**第一章：概论(包括习题与答案及要点)**

　　本章的重点是了解数据结构的逻辑结构、存储结构、数据的运算三方面的概念及相互关系，难点是算法复杂度的分析方法。

　　需要达到<识记>层次的基本概念和术语有：数据、数据元素、数据项、数据结构。特别是数据结构的逻辑结构、存储结构及数据运算的含义及其相互关系。数据结构的两大类逻辑结构和四种常用的存储表示方法。

　　需要达到<领会>层次的内容有算法、算法的时间复杂度和空间复杂度、最坏的和平均时间复杂度等概念，算法描述和算法分析的方法、对一般的算法要能分析出时间复杂度。

--------------------------------------------------------------------------------

　　对于基本概念，仔细看书就能够理解，这里简单提一下：

　　数据就是指能够被计算机识别、存储和加工处理的信息的载体。

　　数据元素是数据的基本单位，有时一个数据元素可以由若干个数据项组成。数据项是具有独立含义的最小标识单位。如整数这个集合中，10这个数就可称是一个数据元素.又比如在一个数据库(关系式数据库)中，一个记录可称为一个数据元素，而这个元素中的某一字段就是一个数据项。

　　数据结构的定义虽然没有标准，但是它包括以下三方面内容：逻辑结构、存储结构、和对数据的操作。这一段比较重要，我用自己的语言来说明一下，大家看看是不是这样。

　　比如一个表(数据库)，我们就称它为一个数据结构，它由很多记录(数据元素)组成，每个元素又包括很多字段(数据项)组成。那么这张表的逻辑结构是怎么样的呢? 我们分析数据结构都是从结点(其实也就是元素、记录、顶点，虽然在各种情况下所用名字不同，但说的是同一个东东)之间的关系来分析的，对于这个表中的任一个记录(结点)，它只有一个直接前趋，只有一个直接后继(前趋后继就是前相邻后相邻的意思)，整个表只有一个开始结点和一个终端结点，那我们知道了这些关系就能明白这个表的逻辑结构了。

　　而存储结构则是指用计算机语言如何表示结点之间的这种关系。如上面的表，在计算机语言中描述为连续存放在一片内存单元中，还是随机的存放在内存中再用指针把它们链接在一起，这两种表示法就成为两种不同的存储结构。(注意，在本课程里，我们只在高级语言的层次上讨论存储结构。)

　　第三个概念就是对数据的运算，比如一张表格，我们需要进行查找，增加，修改，删除记录等工作，而怎么样才能进行这样的操作呢? 这也就是数据的运算，它不仅仅是加减乘除这些算术运算了，在数据结构中，这些运算常常涉及算法问题。

　　弄清了以上三个问题，就可以弄清数据结构这个概念。  
  
--------------------------------------------------------------------------------

　　通常我们就将数据的逻辑结构简称为数据结构，数据的逻辑结构分两大类：线性结构和非线性结构 (这两个很容易理解)

　　数据的存储方法有四种：顺序存储方法、链接存储方法、索引存储方法和散列存储方法。  
  
--------------------------------------------------------------------------------

　　下一个是难点问题，就是算法的描述和分析，主要是算法复杂度的分析方法及其运用。 首先了解一下几个概念。一个是时间复杂度，一个是渐近时间复杂度。前者是某个算法的时间耗费，它是该算法所求解问题规模n的函数，而后者是指当问题规模趋向无穷大时，该算法时间复杂度的数量级。

　　当我们评价一个算法的时间性能时，主要标准就是算法的渐近时间复杂度，因此，在算法分析时，往往对两者不予区分，经常是将渐近时间复杂度T(n)=O(f(n)简称为时间复杂度，其中的f(n)一般是算法中频度最大的语句频度。

　　此外，算法中语句的频度不仅与问题规模有关，还与输入实例中各元素的取值相关。但是我们总是考虑在最坏的情况下的时间复杂度。以保证算法的运行时间不会比它更长。

　　常见的时间复杂度，按数量级递增排列依次为：常数阶O(1)、对数阶O(log2n)、线性阶O(n)、线性对数阶O(nlog2n)、平方阶O(n^2)、立方阶O(n^3)、k次方阶O(n^k)、指数阶O(2^n)。

　　时间复杂度的分析计算请看书本上的例子，然后我们通过做练习加以领会和巩固。  
  
**数据结构习题一**

--------------------------------------------------------------------------------

1.1 简述下列概念：数据、数据元素、数据类型、数据结构、逻辑结构、存储结构、线性结构、非线性结构。

◆ 数据：指能够被计算机识别、存储和加工处理的信息载体。

◆ 数据元素：就是数据的基本单位，在某些情况下，数据元素也称为元素、结点、顶点、记录。数据元素有时可以由若干数据项组成。

◆ 数据类型：是一个值的集合以及在这些值上定义的一组操作的总称。

◆ 数据结构：指的是数据之间的相互关系，即数据的组织形式。一般包括三个方面的内容:数据的逻辑结构、存储结构和数据的运算。

◆ 逻辑结构：指各数据元素之间的逻辑关系。

◆ 存储结构：就是数据的逻辑结构用计算机语言的实现。

◆ 线性结构：数据逻辑结构中的一类，它的特征是若结构为非空集，则该结构有且只有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表就是一个典型的线性结构。

◆ 非线性结构：数据逻辑结构中的另一大类，它的逻辑特征是一个结点可能有多个直接前趋和直接后继。

--------------------------------------------------------------------------------

1.2 试举一个数据结构的例子、叙述其逻辑结构、存储结构、运算三个方面的内容。

◆ 例如有一张学生成绩表，记录了一个班的学生各门课的成绩。按学生的姓名为一行记成的表。这个表就是一个数据结构。每个记录(有姓名，学号，成绩等字段)就是一个结点，对于整个表来说，只有一个开始结点(它的前面无记录)和一个终端结点(它的后面无记录)，其他的结点则各有一个也只有一个直接前趋和直接后继(它的前面和后面均有且只有一个记录)。这几个关系就确定了这个表的逻辑结构。

　　那么我们怎样把这个表中的数据存储到计算机里呢? 用高级语言如何表示各结点之间的关系呢? 是用一片连续的内存单元来存放这些记录(如用数组表示)还是随机存放各结点数据再用指针进行链接呢? 这就是存储结构的问题，我们都是从高级语言的层次来讨论这个问题的。(所以各位赶快学C语言吧)。

　　最后，我们有了这个表(数据结构)，肯定要用它，那么就是要对这张表中的记录进行查询，修改，删除等操作，对这个表可以进行哪些操作以及如何实现这些操作就是数据的运算问题了。

--------------------------------------------------------------------------------

1.3 常用的存储表示方法有哪几种?

常用的存储表示方法有四种:  
◆ 顺序存储方法：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为顺序存储结构。  
◆ 链接存储方法：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为链式存储结构。  
◆ 索引存储方法：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。  
◆ 散列存储方法：就是根据结点的关键字直接计算出该结点的存储地址。

--------------------------------------------------------------------------------

1.4 设三个函数f,g,h分别为 f(n)=100n^3+n^2+1000 , g(n)=25n^3+5000n^2 , h(n)=n^1.5+5000nlgn 请判断下列关系是否成立：

(1) f(n)=O(g(n))   
(2) g(n)=O(f(n))   
(3) h(n)=O(n^1.5)  
(4) h(n)=O(nlgn)

◆ (1)成立。

◇ 这里我们复习一下渐近时间复杂度的表示法T(n)=O(f(n))，这里的"O"是数学符号，它的严格定义是"若T(n)和f(n)是定义在正整数集合上的两个函数，则T(n)=O(f(n))表示存在正的常数C和n0 ,使得当n≥n0时都满足0≤T(n)≤C·f(n)。"用容易理解的话说就是这两个函数当整型自变量n趋向于无穷大时，两者的比值是一个不等于0的常数。这么一来，就好计算了吧。第(1)题中两个函数的最高次项都是n^3,因此当n→∞时，两个函数的比值是一个常数，所以这个关系式是成立的。

◆ （2）成立。

◆ （3）成立。

◆ （4）不成立。

--------------------------------------------------------------------------------

1.5 设有两个算法在同一机器上运行，其执行时间分别为100n^2和2^n,要使前者快于后者，n至少要多大?

