**第二章 线性表**

**一、教学基本要求**

1．掌握线性表的概念。

2．掌握线性表两种存储结构及其特点。

3．理解循环链表和双向链表的实现算法。

4. 一元多项式的表示和相加

重点：是熟练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：是能够使用本章所学到的基本知识设计有效算法解决与线性表相关的应用问题。

**二、学时安排 总学时：6，上机学时 2**

|  |  |
| --- | --- |
| 教 学 内 容 | 学 时 |
| 1．顺序及链式结构实现线性表； | 2 |
| 2．循环链表、双向链表； | 2 |
| 3．静态链表。 | 2 |

三、**教学内容**

一、线性表的类型定义

1.线性表：是数据元素的有限序列。表中的数据元素是属于同一数据对象(数据对象中的数据元素具有相同的性质)，相邻数据元素之间存在着序偶关系。所以可将线性表记为

（a1,…,ai-1,ai,ai+1,…an）

2.线性表的基本运算：表的初始化、求表长、取表中的结点、查找结点、插入结点和删除结点。以上不是全部操作，不同的问题需要的操作不同。

3.抽象数据类型线性表的定义：

ADT List{

数据对象：D={ ai | ai ∈ElemSet, i=1,2,…,n,n>=0}

数据关系：Rl={〈ai-1， ai 〉| ai-1， ai ∈D，i=1,2,…,n }

基本操作：

InitList(&L)

DestroyList(&L)

ClearList(&L)

ListEmpty(L)

ListLength(L)

GetElem (L,i,&e)

LocateElem(L,e,compare())

PriorElem(L,cur\_e,&pre\_e)

NextElem(L,cur\_e,&next\_e)

ListInsert(&L,i,e)

ListDelete(&L,i,&e)

ListTraverse(L,visit())

}ADT List

例1：扩大线性表LA，将存在于线性表LB中而不在LA中的数据元素加入到线性表LA中。

算法思想：逐一取出LB中的元素，判断是否在LA中，若不在，则插之。

【算法2.1】

void unin(ist &La List Lb)

La.len=(ListLength(La)); Lb.len=(ListLength(Lb));

for (i=1;i<=Lb\_len;i++){

GetElem(Lb,i,e);//执行时间和表长无关

if (!LocateElem(La,e,equal)) ListInsert(La,++La\_len,e);

//locatelem()执行时间和表长成正比

//La 中不存在和e 相同的元素，则插入之。

}

} union

算法的时间复杂度：O（ListLength(LA) ×ListLength(LB)）

例2： 线性表LA和LB是非递减的，将两表合并成新的线性表LC，且LC也是非递减的。

算法思想：将LA、LB两表中的元素逐一按序加入到一个新表LC中。

【算法2.2】

void MergeList(List La, List Lb, List &Lc){

InitList(Lc);

i=j=k=1; k=0;

La\_len=(ListLength(La)); Lb\_len=(ListLength(Lb));

while (i<=La\_len)&&(j<=Lb\_len) {

GetElem(La,i,ai); GetElem(Lb,j,bj);

if(ai<=bj){ListInsert(Lc,++k,ai);++i;}

else {ListInsert(Lc,++k,bj);++j;}

}

while (i<=La\_len){

GetElem(La,i++,ai); ListInsert(Lc,++k,ai);

}

while (j<=Lb\_len){

GetElem(Lb,j++,bj); ListInsert(Lc,++k,bj);

}

}//MergeList

算法的时间复杂度：O（ListLength(LA) +ListLength(LB)）

二、线性表的顺序表示和实现

1.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。用物理位置来表示逻辑结构。

LOC(ai+1)=LOC(ai)+*l*

LOC(ai)=LOC(a1)+(i-1)\**l*

其中l为每个元素需要占用的存储单元。

2、顺序表的特点：随机存取的存储结构。只要确定了存储线性表的起始位置，线性表中的任一数据元素可随机存取。

3.线性表的动态分配顺序存储结构（用一维数组）

#define LIST\_INIT\_SIZE 100

#define LISTINCREAMENT 10

typedef struct{

ElemType \*elem; //存储空间基址

int length; //线性表当前长度

int listsize; //当前已分配的存储容量

}SqList

4.顺序线性表的操作

顺序表优点是容易实现访问操作，可随机存取元素。缺点是插入和删除操作需要移动很多数据元素。

注意：在C语言中，数组的下标从“0”开始，也就说表中的第i个元素存放在L.elem[i-1]

