数据结构第三节课官方笔记

目录

一、课件下载及重播方法

二、教材结构图

三、本章知识点及考频总结

（一）选择题（共 8 道）

1. 线性表顺序存储结构的特点是，在逻辑关系上相邻的两个元素在物理位置上也是相邻的，因此可以随机存取表中任一元素。但是，当经常需要做插入和删除操作运算时，则需要移动大量的元素，而采用链式存储结构时就可以避免这些移动。然而，由于链式存储结构存储线性表数据元素的存储空间可能是连续的，也可能是不连续的，因而链表的结点是不可以随机存取的。

2. 单链表的插入运算

```
s= ( ListNode * ) malloc ( sizeof ( ListNode ) ) ;
s->data=x ; s->next=p->next ;
p->next=s ;
```

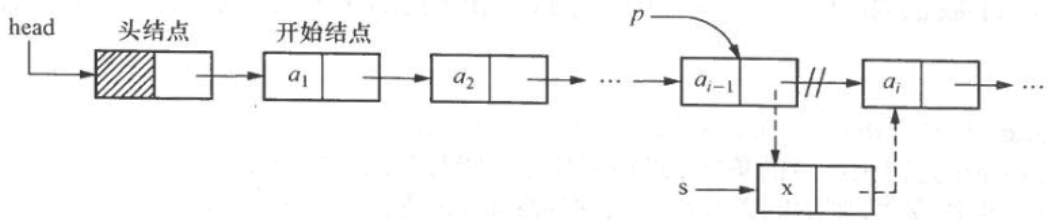

图 2.6　在 p 指向结点之后插入新结点*s 示意图

3. 单链表的删除运算：

```
s=p->next ;              //s 指向第 i 个结点
p->next=s->next ;  //使 p->next 指向第 i+1 个结点
x=s->data ;              //保存被删除结点的值
free ( s ) ;
return x ;
```

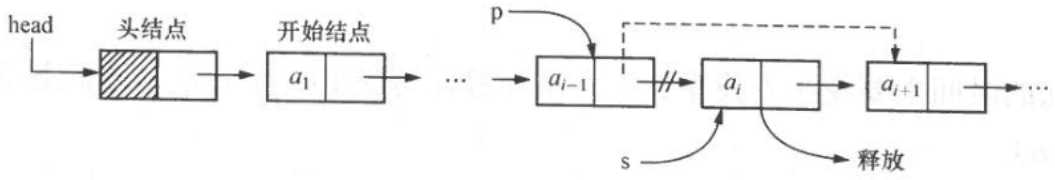

图 2.7　删除第 *i* 个结点示意图

4. 循环链表的结点类型与单链表完全相同，在操作上也与单链表基本一致，差别仅在于算法中循环的结束判断条件不再是 p 或 p->next 是否为空，而是它们是否等于头指针。

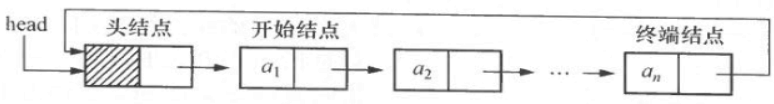

图 2.8　单循环链表示意图

5. 在双向链表的给定结点前插入一结点的操作过程如图 2.11 所示。设 p 为给定结点的指针，x 为待插入结点的值，其实现算法如下：

```
void DLInsert ( DLNode *p , DataType x )
{   //将值为 x 的新结点插入到带头结点的双向链表中指定结点*p 之前
    DLNode *s= ( DLNode * ) malloc ( sizeof ( DLNode ) ) ;   //申请新结点
    s->data=x ;
    s->prior=p->prior ; s->next=p ;
    p->prior->next=s ; p->prior=s ;
}
```

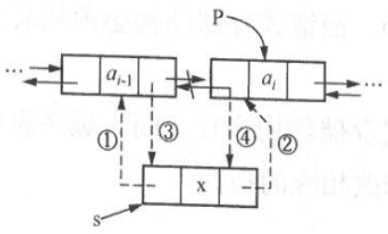

图 2.11　在双向链表上 p 指向结点之前插入新结点*s 示意图

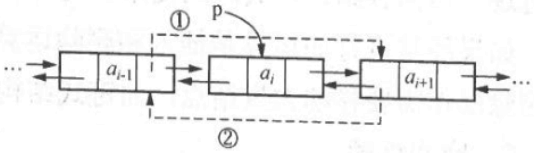

图 2.12　在双向链表上删除 p 指向的结点示意图

6. DataType DLDelete (DLNode *p)

```
{   //删除带头结点的双向链表中指定结点*p
    p->prior->next=p->next:
    p->next->prior=p->prior ;
    x=p->data ;
    free ( p ) ;
    return x ;
 }
```

