全国硕士研究生入学统一考试

计算机学科专业基础综合考试大纲解析

2010年版

● 本书编写组

高等教育出版社
HIGHER EDUCATION PRESS

全国硕士研究生入学统一考试

计算机学科专业基础综合考试大纲解析

2010年版

● 本书编写组

高等教育出版社
HIGHER EDUCATION PRESS

内容简介

本书由全国计算机专业排名领先的清华大学、中国科学院研究生院、国防科技大学等名校的资深教授、专家和一线教学骨干等组成的强大作者队伍精心打造，力求准确、精炼、系统地阐述大纲规定的知识点，注重复习的系统性并与学生实际相结合，深入浅出，不仅让考生学懂学会，还给出大量例题和习题让考生学练结合，达到举一反三、事半功倍的复习效果。

图书在版编目（CIP）数据

全国硕士研究生入学统一考试计算机学科专业基础综合考试大纲解析：2010年版/本书编写组.—北京：高等教育出版社，2009.8

ISBN 978-7-04-026797-6

Ⅰ.全… Ⅱ.本… Ⅲ.电子计算机—研究生—入学考试—自学参考资料 Ⅳ.TP3

中国版本图书馆CIP数据核字（2009）第118107号

策划编辑 刘 佳　　**责任编辑** 何新权　　**封面设计** 王凌波

责任校对 金 辉　　**责任印制** 朱学忠

出版发行	高等教育出版社	**购书热线**	010-58581118
社　　址	北京市西城区德外大街4号	**咨询电话**	400-810-0598
邮政编码	100120	**网　　址**	http://www.hep.edu.cn
总　　机	010-58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经　　销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印　　刷	保定市中画美凯印刷有限公司	**畅想教育**	http://www.widedu.com
开　　本	787×1092　1/16	**版　　次**	2009年8月第1版
印　　张	33	**印　　次**	2009年8月第1次印刷
字　　数	1 120 000	**定　　价**	60.00元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

物料号　26797-00

郑重声明

出版前言

一、《全国硕士研究生入学统一考试计算机科学与技术学科联考计算机学科专业基础综合考试大纲2010年版》规定了2010年全国硕士研究生入学考试计算机科目的考试范围、考试要求、考试形式、试卷结构等。它既是2010年全国硕士研究生入学计算机专业考试命题的唯一依据，也是考生复习备考必不可少的工具书。

二、《全国硕士研究生入学统一考试计算机学科专业基础综合考试大纲解析2010年版》根据教育部制订的《考试大纲》的要求和最新精神，深入研究考研命题的特点及动态特别注重与学生的实际相结合，注重与考研的要求相结合。

本书由四个学科组成，包括数据结构、计算机组成原理、操作系统、计算机网络。其中各部分包括以下三部分：

（一）复习要点——使考生能明确本章的重点、难点及常考点，让考生弄清各知识点之间的相互联系，以及多年考试中本章节的出题情况，以便对本章内容有一个全局性的认识和把握。

（二）考点精讲——本部分参考当前国内最权威的大学教材，对大纲所要求的知识点进行了全面、准确地阐述，以加深考生对基本概念和原理等重点内容的理解和正确应用。本部分讲解考点明确、重点突出、层次清晰、简明实用。

（三）例题与练习——通过对经典例题的分析教会考生分析问题解决问题的方法和技巧。通过大量练习题，使考生学练结合，更好地巩固所学知识，提高实战能力。

本书由清华大学殷人昆教授主编并审定，清华大学王诚、董长洪老师，中国科学院研究生院鲁士文老师，国防科技大学邹鹏、尹俊文等老师分别对所负责章节进行了认真的研究和撰写，在此对他们严谨的治学态度和付出的智慧与努力表示感谢！

为了给考生提供更多的增值服务，凡购正版全国考研辅导班系列用书的考生都可以登录“中国教育考试在线”www.eduexam.com.cn在线做考研全真模拟试卷。

高等教育出版社
2009年7月

目 录

第一部分　数据结构

☑考试要求

2010年大纲中明确提出，对于“数据结构”部分，主要考查：

（1）理解数据结构的基本概念；掌握数据的逻辑结构、存储结构及其差异，以及各种基本操作的实现。

（2）在掌握基本的数据处理原理和方法的基础上，能够对算法进行时间复杂度和空间复杂度分析。

（3）能够选择合适的数据结构和方法进行问题求解；具备采用C或C++或Java语言设计与实现算法的能力。

换句话说，考查的目标有两个：知识和技能。

1. 知识方面

从数据结构的结构定义和使用，以及存储表示和操作的实现两个层次，系统地考查：

（1）掌握常用的基本数据结构（包括顺序表、链接表、栈与队列、数组、二叉树、堆、树与森林、图、查找结构、索引结构、散列结构）及其不同的实现。

（2）掌握分析、比较和选择不同数据结构、不同存储结构、不同算法的原则和方法。

2. 技能方面

（1）系统地掌握基本数据结构的设计方法。

（2）掌握选择结构的方法和算法设计的思考方式及技巧，提高分析问题和解决问题的能力。

第1章 线性表

数据(data)是信息的载体,是描述客观事物属性的数、字符以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。数据结构是指数据对象及其相互关系和构造方法。一个数据结构可用一个三元组表示 Data_Structure = {D, R, A},D 是某一具有相同性质的数据元素的集合,R 是该集合中所有数据元素之间的关系的有限集合,A 是该数据元素集合可提供服务(操作)的集合。

