2014年全国硕士研究生入学统一考试辅导用书

翔高教育计算机教学研究中心 编

2014

计算机学科专业基础综合

复习指南

- 全国**唯一**教材、面授、网授三位一体考研培训机构精心力作
- 每届考生平均每3位中**必有**1位使用过本书
- 连续4年国内同类计算机考研教学辅导书中销量稳居**第一**
- 国内5所计算机名校名师亲自执笔，提供更**权威**的大纲解析
- 凝聚**20**名资深专家，**7**个研究项目组，**1000**多天的心血结晶
- **收集**历年国内50余所名和科学院所考研真题，**筛选**典型题型
- 充分考虑学生应试中的薄弱环节，利于短期内**迅速**强化和提高
- 内容形式两创新，近百万字鸿篇巨制，**全面覆盖**考纲所有考点

复旦大学出版社

翔高教育 计算机教学研究中心 编

2014年全国计算机科学与技术学
科硕士研究生招生联考

2014计算机学科
专业基础综合复习指南

图书在版编目(CIP)数据

2014 计算机学科专业基础综合复习指南/翔高教育计算机教学研究中心编. —6 版.
—上海:复旦大学出版社,2013.8
2014 年全国硕士研究生入学统一考试辅导用书
ISBN 978-7-309-09896-9

Ⅰ.2… Ⅱ.翔… 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(2013)第 160236 号

2014 计算机学科专业基础综合复习指南
翔高教育计算机教学研究中心 编
责任编辑/张志军

复旦大学出版社有限公司出版发行
上海市国权路 579 号 邮编:200433
网址:fupnet@fudanpress.com http://www.fudanpress.com
门市零售:86-21-65642857 团体订购:86-21-65118853
外埠邮购:86-21-65109143
大丰市科星印刷有限责任公司

开本 787×1092 1/16 印张 30 字数 730 千
2013 年 8 月第 6 版第 1 次印刷

ISBN 978-7-309-09896-9/T·481
定价:60.00 元

Preface 前言

Preface

preface

2009年始，我们出版的计算机考研系列图书《复习指南》、《习题精编》、《模拟试卷》就已受到读者的广泛支持。后来又在此前的版本上逐年更新，以更加适用于广大学子备考。

本版（第六版）复习指南针对大纲，按基础知识、基本理论、基本方法及分析问题、解决问题能力的要求编写，在第五版的基础上严格按照2014年计算机统考大纲进行修订的同时，还用最新名校真题替换了大约三分之一的习题，因此更能体现最新的命题趋势。

修订后本版复习指南依然秉承如下特点。

一、对重点难点和命题方向的独特把握

编者主要以两个标准判定重难点，预测命题方向：

第一，统计各大名校计算机考研历年真题命制点，提炼出较为笼统的主要命题知识点，总结出笼统的命题规律，以此预测2013年考试真题；

第二，分析2013年考题及2014年考纲，结合编者多年命制考研试题的经验，以此判定2014大纲包含知识点中的重点难点。

以这两个标准总结出命题方向和重点难点，具有很大的兼容性和稳定性。无论2014年试题如何命制，都不会超出本书对命题方向、重点难点判定的范围。

Contents 目录

Contents

contents

第 1 篇　数据结构

第1篇

数据结构

第1章 线性表

知识点精讲

一、线性表的定义(*历年未考考点*)

线性结构：在数据元素的非空有限集中，(1) 存在唯一的一个被称作“第一个”的数据元素；(2) 存在唯一的一个被称作“最后一个”的数据元素；(3) 除第一个之外，集合中的每个数据元素均只有一个前驱；(4) 除最后一个之外，集合中的每个数据元素均只有一个后继。

线性表：一个线性表是 n 个数据元素(或结点)的有序序列：

$$a_0, a_1, a_2, \cdots, a_{n-1}$$

其中 a_0 是**开始结点**，a_{n-1} 是**终端结点**，a_i 是 a_{i+1} 的**前驱结点**，a_0 无前驱结点，a_{i+1} 是 a_i 的**后继结点**，a_{n-1} 无后继结点。

