**Nankai University**

南 开 大 学

计算机学院

数据结构实验二

**实验二**实验报告

冯思程2112213

年级：2021级

专业：计算机科学与技术

指导教师：王玮

目录

**一、实验题目**

[（一）作业要求](#bookmark51" \o "Current Document)

[（二）问题描述](#bookmark60" \o "Current Document)

**[二、实验方案](#bookmark79" \o "Current Document)**

[（一）实验工具](#bookmark84" \o "Current Document)

[（二） 实验结果分析](#bookmark106" \o "Current Document)

[三](#bookmark147" \o "Current Document)**[、算法设计及分析](#bookmark147" \o "Current Document)**

[i代码展示和分析](#bookmark147" \o "Current Document)

[四](#bookmark313" \o "Current Document)**[、复杂度分析](#bookmark313" \o "Current Document)**

1. **[时间复杂度分析](#bookmark313" \o "Current Document)**
2. **[空间复杂度分析](#bookmark313" \o "Current Document)**

[五](#bookmark338" \o "Current Document)**[、实验总结](#bookmark338" \o "Current Document)**

—' 实验题目

（一） 作业要求

自行输入两个无序 链表A和B，要求均带有头结点，A以单链表 形式存储， B以循环链表 形式存储。

(1)将A和B在自身基础上分别就地逆置，要求不占用额外的存储空间；

(2)删除A链表中节点值与B链表中节点值相同的结点；

(3)找到A链表经过（2）操作后的倒数第k个结点，k要求手动输入。

1.实验顺序不能颠倒；

2.每一小问完成后输出结果；

（二） 问题描述

1.

算法逻辑部分：

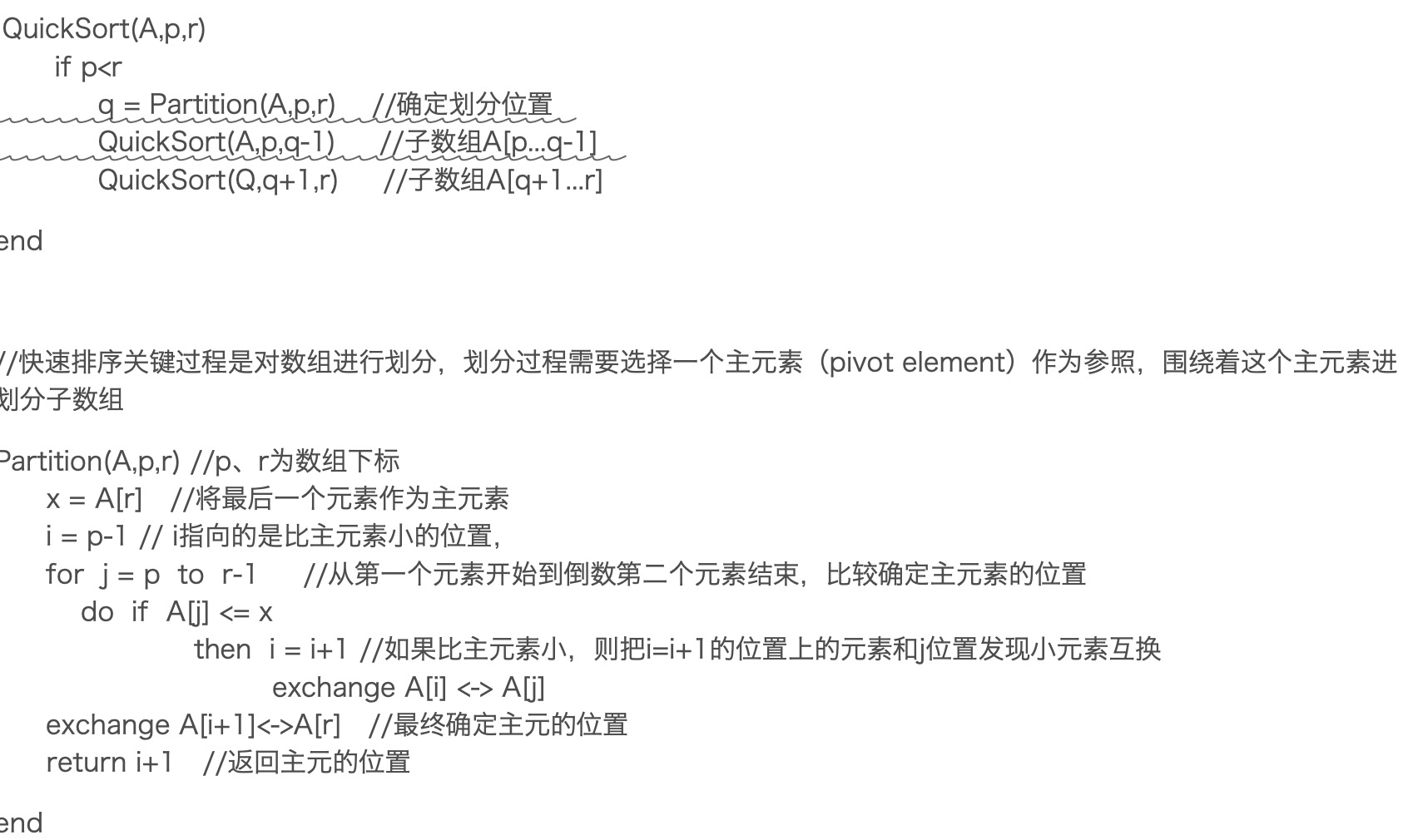
（1）首先定义结点类，在本次实验中数据都是int类型，故不采用模版的写法。然后分别定义了单链表和循环链表的类，并写出各自的构造函数。