◆ 15

◇ 最简单最笨的办法就是拿自然数去代呗。假定n取为10，则前者的值是10000，后者的值是1024,小于前者，那我们就加个5，用15代入得前者为22500，后者为32768，已经比前者大但相差不多，那我们再减个1，用14代入得，前者为19600,后者为16384，又比前者小了，所以结果得出来就是n至少要是15.

--------------------------------------------------------------------------------

1.6 设n为正整数，利用大"O"记号，将下列程序段的执行时间表示为n的函数。

1.6 设n为正整数，利用大"O"记号，将下列程序段的执行时间表示为n的函数。

|  |  |
| --- | --- |
| **(1)** i=1; k=0  while(i<n) { k=k+10\*i;i++; } | ◆ T(n)=n-1  ∴ T(n)=O(n) **◇** 这个函数是按线性阶递增的 |
| **(2)** i=0; k=0; do{ k=k+10\*i; i++;  } while(i<n); | ◆ T(n)=n  ∴ T(n)=O(n) **◇** 这也是线性阶递增的 |
| **(3)** i=1; j=0;  while(i+j<=n)  { if (i<j)j++; else i++; } | ◆ T(n)=n/2  ∴ T(n)=O(n) **◇** 虽然时间函数是n/2,但其数量级仍是按线性阶递增的。 |
| **(4)**x=n; // n>1  while (x>=(y+1)\*(y+1)) y++; | ◆ T(n)=n1/2 ∴ T(n)=O(n1/2) **◇** 最坏的情况是y=0，那么循环的次数是n1/2次，这是一个按平方根阶递增的函数。 |
| **(5)** x=91; y=100;  while(y>0) if(x>100) {x=x-10;y--;} else x++; | ◆ T(n)=O(1) **◇** 这个程序看起来有点吓人，总共循环运行了1000次，但是我们看到n没有? 没。这段程序的运行是和n无关的，就算它再循环一万年，我们也不管他，只是一个常数阶的函数。 |

--------------------------------------------------------------------------------

1.7 算法的时间复杂度仅与问题的规模相关吗?

◆ No,事实上，算法的时间复杂度不仅与问题的规模相关，还与输入实例中的元素取值等相关，但在最坏的情况下，其时间复杂度就是只与求解问题的规模相关的。我们在讨论时间复杂度时，一般就是以最坏情况下的时间复杂度为准的。

--------------------------------------------------------------------------------

1.8 按增长率由小至大的顺序排列下列各函数： 2^100, (2/3)^n，(3/2)^n， n^n , , n! ，2^n ，lgn ,n^lgn, n^(3/2)

◇ 分析如下：2^100 是常数阶； (2/3)^n和 (3/2)^n是指数阶,其中前者是随n的增大而减小的； n^n是指数方阶； √n 是方根阶, n! 就是n(n-1)(n-2)... 就相当于n次方阶；2^n 是指数阶，lgn是对数阶 ,n^lgn是对数方阶, n^(3/2)是3/2次方阶。根据以上分析按增长率由小至大的顺序可排列如下：

◆ (2/3)^n < 2^100 < lgn < √n < n^(3/2) < n^lgn < (3/2)^n < 2^n < n! < n^n

--------------------------------------------------------------------------------

1.9 有时为了比较两个同数量级算法的优劣，须突出主项的常数因子，而将低次项用大"O"记号表示。例如，设T1(n)=1.39nlgn+100n+256=1.39nlgn+O(n), T2(n)=2.0nlgn-2n=2.0lgn+O(n), 这两个式子表示，当n足够大时T1(n)优于T2(n)，因为前者的常数因子小于后者。请用此方法表示下列函数，并指出当n足够大时，哪一个较优，哪一个较劣?

|  |  |  |
| --- | --- | --- |
| 函 数 | 大"O"表示 | 优劣 |
| (1) T1(n)=5n^2-3n+60lgn | ◆ 5n^2+O(n) | ◆ 较差 |
| (2) T2(n)=3n^2+1000n+3lgn | ◆ 3n^2+O(n) | ◆ 其次 |
| (3) T3(n)=8n^2+3lgn | ◆ 8n^2+O(lgn) | ◆ 最差 |
| (4) T4(n)=1.5n^2+6000nlgn | ◆ 1.5n^2+O(nlgn) | ◆ 最优 |

**第一章 概论 复习要点**   
  
本章的复习要点是：

数据、数据元素、数据结构(包括逻辑结构、存储结构)以及数据类型的概念、数据的逻辑结构分为哪两大类，及其逻辑特征、数据的存储结构可用的四种基本存储方法。

时间复杂度与渐近时间复杂度的概念，如何求算法的时间复杂度。

可能出的题目有选择题、填空题或简答题。如：

.........是数据的基本单位，.........是具有独立含义的最小标识单位。

什么是数据结构?什么是数据类型?

数据的............与数据的存储无关，它是独立于计算机的。

数据的存储结构包括顺序存储结构、链式存储结构.......................、...........................

设n为正整数，利用大O记号，将该程序段的执行时间表示为n的函数，则下列程序段的时间复杂度可表示为：(....)

x=91;y=100;  
while(y>10)  
if(x>100){x=x-10;y--;}   
else x++;

A. O(1) B.O(x) C.O(y) D.O(n)   
等等。

顺便一提，基本概念和基本理论的掌握是得分的基本手段。

**第一章：概论(包括习题与答案及要点)**

　　本章的重点是了解数据结构的逻辑结构、存储结构、数据的运算三方面的概念及相互关系，难点是算法复杂度的分析方法。

　　需要达到<识记>层次的基本概念和术语有：数据、数据元素、数据项、数据结构。特别是数据结构的逻辑结构、存储结构及数据运算的含义及其相互关系。数据结构的两大类逻辑结构和四种常用的存储表示方法。

　　需要达到<领会>层次的内容有算法、算法的时间复杂度和空间复杂度、最坏的和平均时间复杂度等概念，算法描述和算法分析的方法、对一般的算法要能分析出时间复杂度。

--------------------------------------------------------------------------------

　　对于基本概念，仔细看书就能够理解，这里简单提一下：

　　数据就是指能够被计算机识别、存储和加工处理的信息的载体。

　　数据元素是数据的基本单位，有时一个数据元素可以由若干个数据项组成。数据项是具有独立含义的最小标识单位。如整数这个集合中，10这个数就可称是一个数据元素.又比如在一个数据库(关系式数据库)中，一个记录可称为一个数据元素，而这个元素中的某一字段就是一个数据项。