线性表的常用基本操作：

ElemType GetElem(Sqlist L, int i, ElemType &e)：用 e 返回 L 中第 i 个数据元素的值。

status ListInsert(Sqlist &L, int i, ElemType e)：在 L 中第 i 个位置之前插入新的数据元素 e，L 长度加 1。

status ListDelete(Sqlist &L, int i, ElemType &e)：删除 L 的第 i 个数据元素，并用 e 返回 s 值，L 长度减 1。

线性表的基本操作还有几个，请大家自己查阅数据结构推荐参考书第 19 页。

【注】本章节中的函数的描述是用类 C 语言的，其中 status 定义如下：

```
typedef enum status{
    OK,
    ERROR,
}status;
```

ElemType 则是元素类型的通称，例如用到整型则可用：typedef int ElemType。

二、线性表的顺序表示和实现(*2010 年考过*)

1. 线性表的顺序表示

线性表的顺序表示指的是用一组地址连续的存储单元依次存储线性表的数据元素。这样，线性表中第 0 个元素的存储位置就是指定的存储位置，第 i 个元素($1 \leqslant i \leqslant n-1$)的存储位置紧接在第 i−1 个元素的存储位置的后面。顺序存储的线性表可简称为顺序表(见图 1.1)。

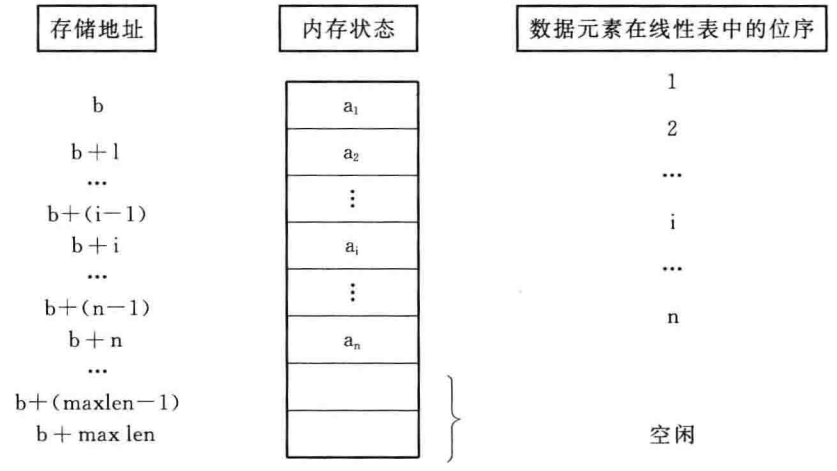


图 1.1 线性表的顺序存储示意图

假定线性表的元素类型为 ElemType，那么每个元素占用的存储空间大小即为 sizeof(ElemType)，图中令 l=sizeof(ElemType)。因此我们可以得出，在顺序存储方式下，只要我们知道线性表的首址以及数据元素的位序以及大小，我们就能够得到这个数据元素的存储地址，因此线性表的顺序存储是可以实现随机存取的。

maxlen 定义为一个整型常量，为线性表中的最大元素数。如果线性表最多只有 100 个元素，可定义如下：

```
#define maxlen 100
```

下面定义顺序表的存储结构：

```
typedef struct{
    ElemType element[maxlen];                    //存放数据元素
    int len;                                     //存放线性表的长度
}Sqlist;
```

2. 顺序表基本操作的实现

(1) ElemType GetElem(Sqlist L,int i)：返回 L 中第 i 个数据元素的值。

```
ElemType GetElem(Sqlist L,int  i)
{
    if(i<0 || i>L.len)
        Error("顺序表下标访问越界");
    else
        return(L.element[i]);
}
```

【分析】 由以上算法我们可以看出，在给出线性表某一结点的位序后，我们可以很容易的得到这个结点。因此实现了对线性表的随机存取。

(2) status ListInsert(Sqlist &L,int i,ElemType e)：在 L 中第 i 个位置之前插入新的数据元素 e，L 长度加 1。

```
status ListInsert(Sqlist  &L,int  i,ElemType e){
    int j;
```

```
        if(i<0 || i>L.len-1)                          //顺序表下标访问越界
            return ERROR;
        else if(L.len==maxlen)
            return ERROR;
        else{
            for(j=L.len-1;j>=i;j--){
                L.data[j+1] = L.data[j];              //结点后移,为插入腾出位置
            }
            L.data[i] = e;                            //插入e
            L.len++;
            return OK;                                //成功插入
        }
    }
```