（2）分别定义出单链表和循环链表的insert函数和reverse函数。其中insert函数是存数据的函数，reverse函数是转置函数，用的算法是三指针法。

（3）然后，定义了单链表的remove函数，即删除有与循环链表数据相同的数据的函数。分为三种情况，分别是相同元素在head的next、中间结点、尾结点。分别按具体情况编写代码。

（4）然后是主函数的部分，分别输入了单链表和循环链表的链表长度和链表数据。首先利用reverse函数实现了转置，并输出。然后进行链表有相同的元素删除，因为我的remove函数有在单链表筛选并删除的功能，所以我只需在主函数中写一个能遍历循环链表的for循环即可实现功能，然后输出经过删除操作后的链表。接下来开始实现第三个功能，首先，我们发现此时单链表的长度是我们无法确定的，所以写一个循环来对单链表进行计数，得出现在单链表的长度，然后倒数第k个数字即是正数第（链表长度+1-k）个，从头循环，直到指针指向目标的结点，然后进行结点数据的输出，实现了第三个功能。

（5）然后是细节处理部分，为了使代码功能更加清晰明了，我在每个功能需要输入和输出的地方加了提示，还有在第三个功能实现的时候，如果想要查找的数据超过了的链表范围，程序会输出“wrong input”然后停止运行。另外考虑到，有可能循环链表的元素把单链表的元素全覆盖上了，我添加了一个if语句来判断经过删除元素的单链表长度是否为0，如果为0，则输出提示，然后结束程序运行。综上可以实现程序的目标和程序的流畅化。



