第2次作业

## 第2章习题

2.11设计算法以删除链表中所有值为x的元素结点。

2.18假设递增有序的带头结点的单循环链表A、B分别表示两个集合，设计算法以求解A= A∪B，并分析算法的时间复杂度。

2.22设计算法将两个递增有序的带头结点的单链表A、B合并为一个递减有序的带头结点的单链表，并要求算法的时间复杂度为两个表长之和的数量级。

2.24设计算法将带头结点的双循环链表L就地逆置，即利用原表各结点的空间实现逆置。

2.11

typedef struct LNode

{

int data;

struct LNode\* next;

}LNode, \* LinkList;

LinkList deletex(LinkList& L, int x)

{

LNode\* r = (LNode\*)malloc(sizeof(LNode));

r = L;

while (r->next != NULL)

{

LNode\* p = r->next;

if (p->data == x)

{

r->next = p->next;

free(p);

}

else

r = r->next;

}

return r;

}

2.18

struct ListNode

{

int val;

ListNode\* next = NULL;

ListNode(int key = 0, ListNode\* list = NULL)

{

val = key;

next = list;

}

};

ListNode\* unionResult(ListNode\* A, ListNode\* B)

{

if (A == NULL)

return B;

if (B == NULL)

return A;

ListNode\* curA = new ListNode(), \* temp = curA;

while (A && B)

{

if (A->val == B->val)

{

curA->next = A;

A = A->next;

B = B->next;

}

else if (A->val < B->val)

{

curA->next = A;

A = A->next;

}

else

{

curA->next = B;

B = B->next;

}

curA = curA->next;

}

while (A)

{

curA->next = A;

curA = curA->next;

A = A->next;

}

while (B)

{

curA->next = B;

curA = curA->next;

B = B->next;

}

return temp->next;

}时间复杂度O（n）;

2.22

LinkList mergelist(LinkList& La, LinkList& Lb, LinkList& Lc)

{

LNode\* r, \* p = La->next, \* q = Lb->next;

Lc->next = NULL;

while (p && q)//当两个列表均非空时

{

if (p->data <= q->data)//若p的值较小

{

r = p->next;//r为p的后继结点，以防断链

p->next = Lc->next;

Lc->next = p;

p = r;

}

else

{

r = q->next;

q->next = Lc->next;

Lc->next = q;

q = r;

}

}

if (p) q = p;//处理剩下的一个列表非空时

while (q)

{

r = q->next;

q->next = Lc->next;

Lc->next = q;

q = r;

}

return Lc;

}

2.24

#include <iostream>

using namespace std;

typedef struct DNode

{

int data;

struct DNode\* prior, \* next;

}DNode, \* DLinklist;

void DLListReverse(DNode\* L)

{

DNode\* p, \* u; //P 指向当前待逆置的结点，u 指向未逆置的下一个结点

p = L->next; //P 指向原链表的首元素结点，此结点逆置后为尾结点，需要单独处理

if (p != L) //非空链表，处理第一个结点

{

u = p;

p = p->next; //p 指向链表下一个结点

u->next = L; //建立 next 向循环

L->prior = u; //建立 prior 向循环

} //循环逆置剩余结点

while (p != L)

{

u = p;

p = p->next; //p 指未逆置部分下一个结点

u->prior = L; //u 头插成为首元素结点，前向指针指向头结点

u->next = L->next; //u->next 指向已逆置部分原来的首元素结点

L->next->prior = u; //已逆置部分原来首元素结点的 prior 指针指向 u

L->next = u; //L->next 指向 u，u 插入成为首元素结点

}

}