第四章 串

串也叫字符串，它是由零个或多个字符组成的的字符序列。

基本内容

1　串的有关概念　串的基本操作
2　串的顺序存储结构，堆分配存储结构,链式存储结构；
3　串的基本操作算法；
4　串的模式匹配算法；

学习要点

1　了解串的基本操作，了解利用这些基本操作实现串的其它操作的方法；
2　掌握在串的堆分配存储结构下，串的基本操作算法；
3　掌握在串的模式匹配算法；

第四章　串

4. 1 串的基本概念

1 什么是串

串是一种特殊的线性表，它是由零个或多个字符组成的有限序列，一般记作 $s = “a_1,a_2, a_3, \ldots a_n”$

其中 s----串名， $a_1,a_2, a_3, \ldots a_n$----串值

串的应用非常广泛，许多高级语言中都把串的作为基本数据类型。在事务处理程序中，顾客的姓名、地址货物的名称、产地可作为字符串处理，文本文件中的每一行字符等也可作为字符串处理。

下面是一些串的例子：

（1）a = “This is a string”

（2）b = “string”

（3）c = “ “

（4）d = “”

（5）e = “你好”

说明：

1) 串中包含的字符个数，称为串的长度。 长度为0的串称为空串，它不包括任何字符;

2) 串中所包含的字符可以是字母、数字或其他字符，这依赖于具体计算机所允许的字符集。

2 串的有关术语

1）子串

串中任意连续的字符组成的子序列称为该串的子串;

例：c = “ DATA STRUCTURE”，f=“DATA” f是c的子串

2）子串的位置

子串t 在主串S中的位置是指主串s 中第一个与T相同的子串的首字母在主串中的位置;

例：s=“ababcabcac” , t=“abc” 子串T 在主串S中的位置为3

3）串相等

两个串相等，当且仅当两个串长度相同，并且各个对应位置的字符都相同;

3 串的基本操作

串的逻辑结构与线性表一样，都是线性结构。但由于串的应用与线性表不同，串的基本操作与线性表有很大差别。

1）串赋值操作assign(s, t) 功能：将串t 的值赋给串变量s ；

2）串相等判断 equal(s,t) 函数

3) 串的联接操作 concat(s , t) 把串t 接在串 s 后面

4) 求串长操作 lenght(s)

5) 求子串操作sub (s, start, len, t) 若

0<=start<length(s) 0<=len<=length(s)-start

则t 中值为从串s 的第start个字符, 起长度为len 的字符序列, 并且函数返回值为1, 否则为0;

6）求子串位置操作index(s, t)

功能：如果s中存在与 t 相同的子串, 则返回s中第1个这样的子串的位置，若不存在返回0

7） 替换操作 replace(s, t ,v)

功能：由串v 替换串s 中出现的所有和 t 相同的不重叠子串；

8）复制串操作 strcopy(s, t)

功能：由串变量 s 复制得到串变量 t ；

9）判空操作 empty(s)

功能：若为空串，则返回1，否则返回0

10）串置空操作 ClearString(s)

功能：将s 清为空串

11) 串插入操作 StrInsert(s, start , t)

功能：将串t 插入到串s 的第start 字符之前

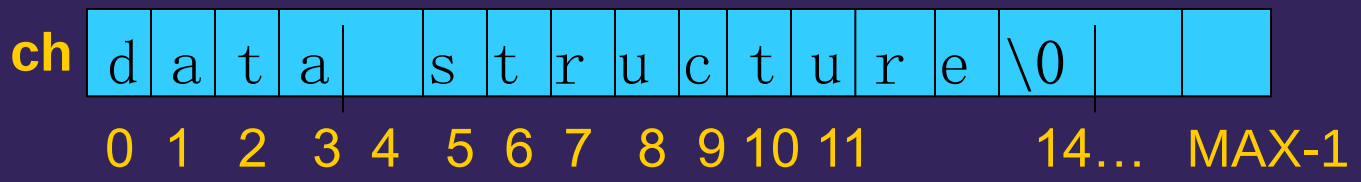
12）串删除操作 StrDelete(s, start , len)