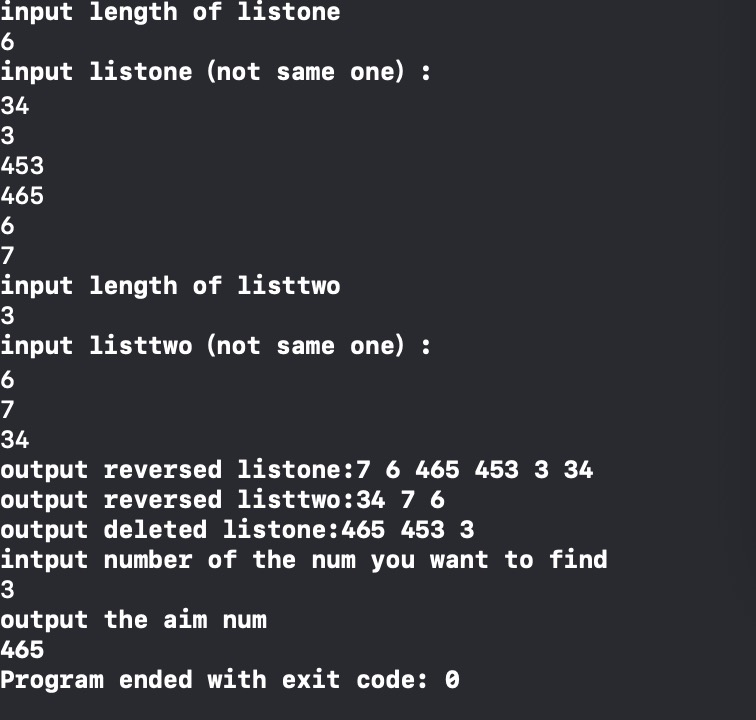
二、 实验方案

（一）实验工具

1.由于本人的电脑是mac下的macos系统，在本次实验用采用在xcode编译器下使用c++语言完成本次实验。

（二）实验结果分析步骤

1.在编写完程序后，输入多组数据进行输出测试，发现可以成功实现所需的功能。如下图：



1. 算法设计及分析

首先展示整个代码实现：

#include <iostream>

#include <cmath>

**using** **namespace** std;

**class** node {

**public**:

**int** num;

    node\* next;

    node(**int** i) {

        num = i;

        next = **nullptr**;

    }

    node() {

        num = 0;

        next = **nullptr**;

    }

};

**class** listone {

**public**:

    node\* head;

    node\* tail;

    listone() {

        head = **new** node(0);

        tail = head;

    }

**void** insertone(**int** i);

**void** removeone(**int** i);

**void** reverseone(listone l);

};

**class** listtwo {

**public**:

    node\* head;

    node\* tail;

    listtwo() {

        head=**new** node(0);

        tail=head;

        tail->next = head;

    }

**void** inserttwo(**int** i);

**void** reversetwo(listtwo l);

};

**void** listone::insertone(**int** i) {

    node\* temp = **new** node(i);

**if** (head == tail) {

        head->next = temp;

        tail = temp;

    }

**else** {

        tail->next = temp;

        tail = temp;

    }

}

**void** listtwo::inserttwo(**int** i) {

    node\* temp = **new** node(i);

**if** (head == tail) {

        head->next = temp;

        tail = temp;

    }

**else** {

        tail->next = temp;

        tail = temp;

        tail->next = head;

    }

}

**void** listone::removeone(**int** i) {

    node\* temp = head;

**if** (i == head->next->num) {

        head->next = head->next->next;

**return** ;

    }

**while** (temp->next->next != **nullptr**) {

**if** (temp->next->num == i) {

            temp->next = temp->next->next;

**return** ;

        }

        temp = temp->next;

    }

**if**(temp->next->num==i){

        temp->next=**nullptr**;

    }

}

**void** listone::reverseone(listone l) {

    node\* p = head->next;

    node\* q = p->next;

    p->next=**nullptr**;

    node\* r=**nullptr**;

**while** (q != **nullptr**) {

        r = q;

        q = r->next;

        r->next=p;

        p=r;

    }

   head->next=r;

    node\* lin=head->next;

**while**(lin!=**nullptr**){

        cout<<lin->num<<" ";

        lin=lin->next;

    }

}

**void** listtwo::reversetwo(listtwo l) {

    node\* p = head->next;

    node\* q = p->next;

    tail=p;

    node\* r=**nullptr**;

**while** (q != head) {

        r = q;

        q = r->next;

        r->next=p;

        p=r;

    }

    head->next=r;

    tail->next=head;

    node\* lin=head->next;

**while**(lin!=head){

        cout<<lin->num<<" ";

        lin=lin->next;

    }

    }

**int** main() {

    listone one;

    listtwo two;

**int** tmp,lenone,i,lentwo;

    cout<<"input length of listone"<<endl;

    cin>>lenone;

    cout<<"input listone（not same one）:"<<endl;

**for**(i=1;i<=lenone;i++){

            cin>>tmp;

            one.insertone(tmp);

        }

    cout<<"input length of listtwo"<<endl;

    cin>>lentwo;

    cout<<"input listtwo（not same one）:"<<endl;

**for**(i=1;i<=lentwo;i++){

                cin>>tmp;

                two.inserttwo(tmp);

            }

    cout<<"output reversed listone:";

    one.reverseone(one);

    cout<<endl;

    cout<<"output reversed listtwo:";

    two.reversetwo(two);

    cout << endl;

    node\* temp1=two.head->next;

