数组 串 广义表

目 录

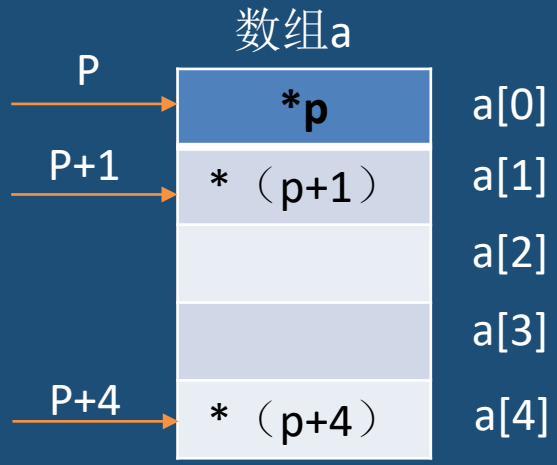
Contents

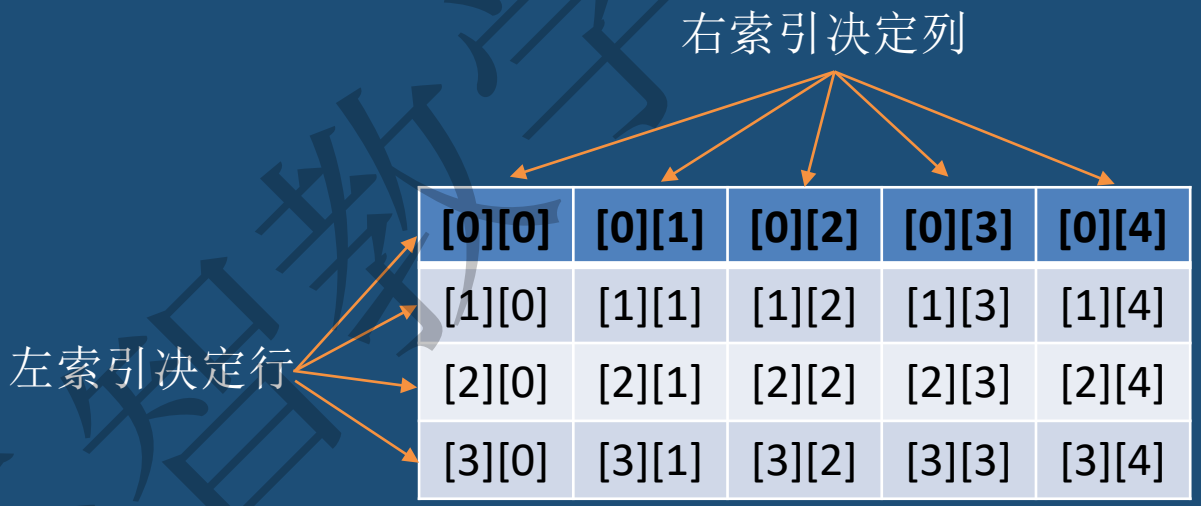
数组

数组是由n个相同类型的数据元素构成的有限序列，每个数据元素称为一个数组元素，每个元素受n个线性关系的约束，每个元素在n个线性关系中的序号称为该元素的下标，并称该数组为n维数组。

数组与线性表的关系：数组是线性表的推广。一维数组可以看做是一个线性表；二维数组可以看做元素是线性表的线性表，以此类推。数组一旦被定义，它的维数和维界就不再改变。因此，除了结构的初始化和销毁之外，数组只会有存取元素和修改元素的操作。