⑴初始化操作

算法思想：构造一个空表。设置表的起始位置、表长及可用空间。

【算法2.3】

Status InitList\_Sq(SqList &L){

L.elem=(ElemType )malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

If (!L.elem)exit(OVERFLOW);

L.length=0;

L.listsize= LIST\_INIT\_SIZE;

Return OK;

}//InitList\_Sq

⑵插入操作

算法思想：在第i个元素之前插入一个新元素，将第n 至i个元素逐一向后移动一个位置。

【算法2.4】

Status ListInsert\_Sq(SqList &L, int i, ElemType e )

if (i<1||i>L.length+1) return ERROR;

if (L.length>=L.listsize){

newbase=(ElemType )realloc(L.elem,(L.listsize+LISTINCREMENT)sizeof(ElemType));

if(!newbase)exit(OVERFLOW);

L.elem=newbase;

L.listsize+= LISTINCREMENT;

}

q=&(L.elem[i-1]);

for (p=&(L.elem[L.length-1]);p>=q;--p)\*(P+1)=\*p;

\*q=e;

++L.length;

return OK;

}//ListInsert\_Sq

插入操作的时间复杂度

在插入操作中移动元素的操作为基本操作。

因为在各个元素之前插入一个元素，所需要移动的元素个数不同，例如如果要在第一个元素之间插入一个新的元素，则整个线性表中的元素都必须后移，而如果要在最后一个元素之后插入一个元素，则不需要移动线性中的任何元素。现在我们求平均时间复杂度。

设线性表中有n个元素，则新的元素可以插入的位置有n+1个，假设插入到各个位置的概率是相等的，则插入到第i个元素之前的概率为,需要移动的元素个数为(n-i+1)

所以在长度为n的线性表中插入一个元素所需要移动元素次数的数学期望(平均值是)

E=

所以插入操作的时间复杂为O(n)。

⑶删除操作：

算法思想：删除第i个元素，将第（i+1）至第n个元素逐一向前移动一个位置。(看书上24页)

⑷在线性表L中查找第1个与e满足compare()的元素的位序

见书上25页

(5)顺序表的合并

已知顺序表La和顺序表Lb,它们的元素按值非递减排序，归并La和Lb,得到一个新的线性表Lc,使Lc也按值非递减排列。(自己看程序)

线性表的链式表示和实现

一、线性表的链式存储结构

1.线性表的链式存储结构的特点：是用一组任意的存储单元存储线性表的数据元素。线性表中逻辑上相邻的两个数据元素，在物理上不一定相邻。数据元素的存储映象用一个结点来表示。结点的其中一个域表示元素本身，另一个是向其后继的指针。

2.链式存储结构的特点：插入、删除操作是不再需要移动大量的元素，但失去了顺序表的可随机存取特点。比如要访问第i个结点，必须从头开始访问。

二、线性链表：

1.单链表：链表中的每一个结点中只包含一个指针域，称为单链表或线性链表。

2.单链表的存储结构

单链表可由头指针唯一确定，在C语言中，用结构指针来描述。

typedef struct LNode{

ElemType data;

struct LNode \*next

}LNode, \*LinkList

3. 单链表的操作：

⑴访问：

算法思想：单链表是非随机存取结构。每个元素的位置信息都包含在前驱结点的信息中，所以取得第i个元素必须从头指针出发寻找。设置一个指针变量指向第一个结点，然后，让该指针变量逐一向后指向，直到第i个元素。