　　数据结构的定义虽然没有标准，但是它包括以下三方面内容：逻辑结构、存储结构、和对数据的操作。这一段比较重要，我用自己的语言来说明一下，大家看看是不是这样。

　　比如一个表(数据库)，我们就称它为一个数据结构，它由很多记录(数据元素)组成，每个元素又包括很多字段(数据项)组成。那么这张表的逻辑结构是怎么样的呢? 我们分析数据结构都是从结点(其实也就是元素、记录、顶点，虽然在各种情况下所用名字不同，但说的是同一个东东)之间的关系来分析的，对于这个表中的任一个记录(结点)，它只有一个直接前趋，只有一个直接后继(前趋后继就是前相邻后相邻的意思)，整个表只有一个开始结点和一个终端结点，那我们知道了这些关系就能明白这个表的逻辑结构了。

　　而存储结构则是指用计算机语言如何表示结点之间的这种关系。如上面的表，在计算机语言中描述为连续存放在一片内存单元中，还是随机的存放在内存中再用指针把它们链接在一起，这两种表示法就成为两种不同的存储结构。(注意，在本课程里，我们只在高级语言的层次上讨论存储结构。)

　　第三个概念就是对数据的运算，比如一张表格，我们需要进行查找，增加，修改，删除记录等工作，而怎么样才能进行这样的操作呢? 这也就是数据的运算，它不仅仅是加减乘除这些算术运算了，在数据结构中，这些运算常常涉及算法问题。

　　弄清了以上三个问题，就可以弄清数据结构这个概念。  
  
--------------------------------------------------------------------------------

　　通常我们就将数据的逻辑结构简称为数据结构，数据的逻辑结构分两大类：线性结构和非线性结构 (这两个很容易理解)

　　数据的存储方法有四种：顺序存储方法、链接存储方法、索引存储方法和散列存储方法。  
  
--------------------------------------------------------------------------------

　　下一个是难点问题，就是算法的描述和分析，主要是算法复杂度的分析方法及其运用。 首先了解一下几个概念。一个是时间复杂度，一个是渐近时间复杂度。前者是某个算法的时间耗费，它是该算法所求解问题规模n的函数，而后者是指当问题规模趋向无穷大时，该算法时间复杂度的数量级。

　　当我们评价一个算法的时间性能时，主要标准就是算法的渐近时间复杂度，因此，在算法分析时，往往对两者不予区分，经常是将渐近时间复杂度T(n)=O(f(n)简称为时间复杂度，其中的f(n)一般是算法中频度最大的语句频度。

　　此外，算法中语句的频度不仅与问题规模有关，还与输入实例中各元素的取值相关。但是我们总是考虑在最坏的情况下的时间复杂度。以保证算法的运行时间不会比它更长。

　　常见的时间复杂度，按数量级递增排列依次为：常数阶O(1)、对数阶O(log2n)、线性阶O(n)、线性对数阶O(nlog2n)、平方阶O(n^2)、立方阶O(n^3)、k次方阶O(n^k)、指数阶O(2^n)。

　　时间复杂度的分析计算请看书本上的例子，然后我们通过做练习加以领会和巩固。  
  
**数据结构习题一**

--------------------------------------------------------------------------------

1.1 简述下列概念：数据、数据元素、数据类型、数据结构、逻辑结构、存储结构、线性结构、非线性结构。

◆ 数据：指能够被计算机识别、存储和加工处理的信息载体。

◆ 数据元素：就是数据的基本单位，在某些情况下，数据元素也称为元素、结点、顶点、记录。数据元素有时可以由若干数据项组成。

◆ 数据类型：是一个值的集合以及在这些值上定义的一组操作的总称。

◆ 数据结构：指的是数据之间的相互关系，即数据的组织形式。一般包括三个方面的内容:数据的逻辑结构、存储结构和数据的运算。

◆ 逻辑结构：指各数据元素之间的逻辑关系。

◆ 存储结构：就是数据的逻辑结构用计算机语言的实现。

◆ 线性结构：数据逻辑结构中的一类，它的特征是若结构为非空集，则该结构有且只有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。线性表就是一个典型的线性结构。

◆ 非线性结构：数据逻辑结构中的另一大类，它的逻辑特征是一个结点可能有多个直接前趋和直接后继。

--------------------------------------------------------------------------------

1.2 试举一个数据结构的例子、叙述其逻辑结构、存储结构、运算三个方面的内容。

◆ 例如有一张学生成绩表，记录了一个班的学生各门课的成绩。按学生的姓名为一行记成的表。这个表就是一个数据结构。每个记录(有姓名，学号，成绩等字段)就是一个结点，对于整个表来说，只有一个开始结点(它的前面无记录)和一个终端结点(它的后面无记录)，其他的结点则各有一个也只有一个直接前趋和直接后继(它的前面和后面均有且只有一个记录)。这几个关系就确定了这个表的逻辑结构。

　　那么我们怎样把这个表中的数据存储到计算机里呢? 用高级语言如何表示各结点之间的关系呢? 是用一片连续的内存单元来存放这些记录(如用数组表示)还是随机存放各结点数据再用指针进行链接呢? 这就是存储结构的问题，我们都是从高级语言的层次来讨论这个问题的。(所以各位赶快学C语言吧)。

　　最后，我们有了这个表(数据结构)，肯定要用它，那么就是要对这张表中的记录进行查询，修改，删除等操作，对这个表可以进行哪些操作以及如何实现这些操作就是数据的运算问题了。

--------------------------------------------------------------------------------

1.3 常用的存储表示方法有哪几种?

常用的存储表示方法有四种:  
◆ 顺序存储方法：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。由此得到的存储表示称为顺序存储结构。  
◆ 链接存储方法：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为链式存储结构。  
◆ 索引存储方法：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。  
◆ 散列存储方法：就是根据结点的关键字直接计算出该结点的存储地址。

--------------------------------------------------------------------------------

1.4 设三个函数f,g,h分别为 f(n)=100n^3+n^2+1000 , g(n)=25n^3+5000n^2 , h(n)=n^1.5+5000nlgn 请判断下列关系是否成立：

(1) f(n)=O(g(n))   
(2) g(n)=O(f(n))   
(3) h(n)=O(n^1.5)  
(4) h(n)=O(nlgn)

◆ (1)成立。

◇ 这里我们复习一下渐近时间复杂度的表示法T(n)=O(f(n))，这里的"O"是数学符号，它的严格定义是"若T(n)和f(n)是定义在正整数集合上的两个函数，则T(n)=O(f(n))表示存在正的常数C和n0 ,使得当n≥n0时都满足0≤T(n)≤C·f(n)。"用容易理解的话说就是这两个函数当整型自变量n趋向于无穷大时，两者的比值是一个不等于0的常数。这么一来，就好计算了吧。第(1)题中两个函数的最高次项都是n^3,因此当n→∞时，两个函数的比值是一个常数，所以这个关系式是成立的。

◆ （2）成立。

◆ （3）成立。

◆ （4）不成立。

--------------------------------------------------------------------------------

1.5 设有两个算法在同一机器上运行，其执行时间分别为100n^2和2^n,要使前者快于后者，n至少要多大?

◆ 15

◇ 最简单最笨的办法就是拿自然数去代呗。假定n取为10，则前者的值是10000，后者的值是1024,小于前者，那我们就加个5，用15代入得前者为22500，后者为32768，已经比前者大但相差不多，那我们再减个1，用14代入得，前者为19600,后者为16384，又比前者小了，所以结果得出来就是n至少要是15.