数组

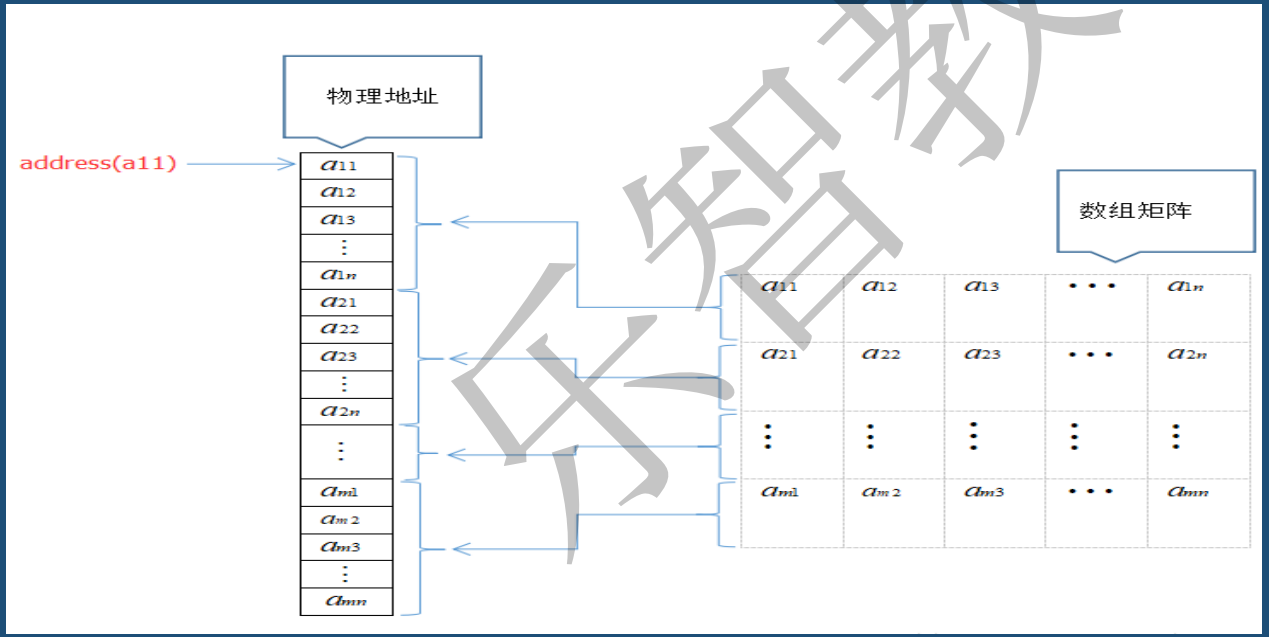




数组的存储结构

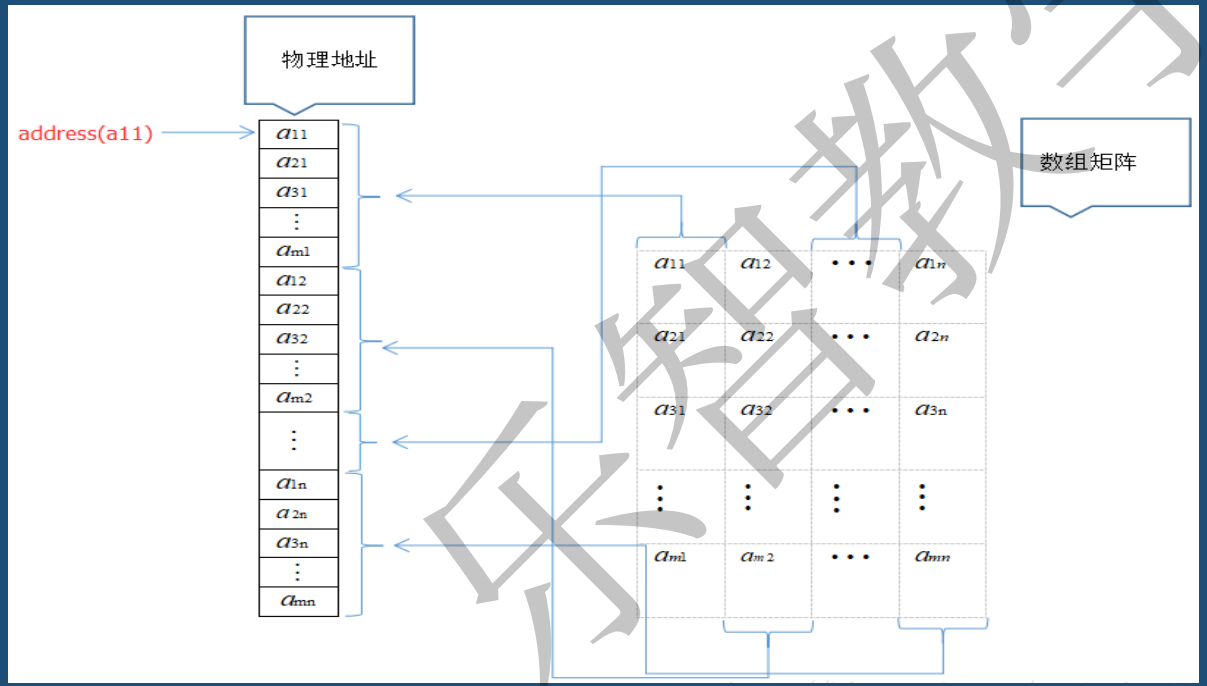
一个数组的所有元素在内存中占用一段连续的存储空间。二维数组具有两种存储方式：

1、以行为主顺序优先存储：



数组的存储结构

2、以列为主顺序优先存储：



数组（真题检测）

1.设有一个二维数组A[10][8],采用以行序为主序的存储方式存放于一个连续的存储空间中，若A[0][0]的存储地址是1000，且每个数组元素占4个字节，求数组元素A[6][4]的内存地址。

	0	1	2	3	4	5	6	7
0	1000	1004	1008	1012	1016	1020	1024	1028
1	1032	…	…	…	…	…	…	…
2	…	…	…	…	…	…	…	…
3	…	…	…	…	…	…	…	…
4	…	…	…	…	…	…	…	…
5	…	…	…	…	…	…	…	…
6	…	…	…	…	…			
7								
8								
9								

分析：从A[0][0]到A[6][4]总共是6*8+5-1=52 个数组单元，又每个数组元素占4个字节，所以一共占52*4=208个字节。又A[0][0]的存储地址是1000，故A[6][4]的内存地址为1000+208=1208

