2

int findE(Sqlist L, int x) {

int i;

for (i = 0; i < L.length; ++i) {

if (x < L.data[i]) {

return i;

}

}

return i;

}

void insertE(Sqlist& L, int x) {

int p, i;

p = findE(L, x);

for (i = L.length - 1; i >= p; --i)

L.data[i + 1] = L.data[i];

L.data[p] = x;

L.length++;

}

就是简单的插入问题，麻烦的无非是顺序表必须把后面的元素集体向后移一位，从最后面移要简单点。

3

（1）17

（2）O（n的平方）

#include<iostream>

using namespace std;

int main() {

int array[18] = { 1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 6, 6, 7, 7, 8, 8, 8, 9 };

int cnt = 18;

int r = 0, l = 1;//双指针

int times = 0;//记录次数

while (r != l - 1 || l != cnt - 1) {

if (array[r] == array[l]) {

for (int i = r; i < cnt - 1; i++) {

array[i] = array[i + 1];

times++;

}

if (r = cnt - 1) {

if (array[r] != array[r - 1]) {

break;

}//剪枝

}

array[cnt - 1] = NULL;

cnt--;

}

else {

l = r;

r++;

}

}

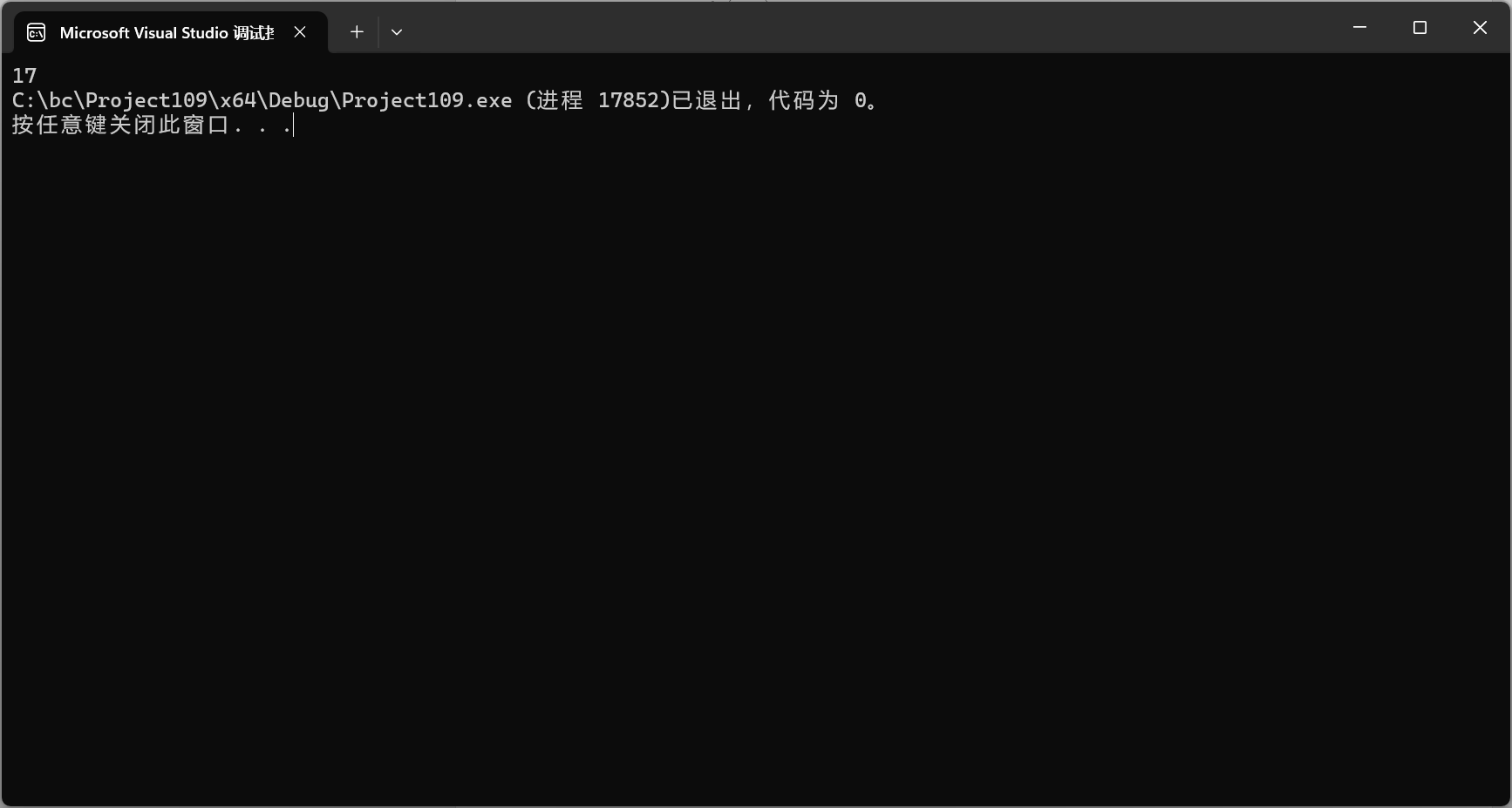
cout << times;