--------------------------------------------------------------------------------

1.6 设n为正整数，利用大"O"记号，将下列程序段的执行时间表示为n的函数。

1.6 设n为正整数，利用大"O"记号，将下列程序段的执行时间表示为n的函数。

|  |  |
| --- | --- |
| **(1)** i=1; k=0  while(i<n) { k=k+10\*i;i++; } | ◆ T(n)=n-1  ∴ T(n)=O(n) **◇** 这个函数是按线性阶递增的 |
| **(2)** i=0; k=0; do{ k=k+10\*i; i++;  } while(i<n); | ◆ T(n)=n  ∴ T(n)=O(n) **◇** 这也是线性阶递增的 |
| **(3)** i=1; j=0;  while(i+j<=n)  { if (i<j)j++; else i++; } | ◆ T(n)=n/2  ∴ T(n)=O(n) **◇** 虽然时间函数是n/2,但其数量级仍是按线性阶递增的。 |
| **(4)**x=n; // n>1  while (x>=(y+1)\*(y+1)) y++; | ◆ T(n)=n1/2 ∴ T(n)=O(n1/2) **◇** 最坏的情况是y=0，那么循环的次数是n1/2次，这是一个按平方根阶递增的函数。 |
| **(5)** x=91; y=100;  while(y>0) if(x>100) {x=x-10;y--;} else x++; | ◆ T(n)=O(1) **◇** 这个程序看起来有点吓人，总共循环运行了1000次，但是我们看到n没有? 没。这段程序的运行是和n无关的，就算它再循环一万年，我们也不管他，只是一个常数阶的函数。 |

--------------------------------------------------------------------------------

1.7 算法的时间复杂度仅与问题的规模相关吗?

◆ No,事实上，算法的时间复杂度不仅与问题的规模相关，还与输入实例中的元素取值等相关，但在最坏的情况下，其时间复杂度就是只与求解问题的规模相关的。我们在讨论时间复杂度时，一般就是以最坏情况下的时间复杂度为准的。

--------------------------------------------------------------------------------

1.8 按增长率由小至大的顺序排列下列各函数： 2^100, (2/3)^n，(3/2)^n， n^n , , n! ，2^n ，lgn ,n^lgn, n^(3/2)

◇ 分析如下：2^100 是常数阶； (2/3)^n和 (3/2)^n是指数阶,其中前者是随n的增大而减小的； n^n是指数方阶； √n 是方根阶, n! 就是n(n-1)(n-2)... 就相当于n次方阶；2^n 是指数阶，lgn是对数阶 ,n^lgn是对数方阶, n^(3/2)是3/2次方阶。根据以上分析按增长率由小至大的顺序可排列如下：

◆ (2/3)^n < 2^100 < lgn < √n < n^(3/2) < n^lgn < (3/2)^n < 2^n < n! < n^n

--------------------------------------------------------------------------------

1.9 有时为了比较两个同数量级算法的优劣，须突出主项的常数因子，而将低次项用大"O"记号表示。例如，设T1(n)=1.39nlgn+100n+256=1.39nlgn+O(n), T2(n)=2.0nlgn-2n=2.0lgn+O(n), 这两个式子表示，当n足够大时T1(n)优于T2(n)，因为前者的常数因子小于后者。请用此方法表示下列函数，并指出当n足够大时，哪一个较优，哪一个较劣?

|  |  |  |
| --- | --- | --- |
| 函 数 | 大"O"表示 | 优劣 |
| (1) T1(n)=5n^2-3n+60lgn | ◆ 5n^2+O(n) | ◆ 较差 |
| (2) T2(n)=3n^2+1000n+3lgn | ◆ 3n^2+O(n) | ◆ 其次 |
| (3) T3(n)=8n^2+3lgn | ◆ 8n^2+O(lgn) | ◆ 最差 |
| (4) T4(n)=1.5n^2+6000nlgn | ◆ 1.5n^2+O(nlgn) | ◆ 最优 |

**第一章 概论 复习要点**   
  
本章的复习要点是：

数据、数据元素、数据结构(包括逻辑结构、存储结构)以及数据类型的概念、数据的逻辑结构分为哪两大类，及其逻辑特征、数据的存储结构可用的四种基本存储方法。

时间复杂度与渐近时间复杂度的概念，如何求算法的时间复杂度。

可能出的题目有选择题、填空题或简答题。如：

.........是数据的基本单位，.........是具有独立含义的最小标识单位。

什么是数据结构?什么是数据类型?

数据的............与数据的存储无关，它是独立于计算机的。

数据的存储结构包括顺序存储结构、链式存储结构.......................、...........................

设n为正整数，利用大O记号，将该程序段的执行时间表示为n的函数，则下列程序段的时间复杂度可表示为：(....)

x=91;y=100;  
while(y>10)  
if(x>100){x=x-10;y--;}   
else x++;

A. O(1) B.O(x) C.O(y) D.O(n)   
等等

**第二章：线性表(包括习题与答案及要点)**

本章的**重点**是掌握顺序表和单链表上实现的各种基本算法及相关的时间性能分析，**难点**是使用本章所学的基本知识设计有效算法解决与线性表相关的应用问题。

要求达到<识记>层次的内容有:线性表的逻辑结构特征；线性表上定义的基本运算，并利用基本运算构造出较复杂的运算。

要求达到<综合应用>层次的内容有：顺序表的含义及特点，顺序表上的插入、删除操作及其平均时间性能分析，解决简单应用问题。

链表如何表示线性表中元素之间的逻辑关系；单链表、双链表、循环链表链接方式上的区别；单链表上实现的建表、查找、插入和删除等基本算法及其时间复杂度。循环链表上尾指针取代头指针的作用，以及单循环链表上的算法与单链表上相应算法的异同点。双链表的定义和相关算法。利用链表设计算法解决简单应用问题。

要求达到<领会>层次的内容就是顺序表和链表的比较，以及如何选择其一作为其存储结构才能取得较优的时空性能。

线性表的逻辑结构特征是很容易理解的，如其名，它的逻辑结构特征就好象是一条线，上面打了一个个结，很形象的，如果这条线上面有结，那么它就是非空表，只能有一个开始结点，有且只能有一个终端结点，其它的结前后所相邻的也只能是一个结点(直接前趋和直接后继)。

关于线性表上定义的基本运算，主要有构造空表、求表长、取结点、查找、插入、删除等。

线性表的逻辑结构和存储结构之间的关系。在计算机中，如何把线性表的结点存放到存储单元中，就有许多方法，最简单的方法就是按顺序存储。就是按线性表的逻辑结构次序依次存放在一组地址连续的存储单元中。在存储单元中的各元素的物理位置和逻辑结构中各结点相邻关系是一致的。

在顺序表中实现的基本运算主要讨论了插入和删除两种运算。相关的算法我们通过练习掌握。对于顺序表的插入和删除运算，其平均时间复杂度均为O(n)。

线性表的链式存储结构。它与顺序表不同，链表是用一组任意的存储单元来存放线性表的结点，这组存储单元可以分布在内存中任何位置上。因此，链表中结点的逻辑次序和物理次序不一定相同。所以为了能正确表示结点间的逻辑关系，在存储每个结点值的同时，还存储了其后继结点的地址信息(即指针或链)。这两部分信息组成链表中的结点结构。

一个单链表由头指针的名字来命名。

对于单链表，其操作运算主要有建立单链表(头插法、尾插法和在链表开始结点前附加一个头结点的算法)、查找(按序号和按值)、插入运算、删除运算等。以上各运算的平均时间复杂度均为O(n).其主要时间是耗费在查找操作上。

循环链表是一种首尾相接的链表。也就是终端结点的指针域不是指向NULL空而是指向开始结点(也可设置一个头结点)，形成一个环。采用循环链表在实用中多采用尾指针表示单循环链表。这样做的好处是查找头指针和尾指针的时间都是O(1)，不用遍历整个链表了。