矩阵的压缩存储（对称矩阵）

矩阵的压缩存储：将矩阵的元素按照某种分布规律存储在较小的存储单元中。

1、对称矩阵的压缩存储

n阶对称矩阵：一个n阶的矩阵A中的元素满足a(ij)=a(ji)(0<i,j<n-1)。由于矩阵中的元素是关于主对角线对称的，那么就可以只存储上三角或者下三角矩阵中的元素，这样就可以把原来需要存储n*n个元素压缩至n*(n+1)/2个元素了。使用一维数组s[k]（数组下表从0开始）以行为主存储对称矩阵A(i,j)的下三角元素，那么k和(i,j)之间的对应关系如下：

$$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1 & \text{当i≥j(下三角区和主对角线元素)}\\ \dfrac{j(j-1)}{2}+i-1 & \text{当i<j(上三角区元素}a_{ij}=a_{ji}\text{)}\end{cases}$$

矩阵的压缩存储（三角矩阵）

2、三角矩阵的压缩存储

上三角矩阵：下三角区的所以元素均为同一常量

下三角矩阵：上三角区的所以元素均为同一常量

上下三角矩阵都可以压缩存储在B[n(n+1)/2 +1]中

跟对称矩阵存储相似，只是一位数组s的最后一位用来存储常数项。

等差数列前n项和公式

$$S_n = na_1 + \frac{n(n-1)}{2}d, n \in N^*$$

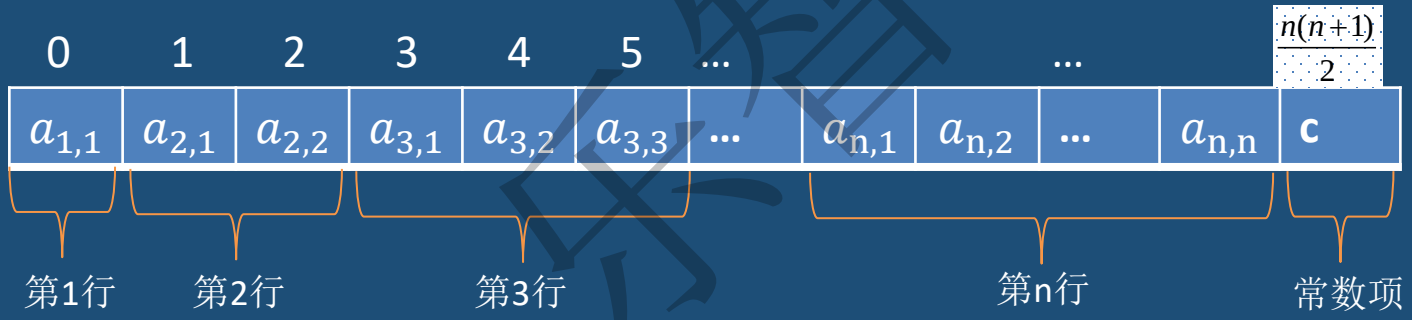
上三角矩阵元素下标的对应关系为：

$$k = \begin{cases} \dfrac{(i-1)(2n-i+2)}{2} + (j-i) & \text{当i≤j(上三角区和主对角线元素)} \\ \dfrac{n(n+1)}{2} & \text{当i>j(下三角区元素)} \end{cases}$$

矩阵的压缩存储（三角矩阵）

下三角矩阵元素下标的对应关系为：

$$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1 & \text{当i≥j(下三角区和主对角线元素)}\\ \dfrac{n(n+1)}{2} & \text{当i<j(上三角区元素)}\end{cases}$$



矩阵的压缩存储（三角矩阵）

3、三对角矩阵的压缩存储

三对角矩阵：所有非零元素集中在主对角线两侧的带状区域内。如图为三对角矩阵。

$$A=\begin{pmatrix} a_{00} & a_{01} & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{43} & a_{44} \end{pmatrix}$$

特点：第一行和最后一行有2个非零元素，第2...n-1行有3个非零元素。使用一维数组存储，需要存储3*n-2个元素。

矩阵的压缩存储（三角矩阵）

$$A=\begin{pmatrix} a_{00} & a_{01} & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{43} & a_{44} \end{pmatrix}$$

三对角矩阵A也可以采用压缩存储，将3条对角线上的元素按行优先存放在一维数组B中，且$a_{1,1}$存放在B[0]z中，如下图所示：

$a_{1,1}$	$a_{1,2}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	…	…	$a_{n-1,n}$	$a_{n,n-1}$	$a_{n,n}$

由此可以计算矩阵A中3条对角线上的元素$a_{i,j}(1 \le i,j \le n, |i-j| \le 1)$在一维数组B中存放的下标为K=2i+j-3

矩阵的压缩存储（稀疏矩阵）

4、稀疏矩阵的压缩存储

稀疏矩阵：大多数元素均为零的矩阵。矩阵元素个数s相对于矩阵中非零元素个数t来说非常多，即s>>t的矩阵称为稀疏矩阵。

数据对象集合表示：稀疏矩阵可以使用三元组顺序表表示，其中三元组格式为(i,j,v)记录了非零元素的行号、列号以及非零元素。

$$M=\begin{bmatrix}4 & 0 & 0 & 0\\ 0 & 0 & 6 & 0\\ 0 & 9 & 0 & 0\\ 0 & 23 & 0 & 0\end{bmatrix}$$