功能：从串s 中删除第 start 个字符起长度len 为的子串

1 顺序存储结构

顺序存储结构类似于C语言的字符数组，以一组地址连续的存储单元存放串值字符序列，其类型说明如下：

```
#define MAX  255
char ch[MAX]
```

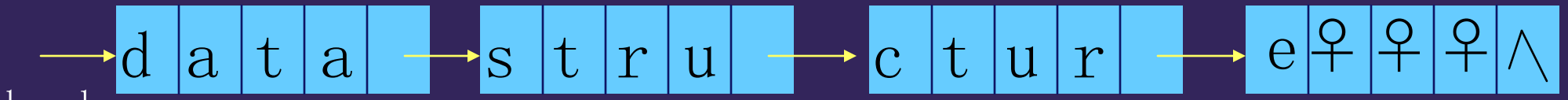


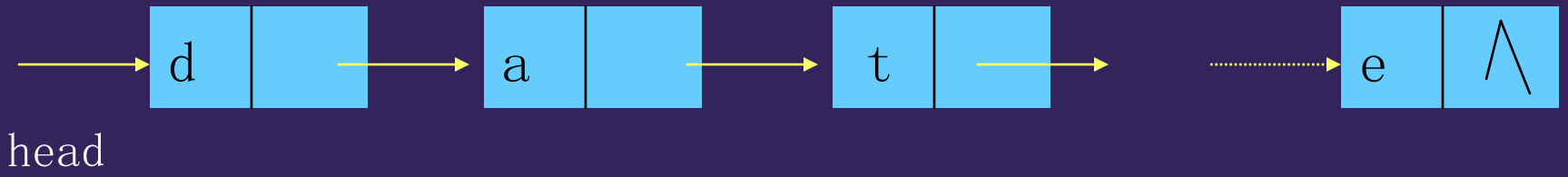
在数组ch中以字符‘\0’表示字符串的结束. 特点是访问容易，但删除或插入麻烦

2 链式存储结构

链式存储结构类似线性链表，由于串结构的特殊性,要考虑每个结点是存放一个字符还是多个字符。一个字符的，插入、删除、求长度非常方便，但存储效率低。

多个字符的，改善了效率，在处理不定长的大字符串时很有效，但插入、删除不方便，可用特殊符号来填满未充分利用的结点。



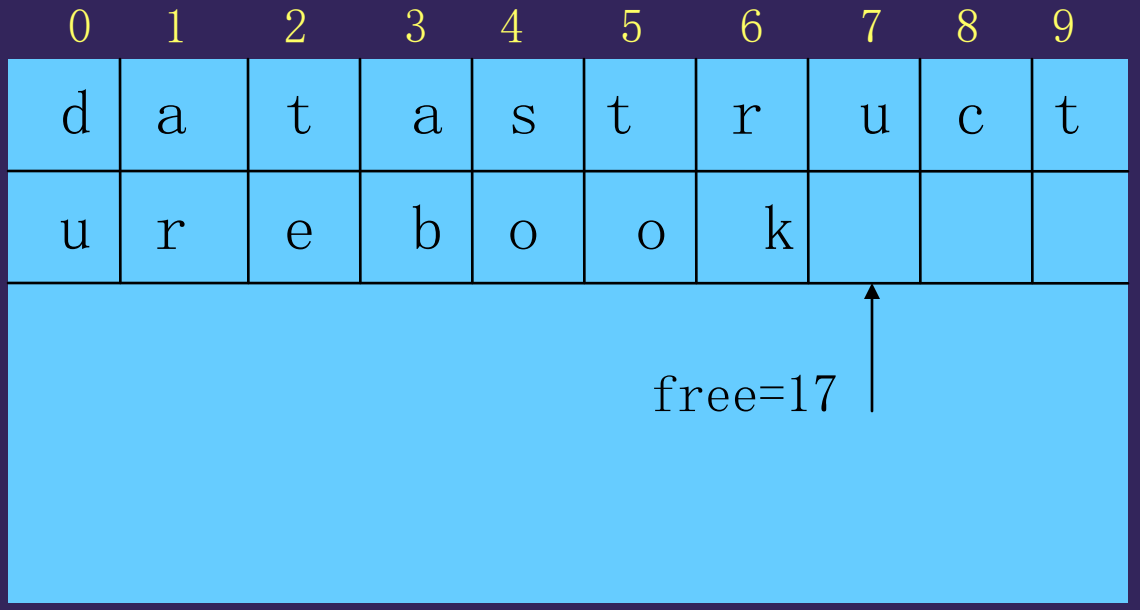


3、堆分配存储

堆分配存储类似于线性表的顺序存储结构，以一组地址连续的存储单元存放串值字符序列，其存储空间是在程序执行过程中动态分配的。一个串值的确定是通过串在堆中的起始位置和串的长度实现的。为此，串名与串值之间要建立一个对照表。

```
char store[MAX];
int   free ; */ 整型域：存放串长*/
```

串名	起址	串长
a	0	4
b	4	9
c	13	4
….		



```
#define MAX 100
Char t[MAX], s[MAX];
```

（1） 求长度 length(s)

```
int length(char s[ ])
 { int i;
     for (i=0, s[ i]!='\0';i++);
     return(i);
 }
```

(2) 求子串算法

```
int sub(char s[ ], int start, int  len , char t[ ])
  {   int n,i;
      n=length(s);
        if (start <0 || start>=n )return (0);
        if ( len<0 || len+start>n)  return (0);  //参数不合法
         for (i=0;i<len ;i++)
           t[i]=s[start+i];
           t[i]='\0';
           return (1);
   }
```

(3) 串连接算法

```
int concat( char s[ ], char t[ ])
{ int m, n, i;
   m= length(s);
   n=length(t);
   if (m+n>=MAX) return(0);
   for(i=0;i<n;i++) s[m+i]=t[i];