数据结构中结点与结点之间的逻辑关系称为数据的逻辑结构,它是面向使用的。按数据的逻辑关系,数据结构可分为线性结构和非线性结构,其中非线性结构又可分为树形结构和图结构。一种特殊的数据结构称为集合。集合结构从逻辑上看,元素之间没有关系,但从实现上来看,它可以用线性结构或非线性结构来表示。

数据结构在计算机中的存储映像称为数据的存储结构。典型的存储方式有 4 种,包括顺序存储结构、链式存储结构、索引存储结构、散列存储结构。

算法与数据结构之间有密切的关系。算法建立在数据结构的基础上,选择合理的数据结构可有效地改进算法的效率。算法是过程性的,按输入—计算—输出的模式解决问题。对算法的分析主要着眼于它的时间代价和空间代价,需要熟练掌握的是大 O 表示法。

1. 线性表的概念:包括线性表的定义和特点,线性表的基本操作。
2. 线性表的存储表示:包括顺序表的定义及基本运算的实现,单链表的定义及基本运算的实现。
3. 线性表的特殊链接表示:循环链表的特殊遍历方式,双向链表的方向性。
4. 线性表的应用:掌握使用线性表基本操作实现应用算法。

1) 在一维数组上的算法,如原地逆置、非零元素压缩、成块元素移动等。

2) 在一维数组上的递归算法:如求和、平均值等。

3) 在顺序表上的查找、插入、删除、合并、求交等算法及其性能分析。

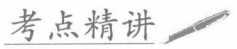
4) 在单链表上的迭代求解算法及性能,包括统计链表结点个数、在链表中寻找与给定值 x 匹配的结点、在链表中寻找第 i 个结点、链表逆转等。

5) 带表头结点的单链表的迭代算法,包括统计链表结点个数、在链表中寻找与给定值 x 匹配的结点、在链表中寻找第 i 个结点、两个有序链表的合并等。

6) 单链表的递归算法,包括统计链表结点个数、在链表中寻找与给定值 x 匹配的结点、在链表中寻找第 i 个结点、求链表各结点值的和、平均值等。

7) 循环链表的迭代算法、双向链表的迭代算法。

5. 多项式的建立,两个多项式的相加,两个多项式的相乘算法。



1.1 线性表的定义和基本操作

1.1.1 线性表的定义

通常,定义线性表为 n 个数据元素(或称为表元)的有限序列。记为 $L = (a_1, a_2, \cdots, a_n)$。其中,$L$ 是表名,a_i 是表中的结点,是不可再分割的数据。n 是表中表元的个数,也称为表的长度,若 $n = 0$ 叫做空表。线性表的特点是,在非空的数据元素集合中:

- 存在唯一的一个称作“第一个”的元素;
- 存在唯一的一个称作“最后一个”的元素;

- 除第一个元素外,集合中的每个元素均只有一个直接前驱;
- 除最后一个元素外,集合中的每个元素均只有一个直接后继。

其中,最后两个特点体现了线性表中元素之间的逻辑关系。

理解线性表的要点是:

1. 表中元素具有逻辑上的顺序性,在序列中各元素排列有其先后次序。
2. 表中元素个数有限。
3. 表中元素都是数据元素。就是说,每一个表元素都是不可再分的原子数据。
4. 表中元素的数据类型都相同。这意味着每一个表元素占有相同数量的存储空间。
5. 表中元素具有抽象性。就是说,仅讨论表元素之间的逻辑关系,不考虑元素究竟表示什么内容。

1.1.2 线性表的操作

线性表的主要操作有:

1. 表的初始化运算:将线性表置为空表。
2. 求表长度运算:统计线性表中表元素个数。
3. 查找运算:查找线性表中第 i 个表元素或查找表中具有给定关键字值的表元素。
4. 插入运算:将新表元素插入到线性表第 i 个位置上,或插入到具有给定关键字值的表元素的前面或后面。
5. 删除运算:删除线性表第 i 个表元素或具有给定关键字值的表元素。
6. 读取运算:读取线性表第 i 个表元素的值。
7. 复制运算:复制线性表所有表元素到另一个线性表中。

主要操作的实现取决于采用哪一种存储结构。存储结构不同,实现的算法也不同。

1.2 线性表的实现

1.2.1 线性表的顺序存储

线性表的顺序存储又称为顺序表。它用一组地址连续的存储单元依次存储线性表中的数据元素,从而使得逻辑关系相邻的两个元素在物理位置上也相邻。因此,顺序表的特点是表中各元素的逻辑顺序与其物理顺序相同。