对应的三元组：

i	j	v
0	0	4
1	2	6
2	1	9
3	1	23

矩阵的压缩存储（真题检测）

1.对稀疏矩阵进行压缩存储的目的是（ ）

A.便于进行矩阵运算　　B.便于输入和输出

C.节省存储空间　　D.降低运算的时间复杂度

2.对n阶对阵矩阵压缩存储时，需要表长为（ ）的顺序表

A.n/2　　B.$n^2/2$

C.n(n+1)/2　　D.n(n-1)/2

答案：C　　C

串

字符串简称串，是一种特殊的线性表，它的数据元素仅由一个字符组成。
串(String)是由零个或多个字符组成的有限序列。
串中字符的个数称为串的长度，含有零个元素的串叫空串。
串中任意连续的字符组成的子序列称为该串的子串,包含子串的串称为主串,某个字符在串中的序号称为这个字符的位置。
通常用子串第一个字符的位置作为子串在主串中的位置。要注意的是,空格也是串字符集合的一个元素,由一个或者多个空格组成的串称为空格串(注意,空格串不是空串)。

串

串的逻辑结构和线性表类似,串是限定了元素为字符的线性表。
从操作集上讲,串和线性表有很大的区别,线性表的操作主要针对表内的某一个元素,而串操作主要针对串内的一个子串。
注意：空白串和空串的不同，例如“ ”和“”分别表示长度为1的空白串和长度为0的空串。

串

子串(substring)：串中任意个连续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。

子串的序号：将子串在主串中首次出现时的该子串的首字符对应在主串中的序号，称为子串在主串中的序号（或位置）。

例如，设有串A和B分别是：A=“这是字符串”，B=“是”

则B是A的子串，A为主串。B在A中出现了一次，其中首次出现所对应的主串位置是2。因此，称B在A中的序号为2 。

特别地，空串是任意串的子串，任意串是其自身的子串。

串

串相等：如果两个串的串值相等(相同)，称这两个串相等。换言之，只有当两个串的长度相等，且各个对应位置的字符都相同时才相等。

长度为n的字符串的子串个数：

1、有n(n+1)/2 +1个子串；　2、非空子串：n（n+1）/2；

3、非空真子串：n（n+1）/2– 1。

串

串的存储表示

串是一种特殊的线性表，其存储表示和线性表类似，但又不完全相同。串的存储方式取决于将要对串所进行的操作。串在计算机中有3种表示方式：

- 定长顺序存储表示：将串定义成字符数组，利用串名可以直接访问串值。用这种表示方式，串的存储空间在编译时确定，其大小不能改变。
- 堆分配存储方式：仍然用一组地址连续的存储单元来依次存储串中的字符序列，但串的存储空间是在程序运行时根据串的实际长度动态分配的。
- 块链存储方式：是一种链式存储结构表示。

串的模式匹配算法

模式匹配(模范匹配)：子串在主串中的定位称为模式匹配或串匹配(字符串匹配）。模式匹配成功是指在主串S中能够找到模式串T，否则，称模式串T在主串S中不存在。

模式匹配的应用在非常广泛。例如，在文本编辑程序中，我们经常要查找某一特定单词在文本中出现的位置。显然，解此问题的有效算法能极大地提高文本编辑程序的响应性能。

模式匹配是一个较为复杂的串操作过程。迄今为止，人们对串的模式匹配提出了许多思想和效率各不相同的计算机算法。介绍两种主要的模式匹配算法。

串

简单模式匹配算法

对一个串中某子串的定位操作称为串的模式匹配,其中待定位的子串称为模式串。算法的基本思想从主串的第一个位置起和模式串的第一个字符开始比较,如果相等,则继续逐一比较后续字符:否则从主串的第二个字符开始,再重新用上一步的方法与模式串中的字符做比较,以此类推,直到比较完模式串中的所有字符。若匹配成功,则返回模式串在主串中的位置:若匹配不成功,则返回一个可区别于主串所有位置的标记,如“0”

串

第1趟	A	B	A	B	C	A	B	C	A	C	B	A	B	失败
	A	B	C	A	C									
第2趟	A	B	A	B	C	A	B	C	A	C	B	A	B	失败
		A	B	C	A	C								
第3趟	A	B	A	B	C	A	B	C	A	C	B	A	B	失败
			A	B	C	A	C							
第4趟	A	B	A	B	C	A	B	C	A	C	B	A	B	失败
				A	B	C	A	C						
第5趟	A	B	A	B	C	A	B	C	A	C	B	A	B	失败
					A	B	C	A	C					
第6趟	A	B	A	B	C	A	B	C	A	C	B	A	B	成功
						A	B	C	A	C				