【分析】 由以上算法我们可以看出对顺序表进行插入时要移动i后的所有元素。因此插入效率不高。顺序表插入操作的平均时间复杂度为O(n)。

(3) status ListDelete(Sqlist &L,int i,ElemType &e):删除L的第i个数据元素,并用e返回其值,L长度减1。

```
status ListDelete(Sqlist &L,int i,ElemType &e){
    int j;
    if(i<0 || i>L.len-1)                              //顺序表下标访问越界
        return ERROR;
    else{
        e = L.data[i];
        for(j=i;j<L.len-1;j++)
            L.data[j] = L.data[j+1];
        L.len--;
        return OK;                                    //成功删除
    }
}
```

【分析】 由以上算法我们可以看出对顺序表进行删除时要将删除位置后的所有元素前移,因此删除的效率也不高。顺序表删除操作的平均时间复杂度为O(n)。

三、线性表的链式存储及其实现(*2009、2012年考过*)

链式存储就是使用链表实现的存储结构,它不需要一组连续的存储单元,而是可以使用一组任意的,甚至是在存储空间中零散分布的存储单元存放线性表的数据,从而解决了顺序存储线性表需要大块连续存储单元的缺点。重点掌握链表的插入、删除操作。

1. 线性链表

为了表示每个数据元素与其直接后继数据元素之间的逻辑关系,我们在每个节点中除包含有数据域外,还设置了一个指针域,用来指向其后续结点。这样构成的链表称为**线性链表**或者**单链表**。

单链表分为带头结点和不带头结点两种。单链表带头结点好处有二：第一，有头结点后，插入和删除数据元素的算法统一了，不再需要判断是否在第一个元素之前插入和删除第一个元素；第二，不论链表是否为空，链表指针不变。因此，为了简便，以下讨论的都是带头结点的单链表。如果为空表，那么头结点的指针域为空(NULL)。

头指针是指指示链表中第一个结点存储位置的指针。**头结点**是指在单链表第一个结点前所附设的一个结点，这个结点的指针域存储指向第一个结点(包括头结点)的指针。请大家注意区别这个概念。在图 1.2 中，L 为头指针，带阴影的结点为头结点。

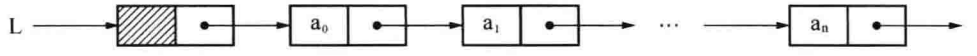


图 1.2 带头结点的单链表

由以上的描述可以知道，如果我们要知道链表中某一个结点的信息，就必须知道这个结点的直接前驱结点的信息，因为此结点的存储位置保存在其直接前驱的地址域里。所以，链表是无法实现随机存取的。

下面定义线性链表的存储结构：

```
typedef struct LNode{
    ElemType data;                          //数据域
    struct LNode * next;                    //指针域
} LNode, * LinkList;
```

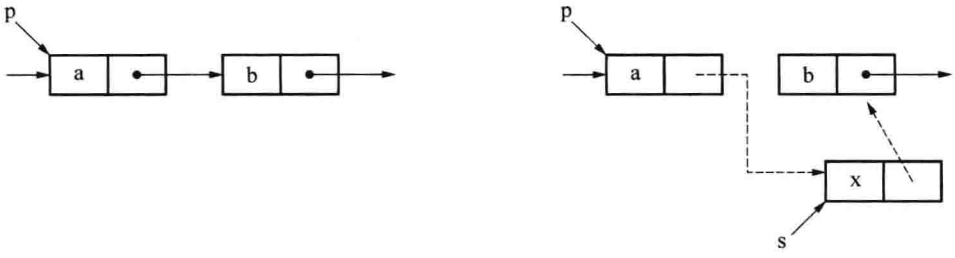
2. 线性链表基本操作的实现

(1) ElemType GetElem(Linklist L,int i)：用 e 返回 L 中第 i 个数据元素的值。

```
ElemType GetElem(LinkList L,int i){
    //L 为带头结点的单链表的头指针
    p = L->next;
    j=0;
    while(p&&j<i){//顺指针往下找，直到 p 指向第 i 个元素或者 p 为空(即下标过界)
        p = p->next;
        j++;
    }
    if(! p||j>i)
        error(0);                           //第 i 个元素不存在，error 退出
    e = p->data;
    return e;
}
```

