**第2章 线性表**

**注：所有未特别说明的链表，均为带头结点**

1. 设线性表有n个元素，以下操作中，\_A\_\_\_\_在顺序表上实现比在链表上实现效率更高

A 输出第i个元素值（i在1-n之间）

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

1. 设线性表中有2n个元素，以下操作中，\_\_A\_\_\_在单链表上实现要比在顺序表上实现效率更高

A 删除指定的元素

B 在最后一个元素的后面插入一个新元素

C 顺序输出前k个元素

D 交换第i个元素和第2n-i-1个元素的值(i在0 - n-1间)

1. 如果最常用的操作是取第i个结点及其前驱，则采用\_D\_\_存储方式最节省时间

A 单链表

B 双链表

C 单循环链表

D 顺序表

1. 将两个各有n个元素的有序顺序表(某个表中的元素，两个表之间的元素，值均有可能相同)归并成一个有序顺序表，其最少比较次数是\_\_A\_\_\_

A n

B 2n-1

C 2n

D n-1

1. 一个长度为n(n>1)的带头结点单链表h上,另设有尾指针r(指向尾结点),执行\_\_B\_\_的操作与链表的长度有关

A 删除单链表中的第一个元素

B 删除单链表的最后一个元素

C 在单链表的第一个元素前插入一个新元素

D 在单链表的最后一个元素后插入一个新元素

1. 双向循环链表中,在p结点之前插入q结点的操作是\_D\_\_\_

A p->prior=q;

q->next=p;

p->prior->next=q;

q->prior=p->prior;

B p->prior=q;

p->prior->next=q;

q->next=p;

q->prior=p->prior;

C q->next=p;

q->prior=p->prior;

p->prior=q;

p->prior->next=q;

D q->next=p;

q->prior=p->prior;

p->prior->next=q;

p->prior=q;

1. 在一个单链表中删除p结点(假设p不是尾结点)时，应执行如下操作:

(1) q=p->next；

(2) p->data=p->next->data；

(3) p->next=\_\_\_\_q->next\_\_\_\_\_\_；

(4) free(q)；

1. 在一个单链表中的p结点之前插入一个s结点，可执行如下操作：

(1) s->next=\_\_\_p->next\_\_\_\_\_\_\_\_

(2) p->next=s;

(3) t=p->data;

(4) p->data=\_\_\_\_s->data\_\_\_\_\_\_\_

(5) s->data=\_\_\_\_t\_\_\_\_\_\_\_

1. 在一个双向循环链表中删除p结点时，应执行如下操作:

(1) \_p->next->prior\_\_\_\_\_\_\_\_\_\_\_\_\_ = p->prior；

(2) p->prior->next = \_\_p->next\_\_\_\_\_\_\_\_\_\_\_；

(3) free(p)；

1. 在单链表、双向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p从相应的链表中删除(不允许进行结点之间数据域的复制)？若可以，时间复杂度各为多少？

单链表不可以

双向链表和单循环链表可以

双向链表只需要直接取p的前驱和后继，时间复杂度为O(1);

单循环链表需要循环查找链表一次来寻找p的前驱，时间复杂度为O(n)

1. 设计一个高效算法，将顺序表的所有元素逆置，要求算法的空间复杂度为O(1)

void UpsidedownList(List &L)

{

Length=ListLength(L); //顺序表长度

for(i=0;i<Length/2;i++)

{

t=L[i]; //交换第i个和第length-i-1个元素的值

L[i]=L[Length-i-1];

L[Length-i-1]=t;

}

}

1. 设计一个高效算法，从顺序表中删除所有元素值为x的元素，要求空间复杂度为O(1)

void UpsidedownList(List &L, ElemType x ,int & Length)

{

Length=ListLength(L); //顺序表长度

for(i=0;i<Length;i++)

{

if(L[i]==x)

{

for(j=i;j < Length-1;i++)

{

L[j] = L[j+1];

}

Length--

}

}

}

1. 用顺序表表示集合，设计一个求集合交集的算法

void Inter(List &La, List Lb)

{

La\_len = Listlength(La);

Lb\_len = Listlength(Lb);

for (i = 1; i <= La; i++)

{

GetElem(La, i, e); //取La中第i个元素

if (!LocateElem(Lb, e, equal)) //Lb中不存在与e相同的元素，则删除La中对应元素

{

ListDelete(La, i);