用 C 语言描述时借用一个一维数组来存储这些表元素。

程序 1.1 顺序表的静态存储分配。

```
#define maxSize 100                    //显式地定义表的长度
typedef int DataType;                  //定义表元素的数据类型
typedef struct {                       //顺序表的定义
    DataType data[maxSize];            //静态分配存储表元素的数组
    int n;                             //实际表元素个数,0≤n≤maxSize
} SeqList;
```

在这种存储方式下,表元素 a_i 存储在 data[i-1]位置。存储结构如图 1.1 所示。

下标位置	0	1	…	$i-1$	i	…	$n-1$	…	maxSize - 1
数组(线性表)存储空间	a_1	a_2	…	a_i	a_{i+1}	…	a_n	…	…

图 1.1 顺序表的示意图

假设顺序表 A 的起始存储位置为 $Loc(1)$,第 i 个表项的存储位置为 $Loc(i)$,则有:

$$Loc(i)=Loc(1)+(i-1)\times \text{sizeof(DataType)}$$

其中,$Loc(1)$是第一个表项的存储位置,即数组中第 0 个元素位置。sizeof(DataType)是表中每个元素所占空间的大小。根据这个计算关系,可随机存取表中的任一个元素。

一维数组可以是静态分配的,也可以是动态分配的。在静态分配存储的情形下,由于数组的大小和空间事先已经固定分配,一旦数据空间占满,再加入新的数据就将产生溢出,此时存储空间不能扩充,就会导致程序停止工作。而在动态分配存储的情形下,存储数组的空间是在程序执行过程中通过动态存储分配的语句分配的,一旦数据空间占满,可以另外再分配一块更大的存储空间,用以代换原来的存储空间,从而达到扩充存储数组空

间的目的，同时需将表示数组大小的常量 maxSize 放在顺序表的结构内定义，可以动态地记录扩充后数组空间的大小，提高结构的灵活性。

程序 1.2 顺序表的动态存储分配。

```
#define initSize 100                          //表长度的初始定义
typedef int DataType;                         //定义表元素的数据类型
typedef struct {                              //顺序表的定义
    DataType *data;                           //指示动态分配数组的指针
    int maxSize,n;                            //数组的最大容量和当前个数
} SeqList;
```

初始的动态分配语句为：

```
data=(DataType *) malloc (sizeof (DataType) * initSize);
//C++要简单得多，是 data=new Data Type[initSize];以后采用此种描述
maxSize=initSize;  n=0;
```

1.2.2 线性表的链式存储

线性表的链式存储又称为线性链表。在这种结构中数据元素存储在结点中，结点之间在空间上可以连续，也可以不连续，通过结点内附的链接指针来表示元素之间的逻辑关系。因此，在线性链表中逻辑上相邻的表元素在物理上不一定相邻。

最简单的线性链表是单链表，用 C 语言描述如下：

程序 1.3 单链表的定义。

```
typedef int DataType;
typedef struct node {
    DataType data;
    struct node *link;
} LinkedNode, *LinkList;
```

此时，使用一个指向链表结点的指针 htp 标识链表的表头：

LinkedNode * hpt 或 LinkList hpt

为了表示链表收尾，链表最后一个结点的链接指针应置为空。

1.2.3 顺序存储与链式存储的比较

从访问方式来看，顺序表可以顺序存取，也可以直接存取（注意它与一维数组的区别），线性链表只能从链头顺序存取。

从表元素的逻辑顺序与物理位置的对应关系来看，顺序表中表元素的逻辑顺序与它们的物理存储顺序是完全相同的，而线性链表中各个表元素的逻辑顺序与物理存储顺序不一定相同。

从存储空间的利用率来看，若定义存储密度为：存储密度 = 表中数据元素占有的空间/分配给表的总空间，则顺序表的存储密度为 1，表示数据元素之间的逻辑关系无须占用附加空间；而线性链表的存储密度小于 1，因为每个数据元素都需附加一个链接指针以表示元素之间的逻辑关系。

从查找速度来看，由于线性链表只能沿链逐个比较，而顺序表可以按照元素序号（下标）直接访问，故顺序表查找速度比线性链表要快；从插入和删除速度来看，如果要求插入和删除后表中其他元素的相对逻辑顺序保持不变，则顺序表平均需要移动大约一半元素，而线性链表只需修改链接指针，不需要移动元素，因此线性链表比顺序表的插入和删除速度快。

从 C 指针的使用来看，顺序表的情形下，指针 p 指示数据元素存储位置，用 *p 可取得该数据的值，用 p++可以顺序进到物理上下一个数据元素的位置；在线性链表的情形下，指针 p 指示链表结点的地址，用 *p 不能取得该结点数据的值，用 p++也不能进到下一个结点位置，只能使用 p->data 取得结点数据的值，用 p = p->link 进到下一个结点。