【分析】 链表中按序号查找元素操作的平均时间复杂度为 O(n)。由以上算法我们可以看出，在给出线性表某一结点的位序后，我们必须从单链表的头结点开始，将位置标志指针 p 一直往下移动，如果我们需要的是线性表的最后一个元素，那么这个指针就将从链表头移动到链表尾，因此用单链表存储的线性表在对结点进行随机查询的效率是非常低的。

(2) status ListInsert(LinkList &L,int i,ElemType e)：在 L 中第 i 个位置之前插入新的数据元素 e。



(a) 插入前　　(b) 插入后

图 1.3　单链表插入结点的指针变化情况

假设我们要在线性表的两个元素 a 和 b 之间插入一个元素 x，指针的指向情况如图 1.3 所示。首先我们要生成一个数据域是 x 的结点，然后使得 x 的指针域指向结点 b，最后修改结点 a 的指针域，使其指向结点 x。注意：这个步骤是不能调换顺序的！简单描述如下：

```
s->next = p->next;
p->next = s;
```

完整算法描述如下：

```
status ListInsert(LinkList &L,int i,ElemType e){
    //在带头结点的单链表 L 中第 i 个位置之前插入元素 e
    p = L;
    j = 0;
    while(p&&j<i-1){              //顺着指针一直向后寻找插入位置
        p = p->next;
        j++;
    }
    if(! p||j>i-1)
        return ERROR;
    s = (LinkList)malloc(sizeof(LNode));  //生成新结点
    s->data = e;                          //插入结点
    s->next = p->next;
    p->next = s;
    return OK;
}
```

【分析】 由以上算法可以知道，当我们需要往线性表的指定位置插入某一个结点的时候，我们不再需要像顺序表一样移动大量的结点以完成结点的插入，而只需要简单的修改几个指针即可，插入效率高。并且还能很方便地实现线性表长度的动态变化，更加灵活的使用计算机内存资源。但其平均时间复杂度仍为 O(n)，算法主要的时间开销在于查找第 i−1 个元素。

(3) status ListDelete(LinkList &L,int i,ElemType &e)：删除 L 的第 i 个数据元素，并用 e 返回其值。

为了删除单链表中的结点 b，仅需修改结点 a 的指针域，使其指向 b 的直接后继结点 c，如图 1.4 所示。其修改指针的语句如下：

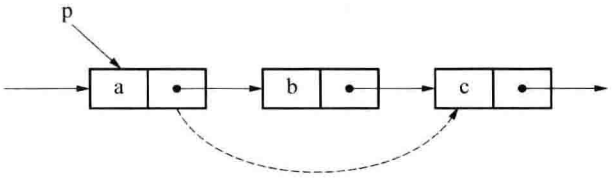


图 1.4 单链表的删除

```
p->next = p->next->next;
```

完整算法描述如下：

```
status ListDelete(LinkList &L,int i,ElemType &e){
//在带有头结点的单链表中删除第 i 个元素
p = L;
j = 0;
while(p&&j<i-1){                    //顺着指针一直向后寻找插入位置
      p = p->next;
      j++;
}
if(! (p->next)||j>i-1)              //删除位置不正确
         return ERROR;
q = p->next;                        //因为考虑到要回收内存,因此用 q 指针保存需删除结点的地址
p->next = q->next;
e = q->data;
free(q);                  //回收内存。对于内存小的系统,这样做是非常有必要的
return OK;
}
```

【分析】 由以上算法我们可以知道，在已知链表中要删除的结点的确切位置的情况下，在单链表中删除一个结点时，仅需修改相关的指针，而不需要移动元素，删除的效率高。该算法的平均时间复杂度与插入操作相同，为 O(n)。