【算法2.5】

Status GetElem\_L(LinkList L, int i, ElemType){

//L为带头结点的单链表的头指针。

//当第i个元素存在时，其值赋给e并返回OK，否则返回ERROR

p=L→next; j=1;

while(p && j<i){

p=p→next; ++j;

}

if(!p||j>i) return ERROR;

e=p→data;

return OK;

}//GetElem\_L

⑵插入操作：要在数据元素a和b 之间插入元素x。

算法思想：决定a和b之间的相邻关系是由a 的指针决定的。若要实现插入，首先生成x结点，然后让x 的指针指向b，a 的指针指向x 。实现三个元a、x和b的逻辑关系。

设p为指向结点a 的指针，s为指向结点x的指针，则修改s、a的指针：

s→next=p→next；p→next=s；

【算法2.6】

Status ListInsert\_L(ListLInk &L, int i, ElemType e){

//在带头结点的单链表L中第i个位置前插入元素

p=L; j=0;

while(p && j<i-1){p=p→next; ++j;}

if (!p || j<i-1) return ERROR;

s=(LinkList)malloc(sizeof(LNode));

s→data=e; s→next=p-next;

p→next=s;

return OK;

}// LinkList\_L

}

⑶删除操作：在单链表数据元素a、b、c三个相邻的元素中删除b，算法思想：就是要让a 的指针直接指向c，使b从链表中脱离。

即，p→next=p→next→next(自己看30页程序)(然后讲，)。

(4)逆位序输入n个元素的值，建立带头结点的单链线性表。

这个程序相当于每次首先动态分配一个结点，然后将所给的值赋给这个结点的数据域，然后再将结点插入到线性链表中。

⑷单链表的合并：

例：将两个有序链表合并为一个有序链表。

设立三个指针pa、pb和pc 分别用来指向两个有序链表和合并表的当前元素。比较两个表的当前元素的大小，将小的元素链接到合并表中，即，让合并表的当前指针指向该元素，然后，修改指针。在归并两个链表为一个链表时，不需要另建新表的结点空间，而只需将原来两个链表中结点之间的关系解除，重新建立关系。

作业1. 已知一个单链表，数据类型为datatype,其中有指针p指向该单链表的头结点，要求在该链表中值为x的结点之后插入m个结点的算法。(P25)

静态链表、循环链表及双向链表

一、 静态单链表

有些高级语言没有指针，我们可以用数组来表示单链表，在数组中以整型游标来代替指针。这种用数组描述的链表称为静态链表。

存储结构：

#define MAXSIZE 1000

typedef struct{

ElemType data;

int cur;

}component, S [MAXSIZE];

其中S[0].cur指示第一个结点的位置，

设i=S[0].cur，则S[i].cur指示第二个节点的位置。

如果数组中第i个节点(按顺序排列)表示链表的第k个结点，则S[i].cur表示链表的第k+1个节点。

静态链表的操作和动态链表相似。以整型游标代替动态指针。

【算法2.7】

int LocateElem\_SL(SLinkList L, ElemType){

//在静态链表L中查找值为e的元素。

//若找到，则返回它在L中的位序，否则返回0。

i=S[0].cur;

while(i &&S[i].data!=e) i=S[i].cur;

return i;

}//LocateElem\_SL

下面算法2.14，2.15，2.16给出了三个过程，(1)将整个数组空间初始化为一个链表，对于给定的初始数组，初始化每个结点的后半部分，即游标部分，(2)从备用空间中取得一个空间节点，相当于C语言中的malloc函数。 开始时，数组中的节点一个接一个连成了一个空闲链表，当第一次需要结点时，将第一个节点在数组中的位置记下，然后将头结点的指针指向第二个结点，(3)释放结点，也就是将空间结点链接到备用链表上如果所释放的结点下标为k，当释放这个结点时，要将它作为空闲链表的第一个结点，也就是说将它的游标指向空闲链表的第二个结点，同时将头结点的游标指向该节点。(自己看一下这三个算法)。