判别链表终止的条件也不同于单链表，它是以指针是否等于某一指定指针如头指针或尾指针来确定。

双链表就是双向链表，就是在单链表的每个结点里再增加一个指向其直接前趋的指针域prior,这样形成的链表就有两条不同方向的链。使得从已知结点查找其直接前趋结点可以和查找其直接后继结点的时间一样缩短为O(1)。

双链表一般也由头指针head惟一确定。双链表也可以头尾相链接构成双(向)循环链表。

关于顺序表和链表的比较，请看下表：

|  |  |  |
| --- | --- | --- |
| 具体要求 | 顺序表 | 链表 |
| 基于空间 | 适于线性表长度变化不大，易于事先确定其大小时采用。 | 适于当线性表长度变化大，难以估计其存储规模时采用。 |
| 基于时间 | 由于顺序表是一种随机存储结构，当线性表的操作主要是查找时，宜采用。 | 链表中对任何位置进行插入和删除都只需修改指针，所以这类操作为主的线性表宜采用链表做存储结构。若插入和删除主要发生在表的首尾两端，则宜采用尾指针表示的单循环链表。 |

**第二章 线性表习题及答案**

--------------------------------------------------------------------------------

一、基础知识题

(答案及点评) 2.1 试描述头指针、头结点、开始结点的区别、并说明头指针和头结点的作用。

一、基础知识题  
2.1 答：  
开始结点是指链表中的第一个结点，也就是没有直接前趋的那个结点。

链表的头指针是一指向链表开始结点的指针(没有头结点时)，单链表由头指针唯一确定，因此单链表可以用头指针的名字来命名。

头结点是我们人为地在链表的开始结点之前附加的一个结点。有了头结点之后，头指针指向头结点，不论链表否为空，头指针总是非空。而且头指针的设置使得对链表的第一个位置上的操作与在表其他位置上的操作一致(都是在某一结点之后)。

--------------------------------------------------------------------------------

(答案及点评) 2.2 何时选用顺序表、何时选用链表作为线性表的存储结构为宜?

2.2 答：  
在实际应用中，应根据具体问题的要求和性质来选择顺序表或链表作为线性表的存储结构，通常有以下几方面的考虑：

1.基于空间的考虑。当要求存储的线性表长度变化不大，易于事先确定其大小时，为了节约存储空间，宜采用顺序表；反之，当线性表长度变化大，难以估计其存储规模时，采用动态链表作为存储结构为好。

2.基于时间的考虑。若线性表的操作主要是进行查找，很少做插入和删除操作时，采用顺序表做存储结构为宜；反之， 若需要对线性表进行频繁地插入或删除等的操作时，宜采用链表做存储结构。并且，若链表的插入和删除主要发生在表的首尾两端，则采用尾指针表示的单循环链表为宜。

--------------------------------------------------------------------------------

(答案及点评) 2.3 在顺序表中插入和删除一个结点需平均移动多少个结点?具体的移动次数取决于哪两个因素?

2.3.答：

在等概率情况下，顺序表中插入一个结点需平均移动n/2个结点。删除一个结点需平均移动(n-1)/2个结点。具体的移动次数取决于顺序表的长度n以及需插入或删除的位置i。i越接近n则所需移动的结点数越少。

--------------------------------------------------------------------------------

(答案及点评) 2.4 为什么在单循环链表中设置尾指针比设置头指针更好?

2.4. 答：

尾指针是指向终端结点的指针，用它来表示单循环链表可以使得查找链表的开始结点和终端结点都很方便，设一带头结点的单循环链表，其尾指针为rear，则开始结点和终端结点的位置分别是rear->next->next 和 rear, 查找时间都是O(1)。

若用头指针来表示该链表，则查找终端结点的时间为O(n)。

--------------------------------------------------------------------------------

(答案及点评) 2.5 在单链表、双链表和单循环链表中，若仅知道指针p指向某结点，不知道头指针，能否将结点\*p从相应的链表中删去?若可以，其时间复杂度各为多少?

2.5 答：

我们分别讨论三种链表的情况。  
1. 单链表。当我们知道指针p指向某结点时，能够根据该指针找到其直接后继，但是由于不知道其头指针，所以无法访问到p指针指向的结点的直接前趋。因此无法删去该结点。

2. 双链表。由于这样的链表提供双向链接，因此根据已知结点可以查找到其直接前趋和直接后继，从而可以删除该结点。其时间复杂度为O(1)。

3. 单循环链表。根据已知结点位置，我们可以直接得到其后相邻的结点位置(直接后继)，又因为是循环链表，所以我们可以通过查找，得到p结点的直接前趋。因此可以删去p所指结点。其时间复杂度应为O(n)。

--------------------------------------------------------------------------------

(答案及点评) 2.6 下述算法的功能是什么?

LinkList Demo(LinkList L){ // L 是无头结点单链表  
ListNode \*Q,\*P;  
if(L&&L->next){  
Q=L;L=L->next;P=L;  
while (P->next) P=P->next;  
P->next=Q; Q->next=NULL;  
}  
return L;  
}// Demo

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

二、算法设计题

(答案及点评) 2.7 设线性表的n个结点定义为(a0,a1,...an-1)，重写顺序表上实现的插入和删除算法：InsertList 和DeleteList.

**

--------------------------------------------------------------------------------

(答案及点评) 2.8 试分别用顺序表和单链表作为存储结构，实现将线性表(a0,a1,...an-1)就地逆置的操作，所谓"就地"指辅助空间应为O(1)。

**

--------------------------------------------------------------------------------

(答案及点评) 2.9 设顺序表L是一个递增有序表，试写一算法，将x插入L中，并使L仍是一个有序表。

2.9 解：  
  
因已知顺序表L是递增有序表，所以只要从头找起找到第一个比它大(或相等)的结点数据，把x插入到这个数所在的位置就是了。算法如下：

void InsertIncreaseList( Seqlist \*L , Datatype x )  
{  
int i;  
for ( i=0 ; i < L -> length && L->data[ i ] < x ; i++) ; // 查找并比较,分号不能少  
InsertList ( L ，x , i ); // 调用顺序表插入函数  
}

--------------------------------------------------------------------------------

(答案及点评) 2.10 设顺序表L是一个递减有序表，试写一算法，将x插入其后仍保持L的有序性。

2.10 解：  
与上题相类似，只要从头找到第一个比x小(或相等)的结点数据，在这个位置插入就可以了。算法如下：

void InsertDecreaseList( Seqlist \*L, Datatype x )  
{  
int i;  
for (i=0; i< L -> length && L-> data[i] > x ; i++) ; //查找  
InsertList ( L , x , i ); // 调用顺序表插入函数  
}

--------------------------------------------------------------------------------

(答案及点评) 2.11 写一算法在单链表上实现线性表的ListLength(L)运算。

2.11 解：

求单链表长只能用遍历的方法了，从头数到尾，总能数出来吧。算法如下：

int ListLength ( LinkList L )  
{  
int len=0 ;  
ListNode \*p;  
p=L; //设该表有头结点  
while ( p->next )  
{  
p=p->next;  
len++;  
}  
return len;  
}

--------------------------------------------------------------------------------

(答案及点评) 2.12 已知L1和L2分别指向两个单链表的头结点，且已知其长度分别为m和n。试写一算法将这两个链表连接在一起，请分析你的算法的时间复杂度。

2.12 解：

算法如下:

LinkList Link( LinkList L1 , LinkList L2 )  
{  
//将两个单链表连接在一起  
ListNode \*p , \*q ;  
p=L1;  
q=L2;  
while ( p->next ) p=p->next; //查找终端结点  
p->next = q->next ; //将L2的开始结点链接在L1之后  
return L1 ;  
}