7. 顺序表和链表的比较

时间性能：如果在实际问题中，对线性表的操作是经常性的查找运算，以顺序表形式存储为宜。因为顺序存储是一种随机存取结构，可以随机访问任一结点，访问每个结点的时间代价是一样的，即每个结点的存取时间复杂度均为 O(1)。而链式存储结构必须从表头开始沿链逐一访问各结点，其时间复杂度为 O(n)。

如果经常进行的运算是插入和删除运算，以链式存储结构为宜。因为顺序表作插入和删除操作需要移动大量结点，而链式结构只需要修改相应的指针。

空间性能：顺序表的存储空间是静态分配的，在应用程序执行之前必须给定空间大小。若线性表的长度变化较大，则其存储空间很难预先确定，设置过大将产生空间浪费，设定过小会使空间溢出，因此对数据量大小能事先知道的应用问题，适合使用顺序存储结构。而链式存储是动态分配存储空间，只要内存有空闲空间，就不会产生溢出，因此对数据量变化较大的动态问题，以链式存储结构为好。

对于线性表结点的存储密度问题，也是选择存储结构的一个重要依据。所谓存储密度就是结点空间的利用率。它的计算公式为

存储密度=（结点数据域所占空间）/（整个结点所占空间）

一般来说，结点存储密度越大，存储空间的利用率就越高。显然，顺序表结点的存储密度是 1，而链表结点的存储密度肯定小于 1。例如，若单链表结点数据域为整型数，指针所占的存储空间和整型数相同，则其结点的存储密度为 50%。因此，若不考虑顺序表的空闲区，则顺序表的存储空间利用率为 100%，远高于单链表的结点存储密度。

（二）主观题（共 1 道）

假设头指针为 La 和 Lb 的单链表（带头结点）分别为线性表 A 和 B 的存储结构，两个链表都是按结点数据值递增有序的。试写一个算法，将这两个单链

表合并为一个有序链表 Lc。

```
LinkList MergeList ( LinkList La , LinkList Lb )
{    //归并两个有序链表 La 和 Lb 为有序链表 Lc
ListNode *pa, *pb, *pc ; LinkList Lc ;
pa=La->next ; pb=Lb->next ;      //pa 和 pb 分别指向两个链表的开始结点
Lc=pc=La;                        //用 La 的头结点作为的 Lc 头结点
while ( pa!=NULL && pb!=NULL ) {
       if ( pa->data<=pb->data ) {
            pc->next=pa ;pc=pa ;pa=pa->next ;
       }
      else {
            pc->next=pb ;pc=pb ;pb=pb->next ;}
}
pc->next=pa!=NULL ? pa : pb ;          //插入链表剩余部分
free ( Lb ) ;                    //释放 Lb 的头结点
return Lc ;                      //返回合并后的表
}
```

四、配套练习题

1、在一个单链表中，已知 q 所指结点是 p 所指结点的后继结点，若在 p 和 q 之间插入 s 所指结点，则正确的操作是（）

A: s->next=p->next;p->next=s;

B: s->next=q;p->next=s->next;

C: q->next=s;s->next=p;

D: p->next=s;s->next=p;

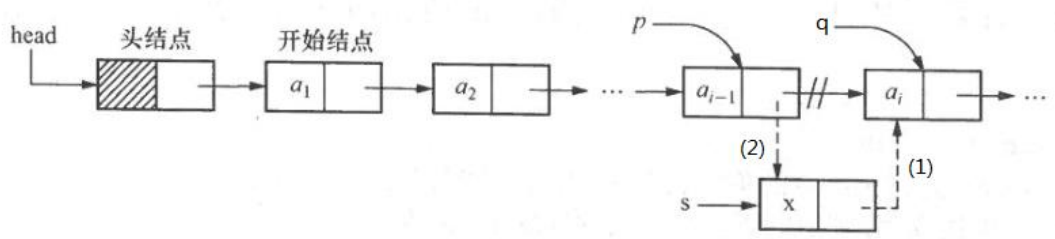

图 2.6　在 p 指向结点之后插入新结点*s 示意图

2、设指针变量 P 指向非空单链表中的结点，next 是结点的指针域，则判断 P 所指结点为尾结点前一个结点的逻辑表达式中，正确的是（）

A: p->next!=NULL&&p->next->next->next ==NULL

B: p->next!=NULL&&p->next->next ==NULL

C: p->next->next==NULL

D: p->next==NULL

3、下列选项中，属于顺序存储结构优点的是（）

A: 插入运算方便

B: 删除运算方便

C: 存储密度大

D: 方便存储各种逻辑结构

[参考答案]：ABC

五、其余课程安排