从空间限制来看，顺序表在静态存储分配的情形下，一旦存储空间装满不能扩充，如果再加入新元素将出现存储溢出；在动态存储分配的情形下虽然存储可以扩充，但需要移动大量元素，将导致操作效率降低。线性链表的结点空间只有在需要的时候才申请，无需事先分配，因此，只要还有空间可分配，就没有存储溢出问题，操作效

率也优于顺序表。

1.2.4 其他线性链表的形式

根据结点中指针信息的实现方式，还有其他几种链表结构：

1. 双向链表：每个结点包含两个指针，指明直接前驱和直接后继元素，可在两个方向上遍历其后及其前的元素。

2. 循环链表：表尾结点的后继指针指向表中的第一个结点，可在任何位置上遍历整个链表。

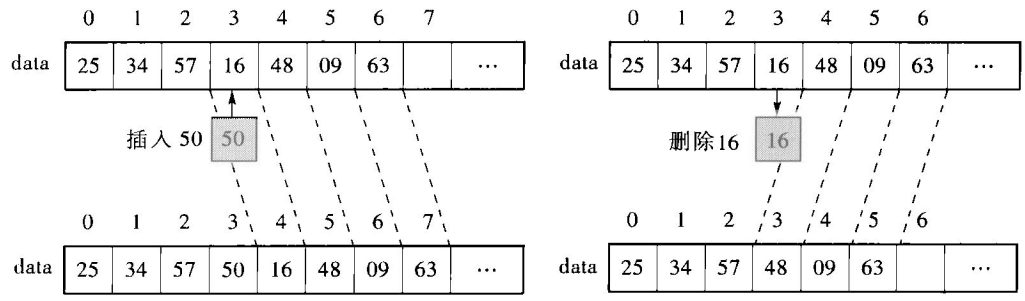
3. 静态链表：借助数组来描述线性表的链式存储结构。

在链式存储结构中，只需要一个指针（头指针）指向第一个结点，就可以顺序访问到表中的任意一个元素。为了简化对链表状态的判定和处理，特别引入一个不存储数据元素的结点，称为头结点，将其作为链表的第一个结点并令头指针指向该结点。

1.3 线性表的插入和删除运算

1.3.1 基于顺序存储结构的运算

如果在插入和删除后不需要保持表中元素的原有逻辑顺序，则可直接把新元素插入（或追加）到表尾，或把表尾元素直接覆盖表中的被删除元素，其平均移动元素个数为1。如果在插入和删除后需要保持表中元素的原有逻辑顺序，则在插入元素前需要移动元素以挪出空的存储单元，然后再插入元素；删除元素时同样需要移动元素，以填充被删元素空出来的存储单元。如图1.2所示。



（a）插入新元素的示例　　（b）删除表中已有元素的示例

图1.2　表项的插入与删除

在等概率下平均移动元素的次数分别为：

$$E_{\text{insert}} = \sum_{i=1}^{n+1} p_i \times (n-i+1) = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$$

$$E_{\text{delete}} = \sum_{i=1}^{n} q_i \times (n-i) = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{n-1}{2}$$

1.3.2 基于链式存储结构的运算

在链式存储结构下进行插入和删除，其实质都是对相关指针的修改。下面给出单链表上的查找、插入和删除运算的实现算法。

程序1.4　单链表的查找运算。

```
LinkList Find_List (LinkList L,int k) {
//L 为带头结点单链表的头指针。算法在表中查找第 k 个元素，若找到，则算法返回
//该元素结点的指针，否则算法返回空指针 NULL。
    LinkList p;   int i;
    i=1;p=L->link;                    //初始时 p 指向第一个元素结点，i 为计数器
    while (p ! =NULL && k < i) {      //顺链接指针逐个向后查找
        p=p->link;   i++;
    }
```

```
    if (p != NULL && i==k) return p;        //存在第 k 个元素且 p 指向该元素结点
    return NULL;                            //第 k 个元素不存在
};
```

程序 1.5 单链表的插入运算。

```
bool Insert_List (LinkList L,int k,int elem){
//L 为带头结点单链表的头指针。算法将元素 elem 插入表中的第 k 个元素之前,若
//成功则算法返回 true,否则返回 false。
    LinkList p,q;
    if (k==1) p=L;                          //元素 elem 插入在第 1 个元素之前
    else p=Find-List (L,k-1);               //查找表中的第 k-1 个元素
    if (p==NULL) return false;              //表中不存在第 k-1 个元素
    q=new LinkedNode;
    if (q==NULL) return false;              //结点存储分配失败
    q->data=elem;
    q->link=p->link; p->link=q;             //元素 elem 插入第 k-1 个元素之后
    return true;
};
```

程序 1.6 单链表的删除运算。

```
bool Delete_List (LinkList L,int k){
//L 为带头结点单链表的头指针。算法删除表中的第 k 个元素结点,若成功则算法
//返回 true,否则算法返回 false。
    LinkList p,q;
    if (k==1) p=L;                          //删除第一个元素结点
    else p=Find_List (L,k-1);               //查找表中的第 k-1 个元素
    if (p==NULL) return false;              //表中不存在第 k-1 个元素
    q=p->link;                              //令 q 指向第 k 个元素结点
    p->link=q->link; delete q;              //删除,C++的 delete q 相当于 C 的 free(q)
    return true;
};
```