本算法的主要操作时间花费在查找L1的终端结点上，与L2的长度无关，所以本算的法时间复杂度为：

m+1=O(m)

--------------------------------------------------------------------------------

(答案及点评) 2.13 设 A和B是两个单链表，其表中元素递增有序。试写一算法将A和B归并成一个按元素值递减有序的单链表C，并要求辅助空间为O(1)，请分析算法的时间复杂度。

**

--------------------------------------------------------------------------------

(答案及点评) 2.14 已知单链表L是一个递增有序表，试写一高效算法，删除表中值大于min 且小于max的结点(若表中有这样的结点)，同时释放被删结点的窨，这里min 和 max是两个给定的参数。请分析你的算法的时间复杂度。

**

--------------------------------------------------------------------------------

(答案及点评) 2.15 写一算法将单链表中值重复的结点删除，使所得的结果表中各结点值均不相同。

2.15 解：

本题 可以这样考虑，先取开始结点中的值，将它与其后的所有结点值一一比较，发现相同的就删除掉，然后再取第二结点的值，重复上述过程直到最后一个结点。

第二种算法是将单链表按值的大小排序，排好后的结点按相同的删除。

具体算法略。

--------------------------------------------------------------------------------

(答案及点评) 2.16 假设在长度大于1的单循环链表中，既无头结点也无头指针。s为指向链表中某个结点的指针，试编写算法删除结点\*s的直接前趋结点。

2.16 解：  
已知指向这个结点的指针是\*s，那么要删除这个结点的直接前趋结点，就只要找到一个结点，它的指针域是指向\*s的，把这个结点删除就可以了。

算法如下：  
void DeleteNode( ListNode \*s)  
{  
//删除单循环链表中指定结点的直接前趋结点  
ListNode \*p, \*q;  
p=s;  
while( p->next->next!=s)   
{  
q=p; /  
p=p->next;  
}  
q->next=s; //删除结点  
free(p); //释放空间  
}

--------------------------------------------------------------------------------

(答案及点评) 2.17 已知由单链表表示的线性表中，含有三类字符的数据元素(如：字母字符、数字字符和其它字符)，试编写算法构造三个以循环链表表示的线性表，使每个表中只含同一类的字符，且利用原表中的结点空间作为这三个表的结点空间，头结点可另辟空间。

**

--------------------------------------------------------------------------------

(答案及点评) 2.18 设有一个双链表，每个结点中除有prior、data和next三个域外，还有一个访问频度域freq，在链表被起用之前，其值均初始化为零。每当在链表进行一次LocateNode(L,s)运算时，令元素值为x的结点中freq域的值加1，并调整表中结点的次序，使其按访问频度的递减序排列，以便使频繁访问的结点总是靠近表头。试写一符合上述要求的LocateNode运算的算法。

**

--------------------------------------------------------------------------------  
  
**第二章 线性表 复习要点**

本章复习要点是：

线性表的逻辑结构特征、常见的线性表的六种基本运算，并可以根据这些基本运算组合得到更复杂的运算。

顺序表的特征、顺序表中结点地址的计算。

顺序表上实现的基本运算(算法)：主要是插入和删除的算法。顺序表的算法应该掌握。算法时间复杂度要记住。

单链表的特征、图形表示法。

单链表的各种算法实现，并能运用这些算法解决一些简单问题；

循环链表的特征、双链表的特征以及它们的主要算法实现。

可能出的题型有：填空题、简答题、应用题和算法题.

如：

在双链表中插入运算的时间复杂度为：(...)

A.O(1) B.O(n) C.O(lgn) D.O(nlgn)

请问头指针，开始结点和头结点的区别与联系。

关于单链表上进行排序的算法设计。 等等。

**三章：栈和队列(包括习题与答案及要点)**

本章介绍的是栈和队列的逻辑结构定义及在两种存储结构(顺序存储结构和链式存储结构)上如何实现栈和队列的基本运算。本章的重点是掌握栈和队列在两种存储结构上实现的基本运算，难点是循环队列中对边界条件的处理。

1.栈的逻辑结构、存储结构及其相关算法(综合应用):

栈的逻辑结构和我们先前学过的线性表相同，如果它是非空的，则有且只有一个开始结点，有且只能有一个终端结点，其它的结点前后所相邻的也只能是一个结点(直接前趋和直接后继),但是栈的运算规则与线性表相比有更多的限制，栈(Stack)是仅限制在表的一端进行插入和删除运算的线性表，通常称插入、删除这一端为栈顶，另一端称为栈底。表中无元素时为空栈。栈的修改是按后进先出的原则进行的，我们又称栈为LIFO表(Last In First Out).

栈的基本运算有六种：

构造空栈：InitStack(S)、

判栈空: StackEmpty(S)、

判栈满： StackFull(S)、

进栈： Push(S,x)、可形象地理解为压入，这时栈中会多一个元素

退栈： Pop(S) 、 可形象地理解为弹出，弹出后栈中就无此元素了。

取栈顶元素：StackTop(S),不同与弹出，只是使用栈顶元素的值，该元素仍在栈顶不会改变。

由于栈也是线性表，因此线性表的存储结构对栈也适用，通常栈有顺序栈和链栈两种存储结构，这两种存储结构的不同，则使得实现栈的基本运算的算法也有所不同。

我们要了解的是，在顺序栈中有"上溢"和"下溢"的概念。顺序栈好比一个盒子，我们在里头放了一叠书，当我们要用书的话只能从第一本开始拿(你会把盒子翻过来吗?真聪明^^)，那么当我们把书本放到这个栈中超过盒子的顶部时就放不下了(叠上去的不算，哼哼)，这时就是"上溢","上溢"也就是栈顶指针指出栈的外面，显然是出错了。反之，当栈中已没有书时，我们再去拿，看看没书，把盒子拎起来看看盒底，还是没有，这就是"下溢"。"下溢"本身可以表示栈为空栈,因此可以用它来作为控制转移的条件。

链栈则没有上溢的限制，它就象是一条一头固定的链子，可以在活动的一头自由地增加链环(结点)而不会溢出，链栈不需要在头部附加头结点，因为栈都是在头部进行操作的，如果加了头结点，等于要在头结点之后的结点进行操作，反而使算法更复杂，所以只要有链表的头指针就可以了。

以上两种存储结构的栈的基本操作算法是不同的，我们主要要学会进栈和退栈的基本算法以解决简单的应用问题。

2.队列的逻辑结构、存储结构及其相关算法(综合应用)。

队列(Queue,念Q音)也是一种运算受限的线性表，它的运算限制与栈不同，是两头都有限制，插入只能在表的一端进行(只进不出)，而删除只能在表的另一端进行(只出不进)，允许删除的一端称为队尾(rear)，允许插入的一端称为队头 (Front) ,队列的操作原则是先进先出的，所以队列又称作FIFO表(First In First Out)

队列的基本运算也有六种：

置空队 ：InitQueue(Q)

判队空： QueueEmpty(Q)

判队满： QueueFull(Q)

入队 ： EnQueue(Q,x)

出队 ： DeQueue(Q)

取队头元素： QueueFront(Q),不同与出队，队头元素仍然保留

队列也有顺序存储和链式存储两种存储结构，前者称顺序队列，后者为链队。

对于顺序队列，我们要理解"假上溢"的现象。

我们现实中的队列比如人群排队买票，队伍中的人是可以一边进去从另一头出来的，除非地方不够，总不会有"溢出"的现象，相似地，当队列中元素完全充满这个向量空间时，再入队自然就会上溢，如果队列中已没有元素，那么再要出队也会下溢。

那么"假上溢"就是怎么回事呢?