3. 静态链表

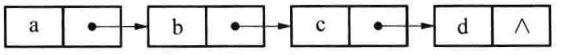
在有的情况下，也可借用一维数组来描述线性链表，其存储结构如下：

```
#define MAX 100
typedef struct{
      ElemType data;
      int cur;
}SLinkList[MAX];
```

这种描述方法使得我们在没有设置指针类型的某些高级程序设计语言中可以使用链表结构，如 Java 中。cur 分量存储的不再是链表中的指针域，而是存储其下一结点在一维数组中的位序。为了和指针描述的线性链表相区别，我们就将这种用数组描述的链表结构叫做**静态链表**(见图 1.5)。

0		2
1	b	6
2	a	1
3	d	0
4		
5		
6	c	3

(a) 静态链表示例



(b) 静态链表对应的单链表

图 1.5 静态链表存储示意图

静态链表以 cur==0 作为其结束的标志。总的来说，静态链表是没有单链表使用起来方便的，但是在不支持指针的高级语言中，这又是一种非常巧妙的设计方法。

4. 循环链表

循环链表是另外一种形式的链式存储结构，其中比较常用的是**循环单链表**和**循环双链表**。

(1) 循环单链表

循环单链表的特点是表中最后一个结点的指针不再为空，而是指向该链表的表头结点，将整个链表链接成一个环。这样，由此表中的任一结点出发均可以访问到链表中的其他结点。

循环单链表的基本操作的实现方法与单链表基本相同，只是在对表尾判断的条件上有所改变。如图 1.6 所示，在一个头指针为 h(此头指针指向头结点)的循环单链表中，判断表空的条件不再是 h->next == null，而是 h->next == h；判断表尾结点的条件是 p->next == h。

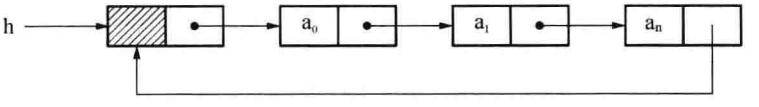


图 1.6 循环单链表

(2) 循环双链表

以上讨论的链式存储结构的结点中只有一个指示直接后继结点的指针域，这就造成了我们只有顺着指针往后访问其他结点。如果要访问某个节点的前驱，则必须从表头指针开始查找。换句话说，就是在以上所有的链式存储结构中，找后继容易，找前驱麻烦。因此，为了克服这个缺点，我们引入了**循环双向链表**(见图 1.7)。

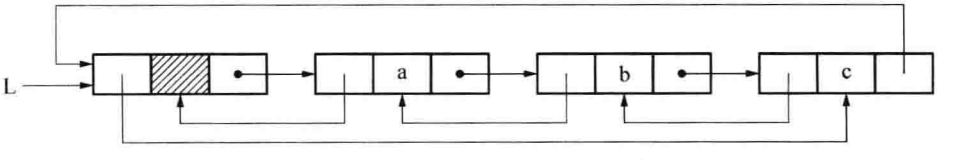


图 1.7 循环双向链表存储示意图

循环链表的存储结构：

```
typedef struct DulNode{
    ElemType data;
    struct DulNode *prior;
```

```
    struct DulNode * next;
}DulNode, * DulLinkList;
```

在双向链表中，若 p 为指向其中某一结点的指针，则显然有

$$p->next->prior = p->prior->next = p$$

5. 循环双链表基本操作的实现

(1) ElemType GetElem(DulLinkList L,int i)：用 e 返回 L 中第 i 个数据元素的值。

```
ElemType GetElem(DulLinkList L,int i){
    j = 0;
    q = L->next;
    while(j<i && q! =L){ //查找第 i 个结点
        q = q->next;
        j++;
    }
    if(q ! = L){              //返回第 i 个元素值
        e = q->data;
        return e;
    }
    else{
        error("位置参数 i 不正确");
        return NULL;
    }
}
```

【分析】 双向链表的 GetElem 操作与单链表没有很大区别，但是在访问结点直接前驱方面是非常方便的，只要简单的顺着 prior 指针往前寻找即可。这一点是单链表无法做到的。因此在我们需要方便访问结点直接前驱的情况下，我们可以考虑使用双向链表这种数据结构。

