存储当前串的实际长度；
- 四种方案：
 - 方案一：把 Length 放到数组最后；
 - 方案二：用 char[0] 充当 Length；
 - 缺点：因为 char 的大小为 char，所以将 Length 的长度限制为 0-25
 - 方案三：以 \0 结尾表示字符串结束；
 - 方案四：没有 Length 变量：以字符'^0'表示结尾(对应 ASCII 码的 0)；
- 图示：



代码定义：静态数组实现

```
//静态数组实现(定长顺序存储)
#define MAXLEM 255 //预定义最大串长255
typedef struct{
    char ch[MAXLEM]; //每个分量存储一个字符
    int length;      //串的实际长度
}SString;
//自动回收
```

代码定义：动态存储

```
//动态存储
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;

//声明空间S.ch=(char *)malloc(MAXLEM* sizeof(char));
//注意：手动free
```

13.2.2 串的链式存储

代码定义：一般存储方法

```
//一般
//存储密度低：每个字符1B，每个指针4B
typedef struct StringNode{
    char ch;    //每个结点存1个字符
    struct StringNode* next;
}StringNode,*String;
```

代码定义：改进存储方法

- 每个 Node 存储多个字符；
- 因为 struct 在系统中就需要站 4 个字节，但一个 char 只要 1 字节；

```
//改进
//存储密度高(如果有空位，则可以用特殊字符填充)
typedef struct StringNode{
    char ch[4];    //每个结点存1个字符
    struct StringNode* next;
}StringNode,*String;
```

13.2.3 串的基本操作

求子字符串：SubString

- 传入空 Sub，在此返回内容；
- 防止越界 `pos+len-1 > S.length`；

```
//求子串
bool SubString(SSrting &Sub,SSrting S,int pos,int len){
    //子串范围越界
    if(pos+len-1 > S.length)
        return false;
    for(int i=pos;i<pos+len;i++){
        Sub.ch[i-pos+1] = S.ch[i];
    }
    Sub.length = len;
    return true;
}
```

比较操作：StrCompare

- 若 $S > T$ ，则返回值 > 0 。若 $S = T$ ，则返回值 $= 0$ 。若 $S < T$ ，则返回值 < 0

```
//比较操作
int StrCompare(SSString S,SSrting 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;
```

```
}
```

定位操作: Index

- 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。

```
//出现返回位置，否则为0
```

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

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

```
    SString sub; //用于暂存子串
```

```
    //从头到尾，依次比较子串
```

```
    while(i<=n-m+1){ //最多对比n-m+1个子串
```

```
        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;
```