return 0;

}

//时间性能O（n^2）

比较后面的元素与前面的是否一致，不一致就删掉



6

public boolean gather(Node a, Node b) {

Node A = a.next; //指向链表A的第一个结点

Node B = b.next; //指向链表B的第一个结点

int n = 0; //用来计数

while (A != null) {

while (B != null) {

if (A.data < B.data) { //如果集合A的数值大于集合B中的数值

A = A.next;//A的指针指向下一个元素，B的指针不变，依旧是这个元素

break;//终止内循环，从外循环开始，A的在一个元素和B的本元素比较

}

if (A.data == B.data) {//如果集合A的数值等于集合B中的数值

A = A.next; //A的指针指向下一个元素

B = B.next; //B的指针指向下一个元素

n++; //当n的值等于A集合的元素个数，说明A是B的子集

break;

}

if (A.data > B.data) { //如果集合A的数值小于集合B的数值

B = B.next;//那么B的指针指向下一个元素，A的指针依旧是指向本元素

if (A.next == null && B.next == null) {

if (A.data == B.data)//A和B均为最后一个元素且相等，则计数

n++;

A = A.next; //如果A和B均为最后一个元素不相等， 则A指向空，让外部循环停止

}

break;

}

}

}

if (n == List\_Length(a)) //当n的值等于A集合的元素个数，说明A是B的子集

return true;

else

return false;

}

7. #include <iostream>

using namespace std;

typedef struct LNode

{

int data;

LNode\* next;

}LNode, \* LinkList;

//创建链表

int CreateList(LinkList& L, int n)

{

LNode\* p, \* r; int i;

L = new LNode;

L->next = NULL;

r = L;

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

{

p = new LNode;

cin >> p->data;

p->next = NULL; r->next = p;

r = p;

}

return 0;

}

//输出链表

void display(LinkList L)

{

LNode\* p;

p = L->next;

cout << "{";

while (p)

{

cout << p->data << " ";

p = p->next;

}

cout << "}" << endl;

}

//交集

void MergeList\_L(LinkList& LA, LinkList& LB, LinkList& LC)

{

LNode\* pa; LNode\* pb; LNode\* pc; LNode\* p;

pa = LA->next; pb = LB->next;

LC = LA;

pc = LC;

while (pa && pb)

{

if (pa->data == pb->data)

{

pc->next = pa;

pc = pa;

pa = pa->next;

pb = pb->next;

}

else if (pa->data < pb->data)

pa = pa->next;

else

pb = pb->next;

}

pc->next = NULL;

LA = LC;

delete LB;

}

int main()

{

LinkList LA; LinkList LB; LinkList LC; int n;

cout << "请输入需要创建的A集合元素的个数：" << endl;

cin >> n;

cout << "请依次输入需要存入的数据(尾插法&&递增)：" << endl;

CreateList(LA, n);

cout << "请输入需要创建的B集合元素的个数：" << endl;

cin >> n;

cout << "请依次输入需要存入的数据(尾插法&&递增)：" << endl;

CreateList(LB, n);

cout << "集合A为：";

display(LA);

cout << "集合B为：";

display(LB);

MergeList\_L(LA, LB, LC);

cout << "二者的交集为：";

display(LA);

return 0;

}



时间复杂度为O(n^2)

8

nodetype\* invert(nodetype\* head)

{

nodetype\* p = head, \* q, \* s;

if (p != NULL)

{

q = p->next;

head->next = head->left; / 逆置头结点

head->left = p->next;

while (q != head)

{

s = q->next;

q->left = q->next;

q->next = p;

p = q;

q = s;

}

}

return head;

}