(2) status ListInsert(DulLinkList &L,int i,ElemType e)：在 L 中第 i 个位置之前插入新的数据元素 e。

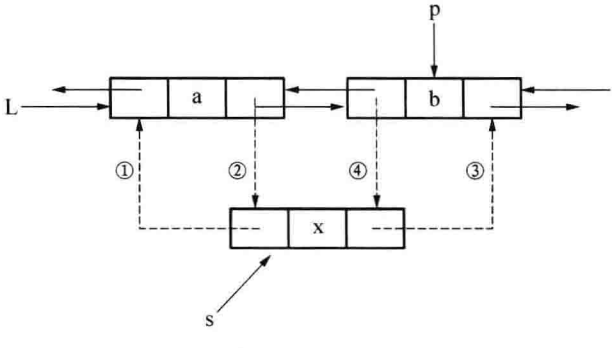


图 1.8 双向链表插入一个结点后的指针变化情况

```
status ListInsert(DulLinkList & L,int i,ElemType e){
  if(! (p = GetElemP_Dul(L,i)))    //在 L 中确定第 i 个元素的位置指针 p
```

```
        return ERROR;    //p=null,即第 i 个元素不存在,位置参数 i 错误
    if(! ( s = (DulLinkList)malloc(sizeof(DulNode))))
        return ERROR;          //内存分配不成功
    s->data = e;
    s->prior = p->prior;       //①
    p->prior->next = s;        //②
    s->next = p;               //③
    p->prior = s;              //④
    return OK;
}
```

【分析】 插入结点后指针的改变步骤见算法和图 1.8(重点掌握此图)的对应标号。在双向链表中插入结点的时候,指针的移动顺序可以改变吗(请考生自己思考一下)? 并且要注意插入点在表头和表尾时结点指针的指向问题。这个请考生自己总结归纳。

(3) status ListDelete(DulLinkList &L,int i,ElemType &e):删除 L 的第 i 个数据元素,并用 e 返回其值。

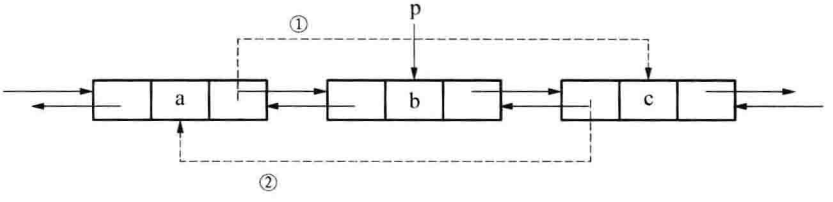


图 1.9 在双向链表中删除结点时指针的变化情况

```
status ListDelete(DulLinkList &L,int i,Elemtype &e){
  if(! (p = GetElem_Dul(L,i)))
        return ERROR;
  e = p->data;
  p->prior->next = p->next;          //①
  p->next->prior = p->prior;         //②
  free(p);                           //内存回收
  return OK;
}
```

【分析】 删除结点后指针的改变步骤见算法和图 1.9 的对应标号。在双向链表中删除结点的时候,指针的移动顺序可以改变吗(请考生自己思考一下)?

6. 线性表在现实生活中的应用问题

线性表在现实生活中的应用主要就是对一元多项式的表示以及相加,这个只需作为一个知识背景进行了解即可。

例题精析

【例1】 带头结点的单链表(头指针为h)为空的判定条件是(　　)。

A. h == NULL　　B. h->next == NULL

C. h->next == h　　D. h ! = NULL

解 在带头结点的单链表中,头指针h指向头结点,而头结点的指针域next指向单链表的第一个元素节点,因此h->next == NULL表示单链表为空。

因此本题答案为B。

【例2】 若线性表最常用的操作是存取第i个元素及其前驱和后继元素的值,为节省时间应采用(　　)的存储方式。【北京理工2004】

A. 单链表　　B. 双向链表　　C. 单循环链表　　D. 顺序表

解 本题实际上就是要求所选的存储结构能够迅速的查找第i个结点和第i-1个结点。A、B和C的结构,都必须顺着指针方向依次查找,时间复杂度均为O(n),而顺序表实现此操作的时间复杂度只有O(1)。