因为在这里，我们的队列是存储在一个向量空间里，在这一段连续的存储空间中，由一个队列头指针和一个尾指针表示这个队列，当头指针和尾指针指向同一个位置时，队列为空，也就是说，队列是由两个指针中间的元素构成的。在队列中，入队和出队并不是象现实中，元素一个个地向前移动，走完了就没有了,而是指针在移动，当出队操作时，头指针向前(即向量空间的尾部)增加一个位置，入队时，尾指针向前增加一个位置，在某种情况下，比如说进一个出一个，两个指针就不停地向前移动，直到队列所在向量空间的尾部，这时再入队的话，尾指针就要跑到向量空间外面去了，仅管这时整个向量空间是空的，队列也是空的，却产生了"上溢"现象，这就是假上溢。

为了克服这种现象造成的空间浪费，我们引入循环向量的概念，就好比是把向量空间弯起来，形成一个头尾相接的环形，这样，当存于其中的队列头尾指针移到向量空间的上界(尾部)时，再加1的操作(入队或出队)就使指针指向向量的下界，也就是从头开始。这时的队列就称循环队列。

通常我们应用的大都是循环队列。由于循环的原因，光看头尾指针重叠在一起我们并不能判断队列是空的还是满的，这时就需要处理一些边界条件，以区别队列是空还是满。方法至少有三种，一种是另设一个布尔变量来判断(就是请别人看着，是空还是满由他说了算)，第二种是少用一个元素空间，当入队时，先测试入队后尾指针是不是会等于头指针，如果相等就算队已满，不许入队。第三种就是用一个计数器记录队列中的元素的总数，这样就可以随时知道队列的长度了，只要队列中的元素个数等于向量空间的长度，就是队满。

以上是顺序队列，我们要掌握相应算法以解决简单应用问题。

队列的链式存储结构称为链队列，一个链队列就是一个操作受限的单链表。为了便于在表尾进行插入(入队)的操作，在表尾增加一个尾指针，一个链队列就由一个头指针和一个尾指针唯一地确定。链队列不存在队满和上溢的问题。在链队列的出队算法中，要注意当原队中只有一个结点时，出队后要同进修改头尾指针并使队列变空。

3.栈和队列的应用(领会)

教材中举了几个例子，对于我们初学者来说，看上去比较繁，我们只要掌握一点，那就是，对于什么情况下用栈和队列作为解决问题的数据结构。

判断的要点就是：如果这个问题满足后进先出(LIFO)的原则，就可以使用栈来处理。如果这个问题满足先进先出(FIFO)的原则，就可以使用队列来处理。

比如简单的说，有一个数组序列，我们输入时按顺序输入，但是输出时需要逆序输出，那么它就可以利用栈来处理，把这个数组存入一个栈中就可以容易地按逆序输出结果了。  
  
**第三章 线性表习题及答案**

一、基础知识题

*(答案及点评)* 3.1 设将整数1，2，3，4依次进栈，但只要出栈时栈非空，则可将出栈操作按任何次序夹入其中，请回答下述问题：

(1)若入、出栈次序为Push(1), Pop(),Push(2),Push(3), Pop(), Pop( ),Push(4), Pop( ),则出栈的数字序列为何(这里Push(i)表示i进栈，Pop( )表示出栈)?

(2) 能否得到出栈序列1423和1432?并说明为什么不能得到或者如何得到。

(3)请分析 1，2 ，3 ，4 的24种排列中，哪些序列是可以通过相应的入出栈操作得到的。

*(答案及点评)* 3.2 链栈中为何不设置头结点?  
  
答：链栈不需要在头部附加头结点，因为栈都是在头部进行操作的，如果加了头结点，等于要对头结点之后的结点进行操作，反而使算法更复杂，所以只要有链表的头指针就可以了。

*(答案及点评)* 3.3 循环队列的优点是什么? 如何判别它的空和满?   
  
  
3.3 答：循环队列的优点是：它可以克服顺序队列的"假上溢"现象，能够使存储队列的向量空间得到充分的利用。判别循环队列的"空"或"满"不能以头尾指针是否相等来确定，一般是通过以下几种方法：一是另设一布尔变量来区别队列的空和满。二是少用一个元素的空间。每次入队前测试入队后头尾指针是否会重合，如果会重合就认为队列已满。三是设置一计数器记录队列中元素总数，不仅可判别空或满，还可以得到队列中元素的个数。

*(答案及点评)* 3.4 设长度为n的链队用单循环链表表示，若设头指针，则入队出队操作的时间为何? 若只设尾指针呢?   
  
  
3.4答：当只设头指针时，出队的时间为1，而入队的时间需要n，因为每次入队均需从头指针开始查找，找到最后一个元素时方可进行入队操作。若只设尾指针，则出入队时间均为1。因为是循环链表，尾指针所指的下一个元素就是头指针所指元素，所以出队时不需要遍历整个队列。

*(答案及点评)* 3.5 指出下述程序段的功能是什么?

**

二、算法设计题

*(答案及点评)* 3.6 回文是指正读反读均相同的字符序列，如"abba"和"abdba"均是回文，但"good"不是回文。试写一个算法判定给定的字符向量是否为回文。(提示：将一半字符入栈)

**

*(答案及点评)*3.7 利用栈的基本操作，写一个将栈S中所有结点均删去的算法void ClearStack( SeqStack \*S)，并说明S为何要作为指针参数?   
  
  
3.7解： 算法如下

void ClearStack (SeqStack \*S)  
{ // 删除栈中所有结点  
S->Top = -1; //其实只是将栈置空  
}  
因为我们要置空的是栈S，如果不用指针来做参数传递，那么函数进行的操作不能对原来的栈产生影响，系统将会在内存中开辟另外的单元来对形参进行函数操作。结果等于什么也没有做。所以想要把函数操作的结果返回给实参的话，就只能用指针来做参数传递了。

*(答案及点评)* 3.8 利用栈的基本操作， 写一个返回S中结点个数的算法 int StackSize( SeqStack S),并说明S为何不作为指针参数?

3.8 解：算法如下：

int StackSize (SeqStack S)  
{  
//计算栈中结点个数  
int n=0;  
if(!EmptyStack(&S))  
{  
Pop(&S);  
n++;  
}  
return n;  
}

类似于上面的原因，我们要计算栈中元素个数就要弹出元素才能"数"得出来，那如果用指针做参数的话，就会把原来的栈中元素"弹"光，要恢复还得用别的办法给它装回去，而不用指针做参数，则可以避免对原来的栈中元素进行任何操作，系统会把原来的栈按值传递给形参，函数只对形参进行操作，最后返回元素个数就可以了。

*(答案及点评)* 3.9 设计算法判断一个算术表达式的圆括号是否正确配对。 (提示： 对表达式进行扫描，凡遇到'('就进栈，遇')'就退掉栈顶的'('，表达式被扫描完毕，栈应为空。  
  
3.9 解：根据提示，可以设计算法如下：  
#include <string.h>  
#include "stack.h"  
int PairBracket( char \*S)  
{  
//检查表达式中括号是否配对  
int i;  
SeqStack T; //定义一个栈  
InitStack (&T);  
for (i=0; i<strlen(S) ; i++)  
{   
if ( S[i]=='(' ) Push(&T, S[i]); //遇'('时进栈  
if ( S[i]==')' ) Pop(&T); //遇')'时出栈  
}  
return !EmptyStack(&T); // 由栈空否返回正确配对与否  
}