双向链表的查找、插入和删除运算需要考虑前驱和后继指针的修改,实现算法请读者自行复习。

例题精解

一、单选填空题

例 1 在线性表中的每一个表元素都是数据对象,它们是不可再分的____。

A. 数据项　　B. 数据记录　　C. 数据元素　　D. 数据字段

【解答】 C。线性表是 $n(n\geq 0)$ 个数据元素的有限序列。数据记录、数据字段是数据库文件组织中的术语。数据项相当于数据元素中的属性。

例 2 数据结构反映了数据元素之间的结构关系。单链表是一种__1__,它对于数据元素的插入和删除__2__。

(1) A. 顺序存储线性表　　B. 非顺序存储非线性表

C. 顺序存储非线性表　　D. 非顺序存储线性表

(2) A. 不需移动结点,不需改变结点指针　　B. 不需移动结点,只需改变结点指针

C. 只需移动结点,不需改变结点指针　　D. 既需移动结点,又需改变结点指针

【解答】 (1) A, (2) B。数据结构通常指的是数据的逻辑结构,它反映了数据元素之间的逻辑关系,单链表属于线性表的一种存储结构,它的每个结点有两个域(data, link),data 域存放元素值,link 域存放表中逻辑上相邻的下一结点的地址指针。通过结点指针,线性表中所有元素按照逻辑顺序链接起来,而在物理上,只要有空间就可以分配给结点使用,结点之间的存储位置可以与逻辑顺序不一致,所以单链表是一种非顺序存储的线性

表。在单链表中删除或插入元素比较方便，无需改变结点的存储位置，只要修改几个结点的指针即可。

例3 通常查找线性表数据元素的方法有__1__和__2__两种方法，其中__1__是一种只适合于顺序存储结构但__3__的方法；而__2__是一种对顺序和链式存储结构均适用的方法。

(1) A. 顺序查找　　B. 循环查找　　C. 条件查找　　D. 折半查找

(2) A. 顺序查找　　B. 随机查找　　C. 折半查找　　D. 分块查找

(3) A. 效率较低的线性查找　　B. 效率较低的非线性查找

C. 效率较高的非线性查找　　D. 效率较高的线性查找

【解答】 (1) D，(2) A，(3) C。在线性表中查找指定元素常用顺序查找法和折半查找法。顺序查找法属于线性查找，效率较低，但它适用于用顺序方式或用链接方式存储的线性表；折半查找法仅适用于已排序的顺序存储线性表，每次根据查找值的大小将查找区间缩小一半继续查找，因此它不是线性查找，它比顺序查找的效率高一些。

例4 顺序表是线性表的____存储表示。

A. 有序　　B. 连续　　C. 数组　　D. 顺序存取

【解答】 C。顺序表是线性表的数组存储表示，也称为线性表的顺序存储结构。注意，顺序存取是一种读写方式，不是存储方式，有别于顺序存储。

例5 若设一个顺序表的长度为 n。那么，在表中顺序查找一个值为 x 的元素时，在等概率的情况下，查找成功的数据平均比较次数为__1__。在向表中第 i 个元素($1 \leqslant i \leqslant n+1$)位置插入一个新元素时，为保持插入后表中原有元素的相对次序不变，需要从后向前依次后移__2__个元素。在删除表中第 i 个元素($1 \leqslant i \leqslant n$)时，同样地，为保持删除后表中原有元素的相对次序不变，需要从前向后依次前移__3__个元素。

(1) A. n　　B. $n/2$　　C. $(n+1)/2$　　D. $(n-1)/2$

(2) A. $n-i$　　B. $n-i+1$　　C. $n-i-1$　　D. i

(3) A. $n-i$　　B. $n-i+1$　　C. $n-i-1$　　D. i

【解答】 (1) C，(2) B，(3) A。在长度为 n 的顺序表中，若各元素查找概率相等，则查找成功的平均查找长度为：

$$ASL_{成功} = \frac{1}{n}\sum_{i=0}^{n-1}(i+1) = \frac{1}{n}(1+2+\cdots+n) = \frac{1}{n}\frac{(1+n)n}{2} = \frac{n+1}{2}$$

在有 n 个元素的顺序表中的第 i 个元素位置插入一个新元素时，需把表中从第 n 个到第 i 个的元素全部后移一个元素位置，以空出第 i 个元素位置供新元素插入，需要移动的元素有 $n-i+1$ 个，剩下的前 $n-1$ 个元素没有移动。

而想要在有 n 个元素的顺序表中删除第 i 个元素，须把第 $i+1$ 个元素到第 n 个元素全部前移，以填补原来第 i 个元素，需要移动 $n-(i+1)+1 = n-i$ 个元素。