    t[i]='\0';
    return (1);
 }
```

(4) 判断串相等算法

```
int equal( char s[ ], char t[ ])
{  int m, n, i;
   m= length(s);
   n=length(t);
   if (m!=n) return(0);
   for(i=0;i<m;i++) if (s[i]!=t[i])return(0);
   return (1);
}
```

4.4 串的模式匹配算法

(1) 求子串位置的定位函数

子串的定位操作index(s, t, start) 通常称作串的模式匹配（其中t 被称为模式串），是各种串处理系统中最重要的操作之一。

s=“abcdef”, t=“cde” index(s, t, 0)=2, index(s, t, 1)=2,

t=“ab” index(s, t, 0)=0

t=“ad” index(s, t, 0)=-1

(2) BF 算法

该算法的基本思想是，在主串 s 中从第 i (i 的 初值为 start)个字符起，并且长度和 t 串相等的子串和 t 比较，若相等，则求得函数值为 i ，否则 i 增1，直至串 s 中不存在从 i 开始和 t 相等的子串为止。

匹配算法1

第一趟匹配 i=0

a b a b c a b c a c b a b

t = a b c a c Subch!=t

第二趟匹配 i=1

a b a b c a b c a c b a b

t = a b c a c Subch!=t

第三趟匹配 i=2

a b a b c a b c a c b a b

t = a b c a c Subch!=t

第四趟匹配 i=3

a b a b c a b c a c b a b

第五趟匹配 i=4

a b a b c a b c a c b a b

第六趟匹配 i=5

a b a b c a b c a c b a b

t = a b c a c

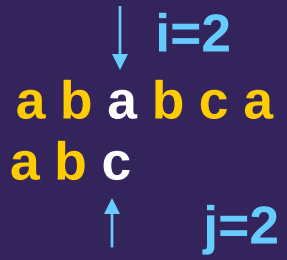
匹配成功

```
int  index( char s[ ], char t[ ],  int start )
 { int  i,eq,m,n;
   char  subch[MAX];
   m=strlen(s); n=strlen(t);
   if  ( start<0||n==0||start+n>m ) return(-1);
   i=start;
   while  (sub(s,i,n,subch))
        {if(equal(t,subch)) break;
          else  i++;eq=sub(s,i,n,subch);}
        if(eq)return(i);
          else return(-1);
   }
```