串

该算法简单，易于理解。在一些场合的应用里，如文字处理中的文本编辑，其效率较高。

该算法的时间复杂度为O(n*m)，其中n、m分别是主串和模式串的长度。

理解该算法的关键点

当第一次$s_k \neq t_j$时：主串要退回到k-j+1的位置，而模式串也要退回到第一个字符（即j=0的位置）。

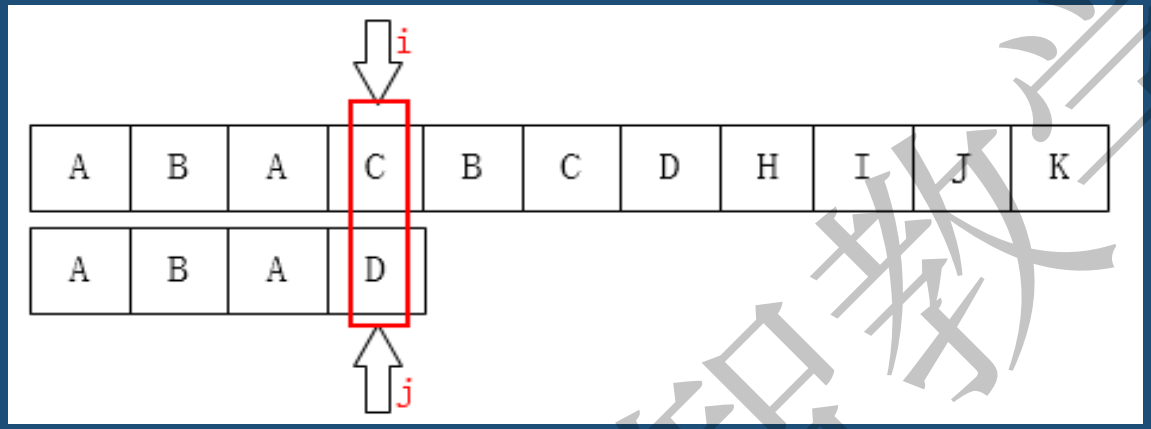
比较出现$s_k \neq t_j$时：则应该有$s_{k-1}=t_{j-1}$，…，$s_{k-j+1}=t_1$，$s_{k-j}=t_0$。

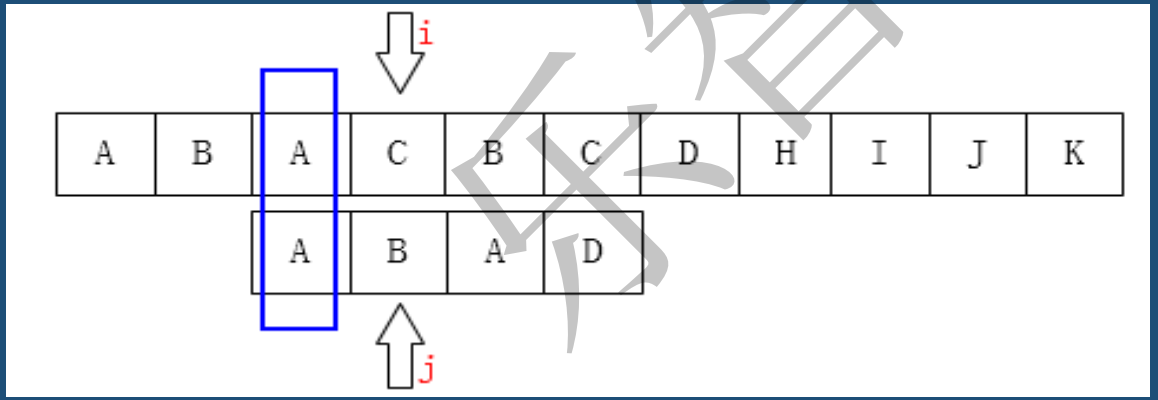
串

KMP算法：该改进算法是由D.E.Knuth，J.H.Morris和 V.R.Pratt提出来的，简称为KMP算法。其改进在于：

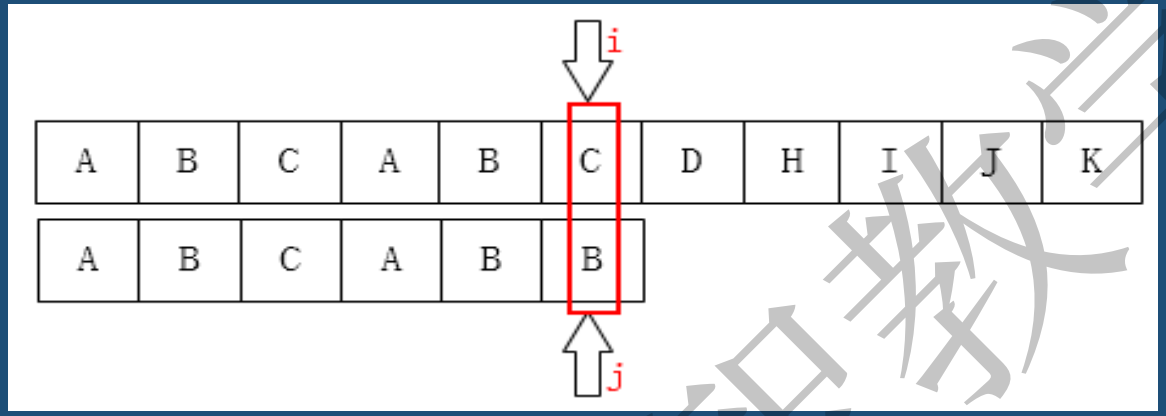
每当一趟匹配过程出现字符不相等时，主串指示器不用回溯，而是利用已经得到的“部分匹配”结果，将模式串的指示器向右“滑动”尽可能远的一段距离后，继续进行比较。