例6 若在长度为 n 的顺序表的表尾插入一个新元素的渐进时间复杂度为____。

A. $O(n)$　　B. $O(1)$　　C. $O(n^2)$　　D. $O(\log_2 n)$

【解答】 B。在有 n 个元素的顺序表的表尾插入一个新元素，可直接在表的第 $n+1$ 个位置插入，渐进时间复杂度为 $O(1)$。

例7 设单链表中结点的结构为

```
typedef struct node {                    //链表结点定义
    ElemType data;                       //数据
    struct node * link;                  //结点后继指针
    } LinkedNode;
```

不带头结点的单链表 first 为空的判定条件是__1__：

(1) A. first == NULL;　　B. first->link == NULL;

C. first->link == first;　　D. first ! = NULL;

带头结点的单链表 first 为空的判定条件是__2__：

(2) A. first == NULL;　　B. first->link == NULL;

C. first->link == first;　　D. first ! = NULL;

已知单链表中结点 *q 是结点 *p 的直接前驱，若在 *q 与 *p 之间插入结点 *s，则应执行以下__3__操作：

(3) A. s->link = p->link;　p->link = s;　　B. q->link = s;　s->link = p;

C. p->link = s->link; s->link = p; D. p->link = s; s->link = q;

已知单链表中结点 *p 不是链尾结点,若在 *p 之后插入结点 *s,则应执行下列__4__操作。

(4) A. s->link = p; p->link = s; B. p->link = s; s->link = p;

C. s->link = p->link; p = s; D. s->link = p->link; p->link = s;

若想在单链表中摘除结点 *p(*p 既不是第一个也不是最后一个结点)的直接后继,则应执行以下__5__操作。

(5) A. p->link = p->link->link; B. p = p->link; p->link = p->link->link;

C. p->link = p->link; D. p = p->link->link;

非空的循环单链表 first 的链尾结点(由 p 所指向)满足__6__:

(6) A. p->link == NULL; B. p == NULL;

C. p->link == first; D. p == first;

设 rear 是指向非空的带表头结点的单循环链表的链尾结点的指针。若想删除链表第一个结点,则应执行以下__7__操作。

(7) A. s = rear; rear = rear->link; delete s;

B. rear = rear->link; delete rear;

C. rear = rear->link->link; delete rear;

D. s = rear->link->link; rear->link->link = s->link; delete s;

设双向循环链表中结点的结构为(data, lLink, rLink),且不带表头结点。若想在结点 *p 之后插入结点 *s,则应执行以下__8__操作。

(8) A. p->rLink = s; s->lLink = p; p->rLink->lLink = s; s->rLink = p->rLink;

B. p->rLink = s; p->rLink->lLink = s; s->lLink = p; s->rLink = p->rLink;

C. s->lLink = p; s->rLink = p->rLink; p->rLink = s; p->rLink->lLink = s;

D. s->lLink = p; s->rLink = p->rLink; p->rLink->lLink = s; p->rLink = s;

【解答】 (1) A, (2) B, (3) B, (4) D, (5) A, (6) A, (7) D, (8) D

(1), (2) 若单链表不带表头结点, *first 即为首元结点(第一个结点),链表为空即 first 为空;若单链表带有表头结点, *first 即为表头结点,链表为空即表头结点后面没有首元结点,first->link 为空。

(3) 已知单链表中结点 *q 是结点 *p 的直接前驱,若在 *q 与 *p 之间插入结点 *s,需要把 *s 链接到 *q 之后,把 *p 链接到 *s 之后:q->link = s; s->link = p。

(4) 已知单链表中结点 *p 不是链尾结点,若在 *p 之后插入结点 *s,需要把原来 *p 后的结点先链接到 *s 之后,再把 *s 链接到 *p 之后:s->link = p->link; p->link = s。

(5) 若想在单链表中摘除结点 *p(*p 既不是第一个也不是最后一个结点)的直接后继,需要做重新链接工作,让 *p 的后继的后继成为 *p 的后继:p->link = p->link->link。

(6) 非空的循环单链表 first 的链尾结点为 *p,则 *p 的后继为空:p->link == NULL。

(7) 设 rear 是非空带表头结点的单循环链表的链尾指针。若想删除链表第一个结点,必须先保存要删除的表头结点的后继结点地址:s = rear->link->link;再做重新链接工作,让 *s 的后继链接到表头结点之后:rear->link->link = s->link;最后做删除:delete s。

(8) 若想在不带表头结点的双向循环链表中结点 *p 之后插入结点 *s,需在两个链上做插入,插入过程中要小心,不要让其中任何一个链断掉。选项 A 和选项 B 首先排除,因为第一条语句 p->rLink = s 就让后继链断掉了。选项 C 和选项 D 前两条语句 s->lLink = p; s->rLink = p->rLink 合理,让 *s 的前驱、后继都链接好且未影响原来的两个链,但选项 C 的第三条语句 p->rLink = s 又断开了后继链,排除它就剩下选项 D,它是对的。

二、综合应用题

例 1 线性表的每一个表元素是否必须类型相同?为什么?