例　　s=“ababcabcac”　t=“abcac”　index(s,t,0)返回值为5

匹配算法2

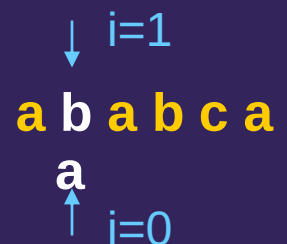
第一趟匹配 i=0



S[2]!=t[2],该趟匹配失败 i=i- j+1,j=0

进入下趟

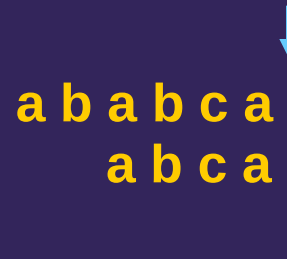
第二趟匹配 i=1



S[1]!=t[0],该趟匹配失败 i=i- j+1,j=0

进入下趟

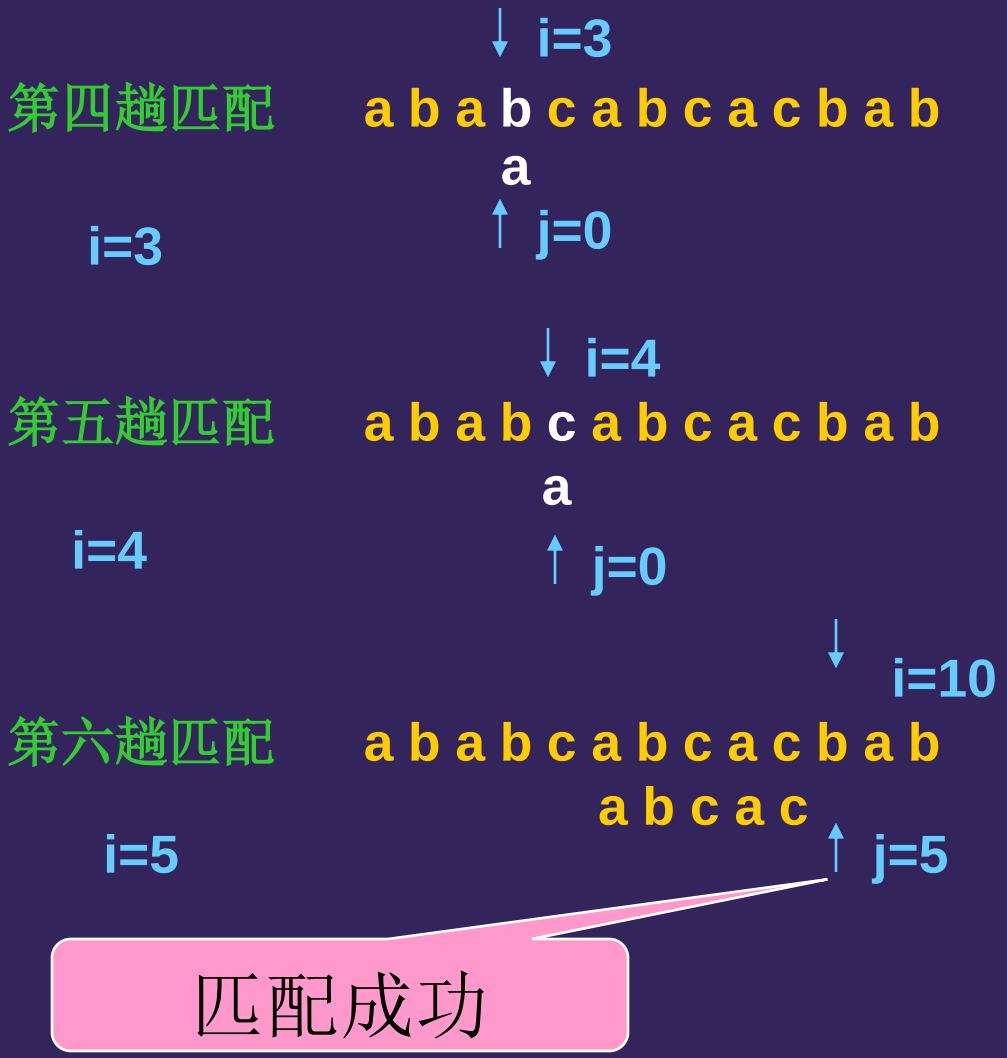
第三趟匹配 i=2



S[6]!=t[4],该趟匹配失败 i=i- j+1,j=0

进入下趟

匹配算法2


i=3
第四趟匹配
a b a b c a b c a c b a b
a
i=3
j=0
S[3]!=t[0],该趟匹配失败 i=i- j+1,j=0
进入下趟
i=4
第五趟匹配
a b a b c a b c a c b a b
a
i=4
j=0
S[4]!=t[0],该趟匹配失败 i=i- j+1,j=0
进入下趟
i=10
第六趟匹配
a b a b c a b c a c b a b
a b c a c
i=5
j=5