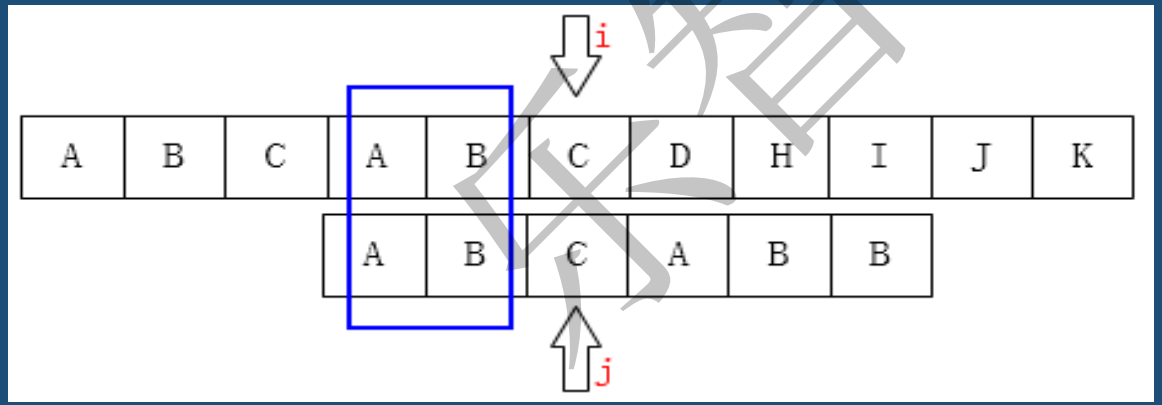
串





串





串

KMP算法的思想是：

设目标串(主串)为s，模式串为t，并设i指针和j指针分别指示目标串和模式串中正待比较的字符，设i和j的初值均为1。若有$s_i=t_j$，则i和j分别加1。否则，i不变，j退回到j=next[j]的位置，再比较s_i和t_j，若相等，则i和j分别加1。否则，i不变，j再次退回到j=next[j]的位置，依此类推。直到下列两种可能：

(1) j退回到某个下一个[j]值时字符比较相等，则指针各自加1继续进行匹配。

(2)退回到j=0，将i和j分别加1，即从主串的下一个字符s_{i+1}模式串的t_1重新开始匹配

串

前缀：指的是字符串的子串中从原串最前面开始的子串，如abcdef的前缀有：a,ab,abc,abcd,abcde

后缀：指的是字符串的子串中在原串结尾处结尾的子串，如abcdef的后缀有：f,ef,def,cdef,bcdef

我们规定任何一个串，next[1]=0　前后缀没有重合时写1　重合时 写重合数+1

j	1	2	3	4	5	6
模式串T	a	b	c	d	e	x
next[j]	0	1	1	1	1	1

j	1	2	3	4	5	6
模式串T	a	b	c	a	b	x
next[j]	0	1	1	1	2	3

串

j	1	2	3	4	5	6	7	8	9
模式串T	a	b	a	b	a	a	a	b	a
next[j]	0	1	1	2	3	4	2	2	3

j	1	2	3	4	5	6	7	8	9
模式串T	a	a	a	a	a	a	a	a	b
next[j]	0	1	2	3	4	5	6	7	8

串

j: 1 2 3 4 5 6 7 8

P: a b a a b c a c

Next[j]:

串（真题检测）

1.（判断题） 空格串和空串是相同的 （ ）

2.串是一种特殊的线性表，其特殊性表现在 （ ）
A.可以顺序存储 B.数据元素是一个字符
C.可以链式存储 D.数据元素可以是多个字符

3.设有两个串p和q，求q在p中首次出现的位置的运算称为（ ）
A.连接 B.模式匹配 C.求字串 D.求串长

4.（填空）串的两种最基本的存储方式是：

答案：错 B B 顺序存储方式和链式存储方式

串（真题检测）

1.设串S1=‘data structures with java’，S2='it'，则子串定位函数index（S1，S2）的值为（ ）

A.15　　B.16　　C.18　　D.19

2.在用KMP算法进行模式匹配时，模式串“ababaaababaa”的next数组值为（）

A.0，1，2，3，4，5，6，7，8，9，9，9

B.0，1，2，1，2，1，1，1，1，2，1，2

C.0，1，1，2，3，4，2，2，3，4，5，6

D.0，1，2，3，0，1，2，3，2，2，3，4

3.子串“str”在主串“datastructure”中的位置是:

答案：C　　C　　5

广义表

广义表简称表，它是线性表的推广。一个广义表是n（n≥0）个元素的一个序列，若n=0时则称为空表。

由于广义表中有两种数据元素，因此需要有两种结构的节点——一种是表结点，一种是原子结点。

广义表

广义表具有如下重要的特性：

（1）广义表中的数据元素有相对次序；

（2）广义表的广度定义为最外层包含元素个数；

（3）广义表的深度定义为所含括弧的重数。其中原子的深度为0，空表的深度为1；

（4）广义表可以共享；一个广义表可以为其他广义表共享；这种共享广义表称为再入表；

（5）广义表可以是一个递归的表。一个广义表可以是自己的子表。这种广义表称为递归表。递归表的深度是无穷值,长度是有限值；

（6）任何一个非空广义表GL均可分解为表头head(GL) = a1和表尾tail(GL) = (a2,…,an) 两部分。（重点！！！）

广义表的表示

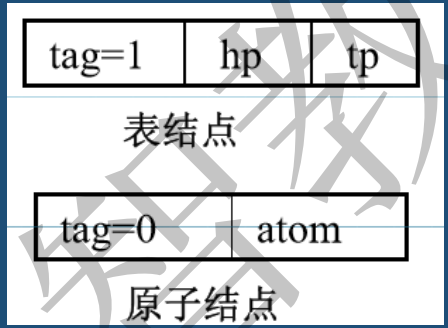
我们规定用小写字母表示原子，用大写字母表示广义表的表名。例如：

A=()
B=(e)
C=(a,(b,c,d))
D=(A,B,C)=((),(e),(a,(b,c,d)))
E=((a,(a,b),((a,b),c)))