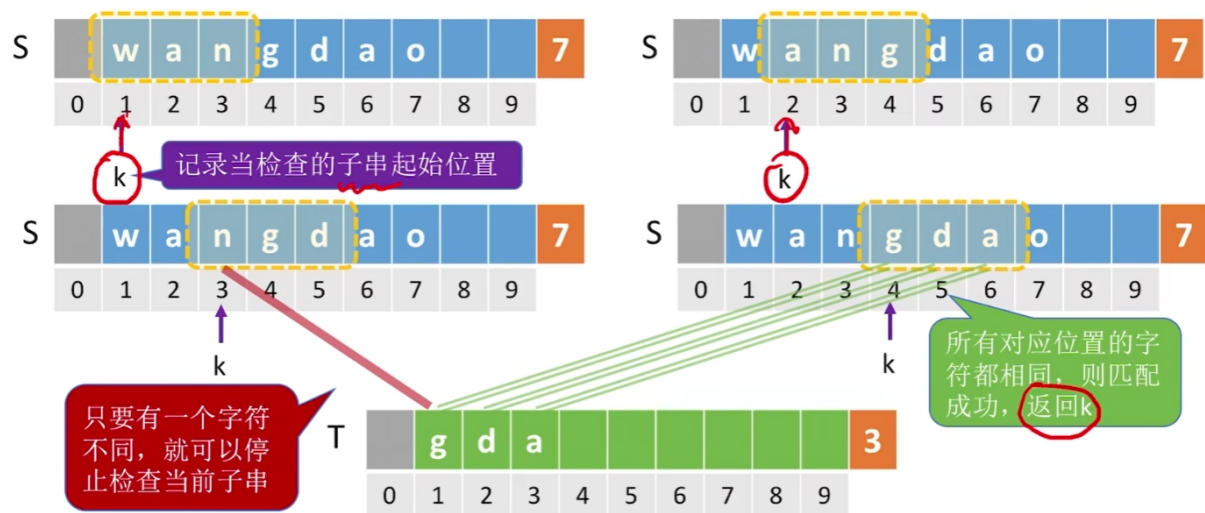
```
}
```

13.3 串的朴素模式匹配算法

13.3.1 串的模式匹配

串的模式匹配

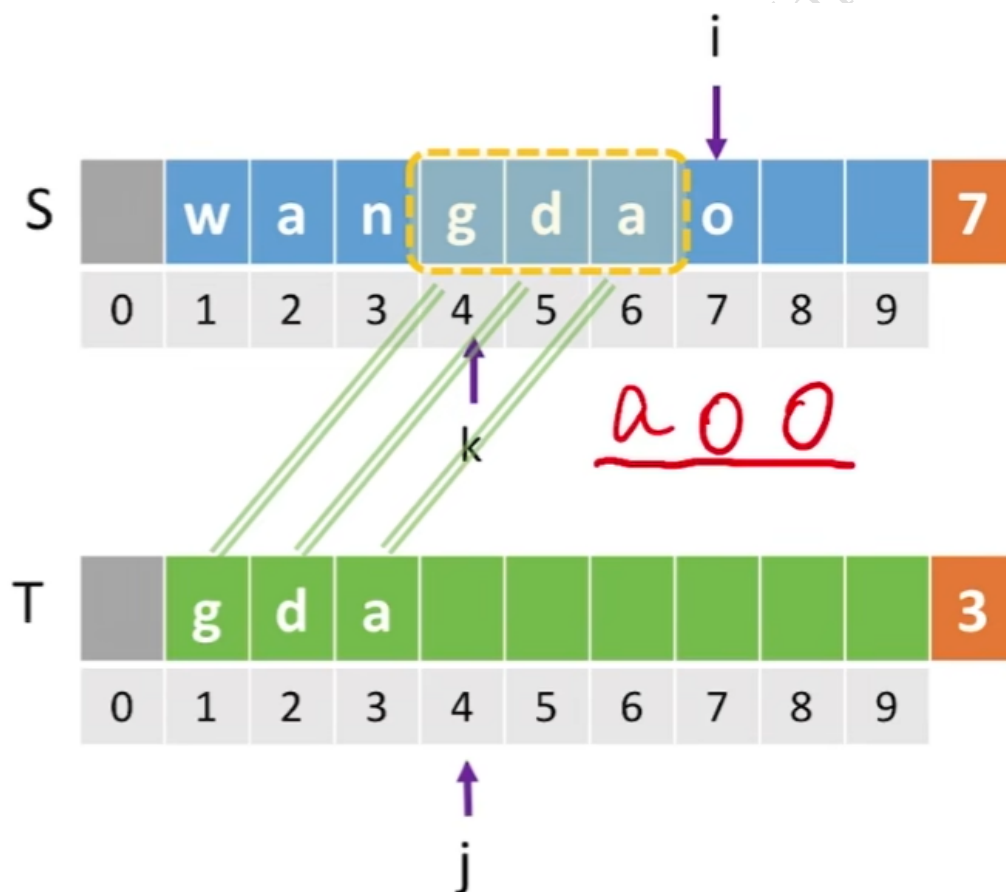
- 概念:
 - 在主串中找到与模式串相同的子串，并返回所在位置。
- 模式串:
 - 想在主串中找到的目标串，其未必能在主串中找到；
- 定位操作:
 - 也称为模式匹配；
 - `Index(S, T)`: 若主串 S 中存在与串 T 值相同的子串，则返回它的主串 S 中第一次出现的位置，否则函数值为 0；
- 图示:



13.3.2 代码实现

实现

- 图示：



朴素模式匹配算法

- 过程：
 - 取出从位置 i 开始，长度为 m 的子串；
 - 子串和模式串对比，若不匹配，则匹配下一个子串；
 - 匹配下一子串：主串指针 i 指向下一个子串的第一个位置，模式串指针 j 回到模式串的第一个位置
- 注意：

- 最多对比 $n-m+1$ 个子串

```
//朴素模式匹配算法
int Index(SSString S,SSString T){
    int k=1;          //记录当前子串开始位置
    int i=k,j=1;
    while(i<S.length && j<=T.length){
        if(S.ch[i] == T.ch[j]){
            ++i;
            ++j;    //继续比较后续字符
        }else{
            k++;    //检查下一个子串
            i=k;
            j=1;
        }
    }
    if(j>T.length)
        return k;
    else
        return 0;
}
```

时间复杂度

- 设主串长度为 n ，模式串长度为 m ，则：最坏时间复杂度 = $O(nm)$
 - 推导：最坏的情况，共 $n-m+1$ 个子串，复杂度= 每个子串都要对比 m 个字符= $O((n-m+1)m) = O(nm)$