例2-3 已知集合A、B，求一个集合，这个集合由一些元素组成，这些元素具有如下特性：它不能同时存在于集合A和集合B中。也就是说如果有一些元素即在集合A中，也在集合B中，就将这些元素除掉。

算法思路：(1)首先为集合A建立一个链表。

(2)每输入一个集合B中的元素，都从集合A中找一遍，看是否存在相同的元素，如果存在，则删去，否则，插入。

二、循环链表

1.循环链表：是另一种形式的链式存储结构，其特点是表中最后一个结点的指针域指向头结点，整个链表形成一个环。这样从表中任何一个结点出发都可以找到其它结点。

循环链表可分为单链和多链的。

2.循环链表的操作：和线性链表基本一致，差别仅在于循环条件判定是否为空改为是否为头指针。

三、双向链表

1.双向链表：在双向链表的结点中有两个指针域，分别指向前驱和后继。

双向链表也可以有循环链表。

2.双向链表的存储结构：

typedef struct DuLNode {

ElemType data;

struct DuLNode \*prior;

struct DuLNode \*next;

} DuLNode, \*DuLinklist;

3.双向链表的操作：

双指针使得链表的双向查找更为方便、快捷。NextElem和PriorElem的执行时间为O（1）。缺点：存储空间加大。

仅需涉及一个方向的指针的操作和线性链表的操作相同。

插入和删除需同时修改两个方向的指针。如算法2.18和算法2.19所示。

四、带头结点的线性链表类型

1.结点类型定义：

typedef struc LNode{

ElemType data;

struct LNode \*next

}\*Link, \*Position;

2、链表类型定义

typedef struc {

Link head,tail;

int len;

}LinkList

2.操作：从实际应用出发，重新定义了线性表极其基本操作。

书上37页和38页是其中的一部分操作，后面的程序都假设这些操作已经实现，例如算法2.20中

首先，直接调用LocatePos(L，i-1，h)，也就是说用h返回线性表L中第i-1个元素的位置；

然后，调用MakeNode(s，e)，分配一个结点，该结点数据域的值为e，用指针s指向该结点。

最后调用InsFirst(h，s);将s指向的结点插入到h之前。

五、一元多项式的表示和相加

如果有两个一元多项式相加，例如有：



要求写一个程序，分别输入A的值和B的值，程序能给出A+B的值。

首先分析一下多项式的特点，如果将多项式的每一项看作一个结点，那么有两种情况，

第一种情况这个节点应该包括两部分，一部分是系数，另一部分是指数，

第二种情况，这个节点只包括一部分，即系数部分，指数部分隐含在节点的序号中。

但是对于如下的方程就会浪费很多的存储空间：，对于这样的多项式，必须建立2001个结点，而其中只有三项的结点不为空。

所以我们采取第一种存储方式，即每个结点即存储系数，又存储指数。

确定了结点的结构，可以采用线性表的两种存储方式，顺序存储方式和链式存储方式，顺序存储方式的优点是可以对每个结点随机访问，结点的插入及删除不太方便，而链式存储的优点是插入删除方便，在这里经常需要插入一个结点，所以我们采用链式存储方式。

(1)首先输入数据，建立链表A和链表B。

(2)对两个指针p、q分别指向链表A和链表B的当前结点。

(3)比较这两个结点的指数项的大小，这里有三种情况

a. p．指数>q. 指数

将q所指的结点插入到A队列。q++

b. p．指数<q. 指数

p++

c. p．指数=q. 指数 且 p．系数+q. 系数=0

将P所指的结点删除，同时释放P、q 所指的结点。

d、p．指数=q. 指数 且 p．系数+q. 系数!=0

改变P指向结点的系数，同时释放q所指向的结点。

最后，如果P已经到了线性表A的表尾，而q还没有到线性表B的表尾，这时将B链表中剩下的元素插入到A链表的表尾。

自己写一下这个程序。