j 等于t 的串长, 该趟匹配成功, 返回i=i-n
匹配成功

```
int  index1(char s[ ], char t[ ],  int start )
 { //返回在主串 s 第start个字符后子串t 的位置。若不存在，则函数值为-1。
    int i,  j,  m,  n;
  m=strlen(s); n=strlen;
  if (start<0||n==0||start+n>m)return(-1);
  i=start;j=0;
  while (i<m && j<n)
      if(s[i]=t[j]){ i++; j++;}     //相等,继续比较后继字符
      else{i=i-j+1; j=0;}  //不等,指针i后退(至当前匹配起始位置的
                                   //下一位置）重新开始匹配
    if(j==n) return(i- n);//匹配成功，返回子串T的位置
  else return(-1);
}
```

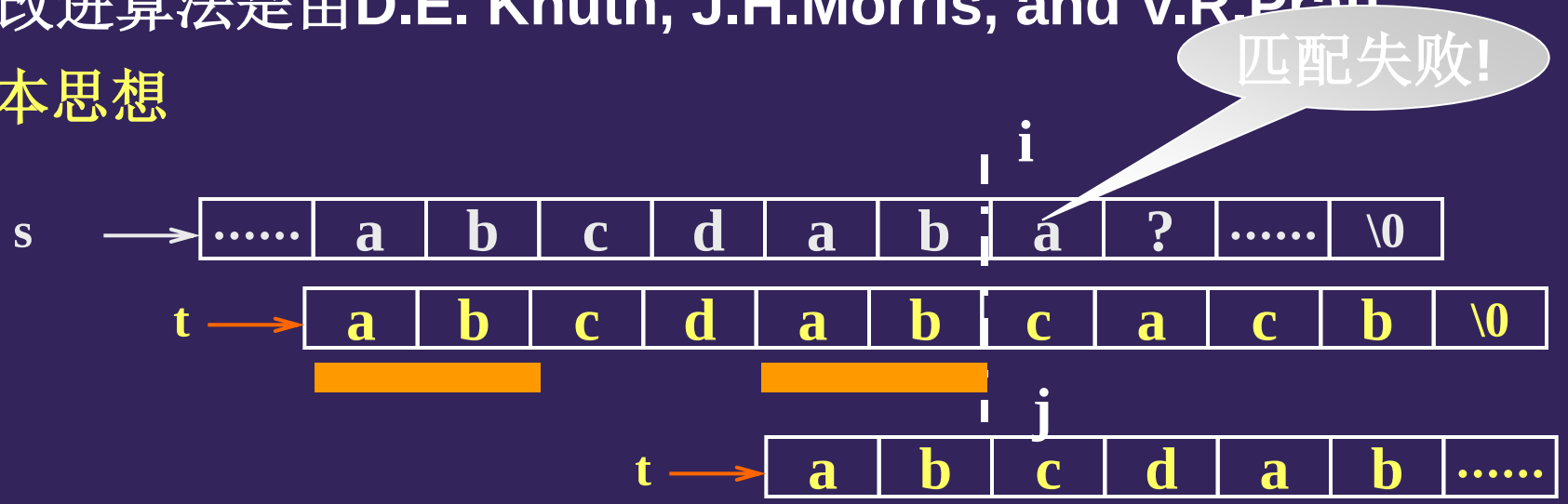
例 s=“ababcabcacbab” t=“abcac” start=0 index1(s,t,start)返回值为5