*(答案及点评)* 3.10 一个双向栈S是在同一向量空间内实现的两个栈，它们的栈底分别设在向量空间的两端。 试为此双向栈设计初始化InitStack ( S ) 、入栈Push( S , i , x) 和出栈Pop( S , i )等算法， 其中i为0 或1， 用以表示栈号。

**

*(答案及点评)* 3.11 Ackerman 函数定义如下：请写出递归算法。

　　　　　　　　[ n+1　　　 当m=0时

AKM ( m , n ) = \{ AKM( m-1 ,1) 当m≠0 ，n=0时

　　　　　　　　[ AKM( m-1, AKM( m,n-1)) 当m≠0, n ≠ 0时

3.11 解：算法如下

int AKM( int m, int n)  
{  
if ( m== 0) return n+1;  
if ( m<>0 && n==0 ) return AKM( m-1, 1);  
if ( m<>0 && n<>0 ) return AKM( m-1, AKM( m, n-1));  
}

*(答案及点评)*3.12 用第二种方法 ，即少用一上元素空间的方法来区别循环队列的队空和队满，试为其设计置空队，判队空，判队满、出队、入队及取队头元素等六个基本操作的算法。

**

*(答案及点评)*3.13 假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素站点(注意不设头指针) ，试编写相应的置空队、判队空 、入队和出队等算法。

**

*(答案及点评)*3.14 对于循环向量中的循环队列，写出求队列长度的公式。

3.14解：公式如下：  
{ 0 　　EmptyQueue   
Queuelen={ rear - front 　　rear>front  
{ rear+QueueSize-front rear<front  
{ QueueSize FullQueue

*(答案及点评)* 3.15 假设循环队列中只设rear和quelen 来分别指示队尾元素的位置和队中元素的个数，试给出判别此循环队列的队满条件，并写出相应的入队和出队算法，要求出队时需返回队头元素。

**

**第三章 栈和队列 复习要点**

本章要点是：

栈的定义、其逻辑结构特征、栈的六种基本运算、栈的上溢、下溢的概念。

队列的逻辑结构，队列的基本运算；循环队列的边界条件处理；

以上各种基本运算算法的实现。算法的简单应用。

可能出的题型有填空、选择、简答、算法等，如:

栈和队列是.........的线性表；

有一栈，元素A，B，C，D只能依次进栈 ,则出栈序列中以下哪个是不可能得到的。(....)

A. D、C、B、A   
B. C、B、A、D  
C. A、B、C、D  
D. D、C、A、B

试写出队列中的出队算法；等等

**第四章：串(包括习题与答案及要点)**

本章介绍了串的逻辑结构，存储结构及串上的基本运算，由于在高级语言中已经提供了较全善的串处理功能，因此本章的重点是掌握在串上实现的模式匹配算法。同时这也是本章的难点。但是从全书来讲，这属于较简单的一章内容。

**1.串及其运算(领会)(这些内容比较容易理解，不用死记)**

串就是字符串，是一种特殊的线性表，它的每个结点仅由一个字符组成。

空串：是指长度为零的串，也就是串中不包含任何字符(结点)。

空白串：指串中包含一个或多个空格字符的串。不同与空串，它的结点就是一个空格字符。

在一个串中任意个连续字符组成的子序列称为该串的子串，包含子串的串就称为主串。子串在主串中的序号就是指子串在主串中首次出现的位置。如A="I love you" B="love",则B在A中的序号为3，注意空格也是字符。

空串是任意串的子串，任意串是他自身的子串。

串分为两种：串常量和串变量。串常量在程序中不能改变，串变量则可以。

关于串的基本运算，基本上在C语言中已经学过，主要有

求串长strlen(char \*s)、

串复制strcpy(char \*to,char \*from)、

串联接strcat(char \*to,char \*from)、

串比较charcmp(char \*s1,char \*s2)

和字符定位strchr(char \*s, char c)等

这些基本运算通过练习来掌握。

2.串的存储结构(简单应用)

串是特殊的线性表(结点是字符)，所以串的存储结构与线性表的存储结构类似。

串的顺序存储结构简称为顺序串，顺序串又可按存储分配的不同分为静态存储分配的顺序串和动态存储分配的顺序串。

静态的意思可简单地理解为一个确定的存储空间，它的长度是不可变的。如直接使用定长的字符数组来定义一个串。它的优点是涉及串长的操作速度快，因为它的最大长度是不变的。

动态存储分配就是在定义串时不分配存储空间，直到需要使用时按所需串的长度分配存储单元给它，并且在运行中还可以根据需要变化串的长度，这就是动态分配。不过这样的串仍是顺序存储的，也就是说指针指向串的首地址，后面的结点是连续存储的。

串的链式存储就是用单链表的方式存储串值，串的这种链式存储结构简称为链串。链串与单链表的差异只是它的结点数据域为单个字符。这种存储结构方便于串的插入和删除操作，但是空间利用率不高，因为存放每一个字符要"搭配"一个指向下一字符的地址,而地址所占空间是比较大的。为了解决这种"存储密度"过低的状况，可以让一个结点存储多个字符，事实上这是顺序串和链串的综合(折衷)。

本章的重点和难点就是串运算的实现，特别是顺序串上子串定位的运算。

子串定位运算又称串的"模式匹配"或"串匹配"，就是在主串中查找出子串出现的位置，这在应用中非常广泛，比如文本编辑中的"查找和替换"用到的就是子串定位运算的算法。

在串匹配中，将主串称为目标(串)，子串称为模式(串)，我们这样想象，子串就如同一个模板(样本)，用它在目标上对比，从头往后比较，凡是遇到一模一样的那么一段，就算找到一个位置了(我们就说，从这个位置开始的匹配成功)。用很专业的很酷的话说就是"模式在目标中出现"(我想起了警匪片里的对话),如果这个模板对应的目标串中有不一样的字符出现，那么这个位置就匹配失败。

当我们用这个模子依次从目标的头部往后移,移动到的位置就叫位移，如果每次向右移动1格，那么每次的位移就加上1。

每次移动后要看看模板里的字符和目标中相应的字符是否相等，如果都相同，这次位移就叫有效位移(其实就是从这个位置开始的匹配成功)

另外有一个合法位移和不合法位移的概念，就是说，移动一个位置后，如果模板的最后一个字符还没有超出目标串中最后一个字符时，这个位移就是合法位移，如果超出了，那么就没有比较的意义了，这时就是不合法位移。

这是比较容易理解的，串匹配问题就是找出给定模式串P在给定目标串T中首次出现的有效位移或者是全部有效位移。

具体的串匹配算法也不是很难理解，就是用两个循环，外循环用于进行模式的位移，内循环进行模板内每个字符的比较(判断是否有效位移)。关于串匹配的时间复杂度，在最坏的情况下就是目标串和模式串分别是"a^n-1b"和"a^m-1b"的形式，就是说，每一次合法位移后，在内循环中都要比较m个字符才知道是不是有效位移(前面的字符都是一样的)。所以最坏的情况下时间复杂度是O((n-m+1)m),假如m与n同阶的话则它是O(n^2)。

链串上的子串定位运算的不同之处就是位移是结点地址而不是整数。理解一下算法即可。

真正的应用主要还是要掌握串的基本算法并用它们构造出可以解决具体问题的简单算法。

第四章 串 复习要点

本章复习要点是：

串是一种特殊的线性表，它的结点仅由一个字符组成。

空串与空白串的区别：空串是长度为零的串，空白串是指由一个或多个空格组成的串。

串运算的实现中子串定位运算又称串的模式匹配或串匹配。

串匹配中，一般将主串称为目标(串),子串称为模式(串)。

本章可能出的题型多半为选择、填空等。