La\_len--; //la长度减1

}

}

}//Inter

1. 从带头结点的循环单链表中删除值为x的第一个结点

void ListDelete(List &L,ElemType e)

{

List p = L;

while (p->next != L);

{

if (compare(p->next->data, e)==TRUE)

{

q = p->next->next;

p = q;

free(p->next);

break;

}

else

{

p = p->next;

}

}

if (p->next == L)

return FAIL;

else

return OK;

}//Inter

1. 假定有一个带头结点的链接表，头指针为HL，每个结点含三个域:data，next和range，其中data为值域，next和range均为指针域，现在所有结点已经由next域链接起来，试编一算法，利用range域(此域的初始值均为NULL)把所有结点按照其值从小到大的顺序链接起来（next域不变）

void Range(listlink \*L)

{

for (p = \*L; p; p = p->next)

{

for (r = \*L; r->range; r = r->rang)

{

if (p->data < r->range->data)

{

p->range = r->range->range;

r->range = p;

}

}

if (!r->range)

r->range;

}

}

1. 已知带头结点的单链表L是一个递增有序表，设计一个高效算法，删除表中data值在[min .. max]之间的所有结点，并分析算法的时间复杂度

void DelMintoMax(LinkList \*&L, ElemType min, ElemType max)

{

LinkList \*p = L->next, \*q, \*r = L;

while (p != NULL && p->data < min)

{ //p指向第一个大于等于min的结点

r = p; //r为\*p的前驱结点

p = p->next;

}

q = p;

while (q != NULL && q->data <= max) //q指向第一个大于max的结点

q = q->next;

printf("p:%d q:%d\n", p->data, q->data); //删除\*p到\*q之间的所有结点

r->next = q;

r = p->next;

while (r != q) { //释放被删除结点的空间

free(p);

p = r;

r = r->next;

}

}

算法中三个循环，前两个循环至多将链表扫描一遍，第三个循环至多将链表扫描一遍，删除结点只用了一次操作，

总执行次数与链表长度成正比，所以时间复杂度是O(n)

1. 有一个值按非递减有序排列的单链表，设计一个算法删除值域重复的结点，并分析算法的时间复杂度

void DelSame(LinkList \*&L)

{

LinkList \*p = L->next, \*q;

while (p->next != NULL)

{

if (p->data == p->next->data) //找到重复值的结点

{

q = p->next; //q指向这个重复值的结点

p->next = q->next; //删除\*q结点

free(q);

}

else p = p->next;

}

}

算法中循环将链表扫描一遍，执行次数与链表长度成正比，所以时间复杂度为O(n)

1. 用单链表表示集合，设计一个算法表示集合的交

void Inter(List &La, List Lb)

{

pa=La->next;

while(pa)

{

if (!LocateElem(Lb, pa ->data, equal)) //Lb中不存在与e相同的元素，则删除La中对应结点

{

qa = pa->next; //q指向第i个结点

pa->next = qa->next; //第i-1个结点的next域指向第i+1个

free(qa);

}

pa=pa->next;

}

}//Inter

1. 写出将带头结点的双向循环链表倒置的算法

Status linklist\_reverse(listlink &L)

{

p = L;

q = p->next;

linklist\_DUL s;

while (q)

{

s = q->next;

q->next = p;

p->prior = q;

p = q;

q = s;

}

L->next = p;

p->prior = L;

return OK;

}

1. 设有一个双向链表h,设计一个算法查找第一个元素值为x的结点，将其与后继结点进行交换

Status linklist\_SEARCH(linklist &L)

{

p = L->next;

while(p)

{

if(p->data == x)

{

q = p->next; //抓住下一个结点

q->pre = p->pre; //实现结点对调

q->next->pre = p;

p->next = q->next;

p->pre->next = q;

q->next = p;

p->pre = q;

break; //对调完第一个就break掉

}

P = p->next;

}

return OK;

}

**【作业要求：】**

1、**5月15日前**网上提交本次作业（直接在本文件中作答，转换为PDF后提交即可）

2、每题所占平时成绩的具体分值见网页

3、超过截止时间提交作业会自动扣除相应的分数，具体见网页上的说明

4、**答案用蓝色标注(选择题将正确选项直接设置为蓝色文字即可)**