s = “aaaaaaaaaaaaaaaaba”
t = “aaaaaaaba”

匹配算法3

一种改进算法是由D.E. Knuth, J.H.Morris, and V.R.Pratt

基本思想



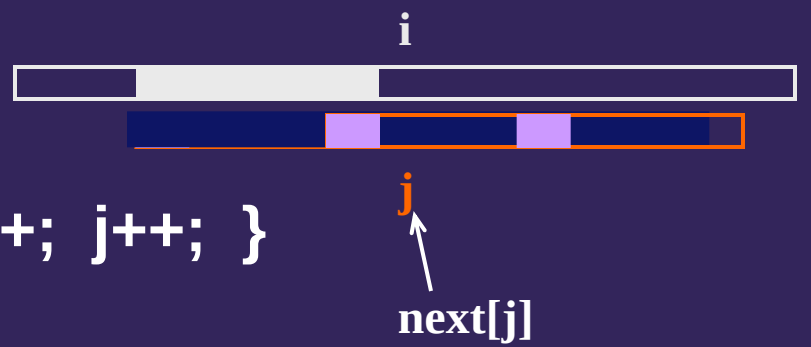
设 t = "$t_0 t_1 \cdots t_n$ 是模式,函数*next( )* 的定义如下:

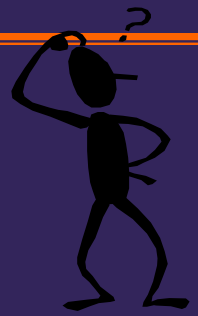
$$next(j)=\begin{cases}-1 & j=0\\ Max\{k \mid \text{largest } 0<k<j \text{ such that } t_0\cdots t_{k-1}=t_{j-k}\cdots t_{j-1}\} & \\ 0 & \text{其它}\end{cases}$$

j	0	1	2	3	4	5	6	7	8	9	10
t	a	b	c	a	b	c	a	c	a	b	\0
next	-1	0	0	0	1	2	3	4	0	1	\

KMP匹配算法

```
int  Index_KMP ( char s[ ],char t[ ], int next[ ] )
{
    int  i = 0 ; j=0, m,n ;
    m=strlen(s); n=strlen(t);
    while ( i <m &&  j <n )
        if (j==-1|| s[ i ]==t[ j ]  )  { i++;  j++;  }
        else   j = next[ j ] ;
       if ( j>=n) return  (  i- n);
         else return(-1);
}
```