其中A是一个空表，其长度为0,其深度为1；
B是只含有单个原子e的表，其长度为1 ,其深度为1 ；
C有两个元素，一个是原子a，另一个是子表，其长度为2 ,其深度为2 ；
D有三个元素，每个元素都是一个表，其长度为3 ,其深度为3 ；
E中只含有一个元素，是一个表，它的长度为1 ,其深度为4 ；

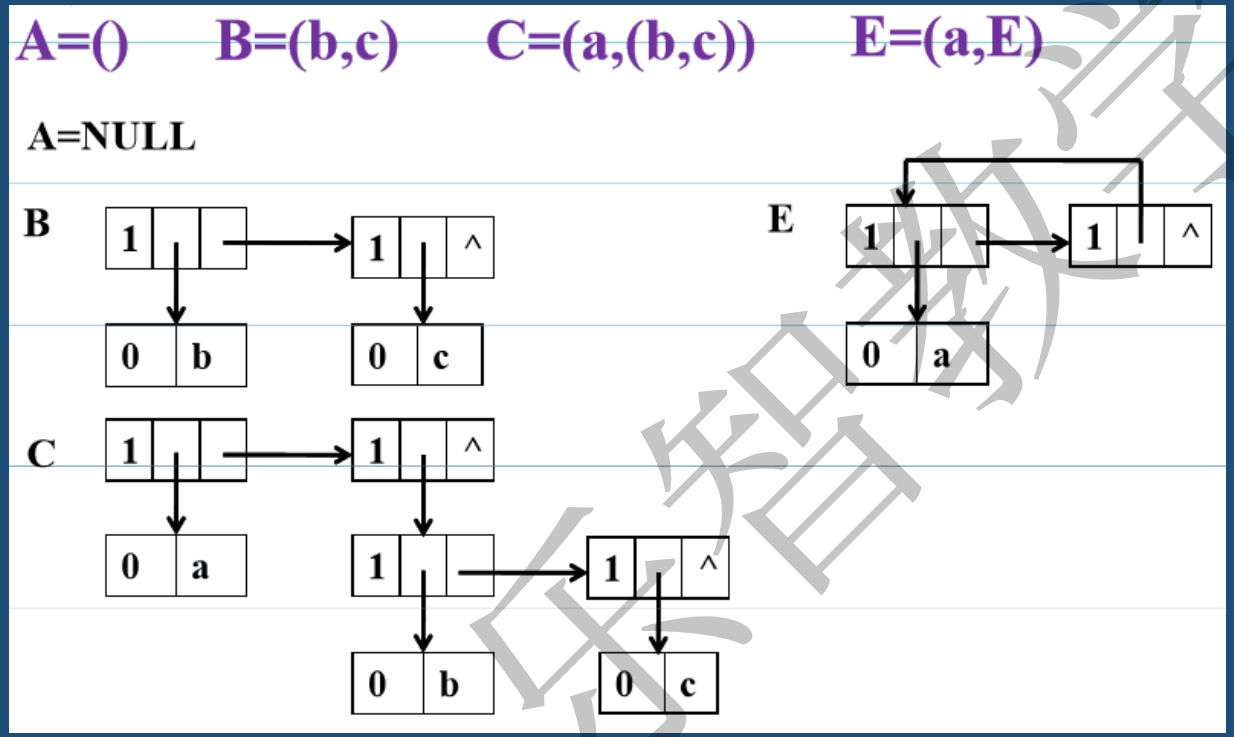
广义表的存储结构

广义表是一种递归的数据结构,广义表中的数据元素可以具有不同的结构，因此，难以用顺序存储结构表示，通常采用链式存储结构，每个数据元素可用一个节点表示。由于广义表中有两种数据元素，因此需要有两种结构的节点——一种是表结点，一种是原子结点。



表结点由三个域组成：标志域、指示表头的指针的指针域和指示表尾的指针域；
而原子域只需两个域：标志域和值域。
注意：表结点的tag=1，原子结点的tag=0

广义表的存储结构



广义表中的head与tail运算

根据表头、表尾的定义可知：任何一个非空广义表的表头是表中第一个元素，它可以是原子，也可以是子表，而其表尾必定是子表。

也就是说，广义表的head操作，取出的元素是什么，那么结果就是什么。但是tail操作取出的元素外必须加一个表——“（）”