【解答】 线性表每一个表元素的数据空间要求相同,但如果对每一个元素的数据类型要求不同时可以用等价类型(union)变量来定义可能的数据元素的类型。如

```
typedef union {                    //联合
    int integerInfo;               //整型
    char charInfo;                 //字符型
```

```
    float floatInfo;                    //浮点型
} info;
```

利用等价类型,可以在同一空间(空间大小相同)info 中存放不同数据类型的元素。但要求用什么数据类型的变量存的,就必须以同样的数据类型来取它。

例 2(2009 年全国统考真题) 已知一个带表头结点的单链表,结点的结构为(data,link)。假设该链表只给出了表头指针 list,在不改变链表的前提下请设计一个尽可能有效的算法,查找链表中倒数第 k 个位置上的结点(k 为正数)。若查找成功,算法输出该结点的 data 域的值,并返回 1,否则只返回 0。要求:

(1) 描述该算法的基本设计思想;

(2) 描述该算法的详细实现步骤;

(3) 根据算法的基本设计思想和详细实现步骤,采用程序设计语言描述算法,关键之处请给出简要注释。

【解答】 (1) 算法的基本设计思想。

问题的关键是设计一个尽可能高效的算法,通过链表的一趟遍历,找到倒数第 k 个结点的位置。算法的基本设计思想是:定义两个遍历指针 p 和 q。初始时均指向表头结点的下一个结点(即链表的第一个结点)。首先让指针 p 移动到链表第 k 个结点,然后指针 q 与指针 p 同步移动;当指针 p 移动到链表最后一个结点时,指针 q 所指示的结点就是倒数第 k 个结点的位置。

(2) 算法的详细实现步骤。

① 定义指针 p 和指针 q,让 p = q = list->link; 定义计数器 count = 1;

② 当 p->link == NULL 时转移到⑤,否则重复下列③、④步。

③ 如果 count < k 时执行 count = count + 1;

否则 q = q->link;

④ p = p->link;

⑤ 如果 count < k 表明 k 值太大超过了表的长度,函数返回 0。

否则输出 q 指针所指结点的 data 值并返回 1。

⑥ 算法结束。

(3) 用 C 语言描述算法如下:

```
typedef int Type;                                   //链表数据的类型定义
typedef struct ListNode {                           //链表结点的结构定义
    Type data;                                      //结点数据
    Struct ListNode *link;                          //结点链接指针
} *LinkedList;
int Search_K ( LinkedList list, int k ) {
    ListNode *p = list->link, *q = list->link;      //指针 p、q 置于链表前端
    int count = 0;
    while ( p != NULL ) {                           //遍历链表到最后一个结点
        if ( count < k ) count++;                   //让 p 移动到第 k 个结点
        else q = q->link;                           //之后让 p、q 同步移动
        p = p->link;
    }
    if ( count < k ) return 0;                      //查找失败返回 0
    else { printf ("%d", q->data );  return 1; }    //否则打印并返回 1
}
```

此算法的时间代价为 $2n-k$,空间代价是使用了一个计数器 count。除此之外,还有一些解法,如:

① 两趟遍历

此算法分两步走:第一趟遍历计算表的长度 n,如果 $n-k+1 < 1$ 则返回 0;第二趟遍历停止于第 $n-k+1$ 个结点位置,它就是倒数第 k 个结点,输出值并返回 1。

```
int Search_K ( LinkedList list, int k ) {
```

```
    ListNode *p = list->link, *q = list->link;
    int n = 0;
    while ( p != NULL ) { p = p->link;  n++; }    //统计表的长度 n
    if ( n-k+1 < 1 ) return 0;                    //不存在倒数第 k 个结点
    while ( n-k > 0 ) { q = q->link;  n--; }      //寻找倒数第 k 个结点
    printf ("%d", q->data );  return 1;
}
```

此算法的时间代价也为 $2n-k$,空间代价是使用了一个计数器 n。

② 使用一个与链表长度同样大小的辅助数组

此算法对单链表做一趟遍历,依次读出各结点的 data 值并从下标位置 1 开始顺序存入该辅助数组,同时计算链表长度 n,如果 $n-k+1<1$ 则返回 0,否则直接输出辅助数组中第 $n-k+1$ 个元素的值并返回 1。

```
#define maxSize 100                               //预估一个辅助数组容量
int Search_K ( LinkedList list, int k ) {
    Type X[maxSize];
    ListNode *p = list->link;  int n = 0;
    while ( p != NULL ) { X[n++] = p->data;  p = p->link; }
    if ( n-k+1 < 1 ) return 0;                    //不存在倒数第 k 个结点
    printf ("%d", X[n-k] );  return 1;
}
```