因此本题的答案为D。

【例3】 请简要叙述顺序存储和链表存储两种存储结构的特点。

解 **顺序存储:** 存储密度大,可随机存储元素;但是顺序表大小固定,不利于增减结点;在进行插入和删除操作时,需要移动大量元素;由于难以估计,必须预先分配较大的存储空间,往往使得存储空间不能得到充分利用;表的容量难以扩充。

链式存储: 指针域的使用增加了存储空间的开销且链式存储不可随机存取元素;但是,在进行插入和删除时,不需移动元素,只需简单地修改指针;不需要预先分配空间,可根据需要动态申请空间;表容量只受可用内存空间的限制。

【注】本题是对顺序存储和链表存储结构特点的总结,考生可在理解的基础上熟记这些特点,以应对考试中可能会涉及的题目(如本题前的两个题目),达到举一反三的目的。

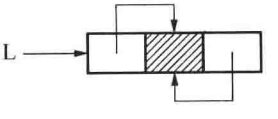
【例4】 带头结点的循环双向链表(头指针为L)为空的判定条件是(　　)。

A. L == NULL

B. L->next->prior == NULL

C. L->prior == NULL

D. L->next == L



解 如右上图所示,则易知该循环双向链表为空的条件为L->next == L或者L->prior == L。

因此本题答案为D。

【例5】 设线性表有n个元素,以下操作中,(　　)在顺序表上实现比在链表上实现效率更高。【武汉大学2006】

A. 输出第 i(1≤i≤n)个元素值
B. 交换第 1 个元素与第 2 个元素的值
C. 顺序输出这 n 个元素的值
D. 输出与给定值 x 相等的元素在线性表中的序号

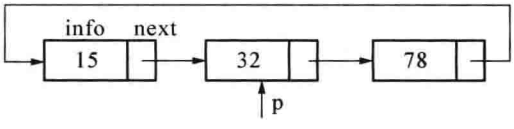
解 B、C、D 三种情况在顺序表和链表中的实现效率是一样的。A 项操作在顺序表中的时间复杂度是 O(1),在链表中的时间复杂度是 O(n)。因此本题答案为 A。

【例 6】 以下关于链式存储结构的叙述中,(　　)是不正确的。 【中科院 2004】

A. 结点除自身信息外还包括指针域,因此存储密度小于顺序存储结构
B. 逻辑上相邻的结点物理上不必邻接
C. 可以通过计算直接确定第 i 个结点的存储地址
D. 插入、删除运算操作方便,不必移动结点

解 本题考查链式存储结构。链式结构的优点是各结点逻辑上相邻,但是物理上可以不相邻。基于链式结构上的插入和删除操作比较方便,不需要移动其他数据元素,只需要修改链接。一般的结点结构分为数据域和指针域。因此本题答案为 C。

【例 7】 如果环形链表结构如下图所示,则表达式 p->next->next 的值是(　　)。 【华南理工 2007】



A. 15　　B. 32　　C. 78　　D. 全不是

解 p->next->next 的值是指针,指针是地址,它指向某个数。因此本题答案为 D。

【例 8】 线性表的顺序存储结构是一种(　　)。 【北京理工 2006】

A. 随机存取的存储结构　　B. 顺序存取的存储结构
C. 索引存取的存储结构　　D. Hash 存取的存储结构

解 线性表的顺序存储结构是一种随机存取的存储结构,即可随意访问表中任何位置的结点。线性表的链式存储结构是一种顺序存取的存储结构,即要访问表中某一位置的结点时,要从表头开始,按指针顺序一直往后查找。因此本题选择 A。

【例 9】 单链表中,增加一个头结点的目的是为了(　　)。 【江苏大学 2005】

A. 使单链表至少有一个结点　　B. 标识表结点中首结点的位置
C. 方便运算的实现　　D. 说明单链表是线性表的链式存储

解 单链表带头结点好处有二:第一,有头结点后,插入和删除数据元素的算法统一了,不再需要判断是否在第一个元素之前插入和删除第一个元素;第二,不论链表是否为空,链表指针不变。则可以看出设置头结点的目的就为了方便运算的实现。因此本题答案选择 C。