举一个简单的列子：已知广义表LS=((a,b,c),(d,e,f)),如果需要取出这个e这个元素，那么使用tail和head如何将这个取出来。

利用上面说的，tail取出来的始终是一个表，即使只有一个简单的一个元素，tail取出来的也是一个表，而head取出来的可以是一个元素也可以是一个表。

广义表中的head与tail运算

解：

tail(LS) = ((d,e,f))

head(tail(LS)) = (d,e,f)

tail(head(tail(LS))) = (e,f)　　//无论如何都会加上这个()括号

head(tail(head(tail(LS)))) = e　　//head可以去除单个元素

广义表（真题检测）

1.广义表A=(a,b,(c,d),(e,(f,g))),则head(tail(head(tail(tail(A)))))=（　）

A.(g)　　B.(d)　　C.c　　D.d

分析：head--tail操作取广义表中元素操作是从里往外取，具体步骤：

A=(a,b,(c,d),(e,(f,g)))

tail(A)　　(b,(c,d),(e,(f,g)))

tail(tail(A))　　((c,d),(e,(f,g)))

head(tail(tail(A)))　　(c,d)

tail(head(tail(tail(A))))　　(d)

head(tail(head(tail(tail(A)))))　　d

提问：A的深度与广度？　答：深度为3，长度为4

广义表（真题检测）

2.已知广义表L=((x,y,z),a,(u,t,w)),从L表中取出原子项t的运算是（　）

A.head(tail(tail(L)))　　B.tail(head(head(tail(L))))

C.head(tail(head(tail(L))))　　D.head(tail(head(tail(tail(L)))))

分析：具体步骤：

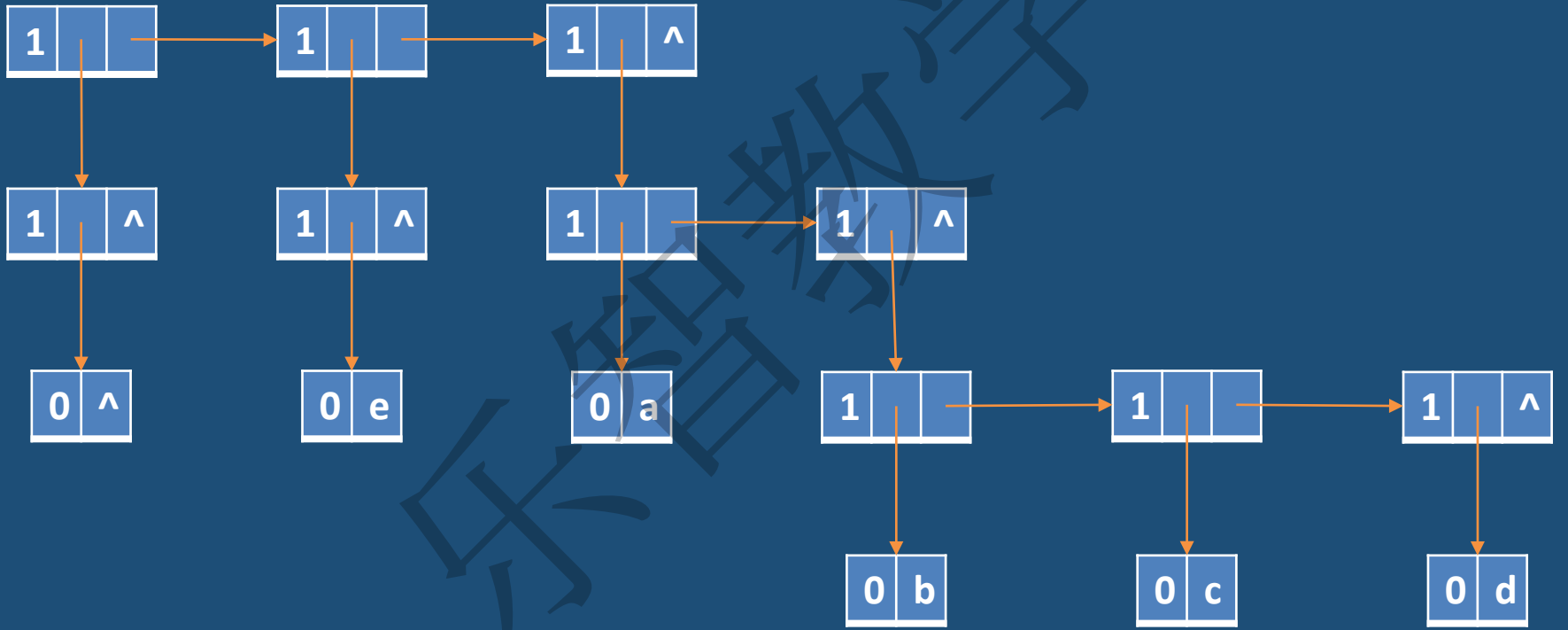
L=((x,y,z),a,(u,t,w))

tail(L)	(a,(u,t,w))
tail(tail(L))	((u,t,w))
head(tail(tail(L)))	(u,t,w)
tail(head(tail(tail(L))))	(t,w)
head(tail(head(tail(tail(L)))))	t

提问：L的深度与广度？　答：深度为2，长度为3

广义表（真题检测）

3.画出广义表LS=((),(e),(a,(b,c,d)))的头尾链表存储结构。



谢 谢 观 看