怎样计算next [j]？

【Method 1】按定义

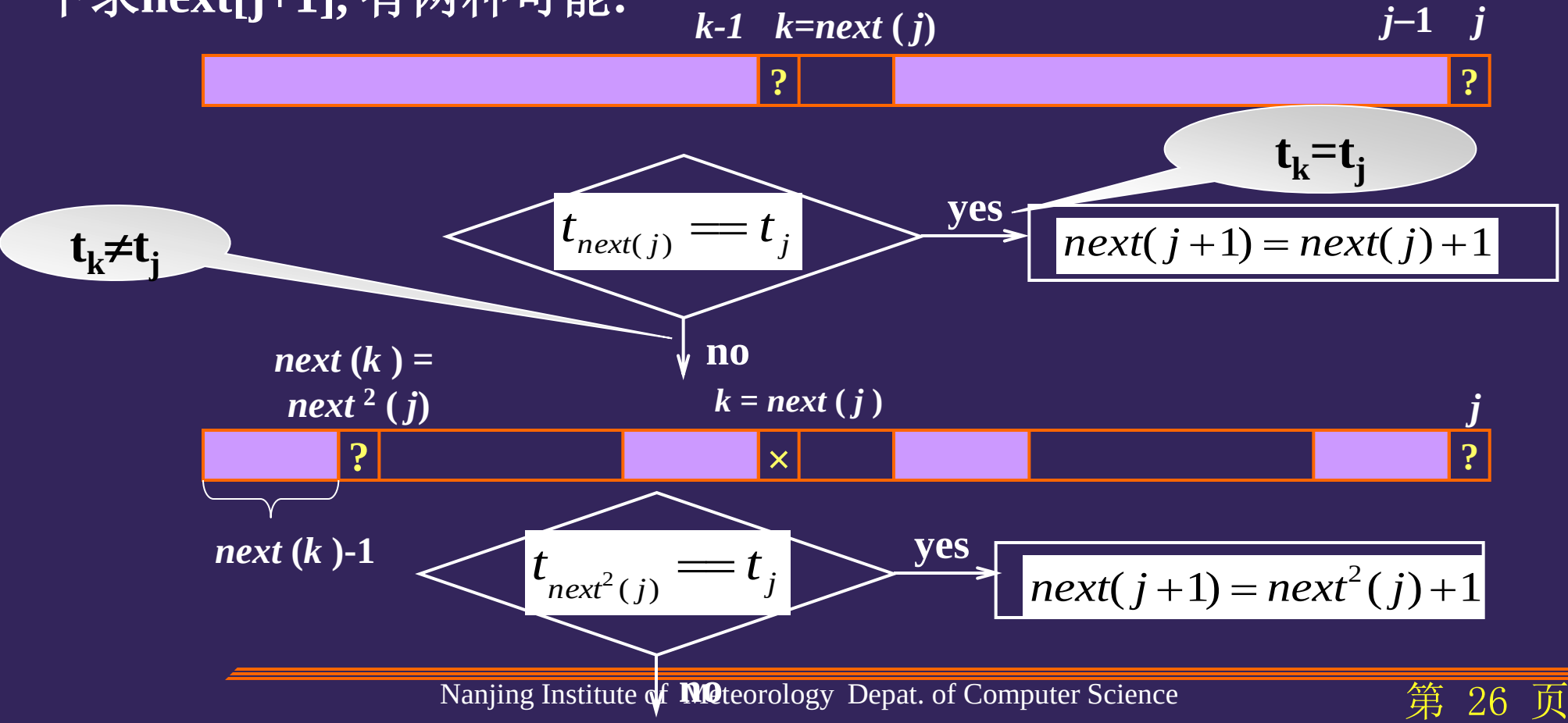
$$next(j)=\begin{cases}-1 & j=0 \\ Max\{k \mid \text{largest } 0<k<j \text{ such that } t_0\cdots t_{k-1}=t_{j-k}\cdots t_{j-1}\} & \text{其它} \\ 0 & \end{cases}$$