此算法的时间代价为 n,空间代价也是 n,使用了一个辅助数组 $X[\,]$。

③ 使用一个与链表长度同样大小的栈

此算法对单链表做一趟遍历,依次读出各结点的 data 值并存入初始为空的栈中;然后逐个元素出栈,直到退出第 k 个栈元素为止。如果还未退到第 k 个元素栈就空了,则返回 0,否则输出退出的第 k 个栈元素的值并返回 1。

```
#include "Stack.h"
int Search_K ( LinkedList list, int k ) {
    ListNode *p = list->link;  int n = 0;
    while ( p != NULL ) { p = p->link;  n++; }
    Stack S[n];  Type Result;
    p = list->link;  initStack(S);
    while ( p != NULL ) { Push(S, p->data);  p = p->link; }
    while ( k > 0 && ! StackEmpty(S) ) { Pop(S, Result);  k--; }
    if ( k == 0 ) { printf ("%d", Result );  return 1; }
    return 0;
}
```

此算法的时间代价为 $n+k$,空间代价是 n,使用了一个辅助栈 S。

④ 使用一个可保存 k 个元素的一维数组

此算法对单链表做一趟遍历,依次读出各结点的 data 值并存入初始为空的数组中。如果链表所有数据都存入数组而数组不满,则返回 0;否则将链表数据循环存入数组中,当链表所有数据都处理完,则最后存入数组元素的下一个即为链表的倒数第 k 个元素,输出它的值并返回 1。

```
int Search_K ( LinkedList list, int k ) {
    Type X[k];  int i = 0;
    ListNode *p = list->link;
    while ( p != NULL && i < k ) { X[i] = p->data;  p = p->link;  i++; }
    if ( p == NULL && i < k ) return 0;           //找不到倒数第 k 个结点返回 0
    while ( p != NULL ) { X[i % k] = p->data;  p = p->link;  i++; }
    printf ("%d", X[i % k];  return 1;
```

```
}
```

此算法的时间代价为 n,空间代价是 k,使用了一个长度为 k 的辅助一维数组。

⑤ 递归算法

单链表递归算法利用了单链表结构的递归特性,即单链表的链接指针非空,则指向了一个非空的单链表;单链表的链接指针空,则指向了一个空的单链表。本算法的目的是通过递归,逐步找到链表的尾结点。然后,在退出递归的过程中计数,直到退到倒数第 k 个结点为止。

```
int Find_K ( ListNode *t, ListNode *& s, int k, int& i ) {
//指针 t 是递归调用的链表头指针,指针 s 返回倒数第 k 个结点地址,i 返回倒数
//第几个结点,函数返回是否找到的标志( =0 未找到, =1 找到)
    if ( t == NULL ) { i = 0; return 0; }              //递归结束
    else {
        int j = Find_K( t->link, s, k, i );            //对后续链表递归查找
        if ( j == 1 ) return 1;                        //返回 1 表明已找到,继续回退
        else if ( ++i == k ) { s = t; return 1; }      //找到倒数第 k 个结点,用 s 记忆
        else return 0;                                 //未找到
    }
}
int Search_K ( LinkedList list, int k ) {
//外部调用递归算法求倒数第 k 个结点
    ListNode *s;  int i, j;
    j = Find_K ( list->link, s, k, i );
    if ( j ) { printf ( "%d", s->data ); return 1; }
    else return 0;
}
```

此算法的时间代价为 $n+k$,空间代价是隐含了一个长度为 n 的递归栈。

同步练习

一、单项选择题

1. 在下列关于线性表的叙述中,正确的是____。

A. 线性表的逻辑顺序与物理顺序总是一致的。

B. 线性表的顺序存储表示优于链式存储表示。

C. 线性表若采用链式存储表示时所有存储单元的地址可连续可不连续。

D. 每种数据结构都应具备三种基本运算:插入、删除和查找。

2. 若长度为 n 的非空线性表采用顺序存储结构,在表的第 i 个位置插入一个数据元素,i 的合法值应该是____。

A. $i>0$　　B. $1\leqslant i\leqslant n$　　C. $0\leqslant i\leqslant n-1$　　D. $0\leqslant i\leqslant n$

3. 对于顺序存储的线性表,其算法的时间复杂度为 $O(1)$ 的运算应是____。

A. 将 n 个元素从小到大排序　　B. 从线性表中删除第 i 个元素($1\leqslant i\leqslant n$)

C. 查找第 i 个元素($1\leqslant i\leqslant n$)　　D. 在第 i 个元素($1\leqslant i\leqslant n$)后插入一个新元素

4. 已知 L 是带表头的单链表,L 是表头指针,则摘除首元结点的语句是____。

A. L = L->link;　　B. L->link = L->link->link;

C. L = L->link->link;　　D. L->link = L;

5. 从一个具有 n 个结点的有序单链表中查找其值等于 x 的结点时,在查找成功的情况下,需要平均比较的结点个数为____。

A. n　　B. $n/2$　　C. $(n-1)/2$　　D. $(n+1)/2$

6. 在一个具有 n 个结点的单链表中插入一个新结点并可以不保持原有顺序的算法的时间复杂度是____。

A. $O(1)$　　B. $O(n)$　　C. $O(n^2)$　　D. $O(n\log_2 n)$