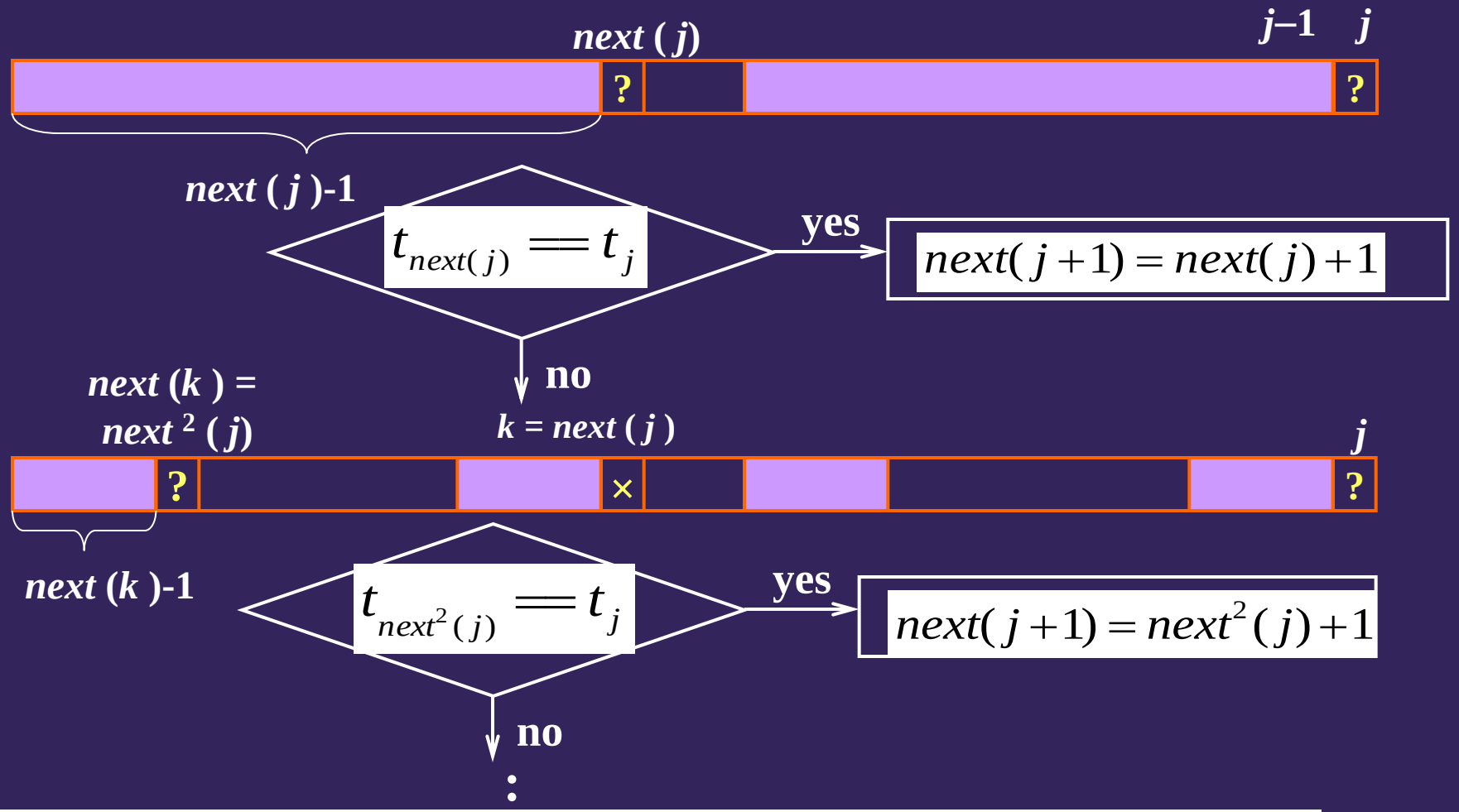
next[0]=-1

设 next[j]=k, 则

“$t_0t_1\cdots t_{k-1}$”=“$t_{j-k}\ t_{j-k+1}\ \cdots\ t_{j-1}$”

下求next[j+1], 有两种可能:





$$next(j+1)=\begin{cases}-1 & \text{if}\quad j=0\\ next^{m}(j)+1 & \text{这里 } m \text{ 是使 } t_{next^{m}(j-1)+1}=t_j \text{ 最小者}\\ 0 & \text{其它}\end{cases}$$

j →	0	1	2	3	4	5	6	7	8	9	10
t →	a	b	c	a	b	c	a	c	a	b	\0
next →	-1	0	0	0	1	2	3	4	0	1	\

```
void  get_next ( char t[ ], int next[ ], int n)
{   int j, k;
    j=0; k=-1; next[0]=-1;
    while(j<n )
       if (k==-1|| t[ j ]==t[ k ])
        { j++;k++; next[ j ]=k;}
       else  k=next [ k ];
    }
```

第四章结束