**for**(**int** i=0;i<lentwo;i++,temp1=temp1->next){

        one.removeone(temp1->num);

    }

    cout<<"output deleted listone:";

    node\* lin = one.head->next;

**while** (lin!=**nullptr**) {

        cout << lin->num << " ";

        lin = lin->next;

    }

    cout<<endl;

**int** lenonet=0;

    node\* temp0=one.head->next;

**while**(temp0!=**nullptr**){

        lenonet++;

        temp0=temp0->next;

}

**if**(lenonet==0){

        cout<<"deleted list is empty"<<endl;

**return** 0;

    }

    cout<<"intput number of the num you want to find"<<endl;

**int** k;

    cin>>k;

**if**(k>lenonet){

        cout<<"wrong input"<<endl;

    }

**else**{

**int** m;

    m=lenonet-k+1;

    node\* temp3=one.head;

**for**(**int** n=0;n<m;n++){

        temp3=temp3->next;

    }

    cout<<"output the aim num"<<endl;

    cout<<temp3->num<<endl;

    }

**return** 0;

}

下面进行分部解释代码：

1. 结点和单、循环链表定义部分：

**class** node {

**public**:

**int** num;

    node\* next;

    node(**int** i) {

        num = i;

        next = **nullptr**;

    }有参数的结点构造函数

    node() {

        num = 0;

        next = **nullptr**;

    }无参数的结点构造函数

};

**class** listone {

**public**:

    node\* head;

    node\* tail;

    listone() {

        head = **new** node(0);

        tail = head;

    }单链表的定义部分

**void** insertone(**int** i);

**void** removeone(**int** i);

**void** reverseone(listone l);

//分别是insert函数、remove函数、reverse函数的说明

};

**class** listtwo {

**public**:

    node\* head;

    node\* tail;

    listtwo() {

        head=**new** node(0);

        tail=head;

        tail->next = head;

    }循环链表的定义部分

**void** inserttwo(**int** i);

**void** reversetwo(listtwo l);

//分别是insert函数、reverse函数的说明

};

1. 各个类内函数的定义部分：

**void** listone::insertone(**int** i) {//insert函数定义

    node\* temp = **new** node(i);

**if** (head == tail) {

        head->next = temp;

        tail = temp;

}当链表只有一个结点时，数据存入head的next里面。

**else** {

        tail->next = temp;

        tail = temp;

    }不只有一个结点，则新输入的数据依次往下存储。

}

**void** listtwo::inserttwo(**int** i) {

    node\* temp = **new** node(i);

**if** (head == tail) {

        head->next = temp;

        tail = temp;

    }

**else** {

        tail->next = temp;

        tail = temp;

        tail->next = head;

    }

}//与上面的单链表的insert同理，因为是循环链表需要加上tail的next指向head

**void** listone::removeone(**int** i) {//remove函数的定义

    node\* temp = head;

**if** (i == head->next->num) {

        head->next = head->next->next;

**return** ;

    }这是第一种情况，所筛选的元素等于链表的第一个元素，即使head下一个指向下下一个结点即可。

**while** (temp->next->next != **nullptr**) {

**if** (temp->next->num == i) {

            temp->next = temp->next->next;

**return** ;

        }

        temp = temp->next;

}中间数据存储段筛选，直到尾结点的上一个停止，如果遇到相同元素则删掉。  **if**(temp->next->num==i){

        temp->next=**nullptr**;

    }是尾结点的情况，此时temp是尾结点的上一个，则直接将尾结点的上一个指向nullptr即删除尾结点。

}

**void** listone::reverseone(listone l) {//reverse函数定义

    node\* p = head->next;

    node\* q = p->next;

    p->next=**nullptr**;

    node\* r=**nullptr**;

**while** (q != **nullptr**) {

        r = q;

        q = r->next;

        r->next=p;

        p=r;

    }

   head->next=r;

    三指针法进行链表的转置，代码思想：设单链表的头结点为A,设三个指针分别为p,q,r。

1.令p指向空

2.q指向头结点之后的第一个结点

3.r指向q的后继结点，q的后继指针指向p

4.p指向q,q指向r（一定要让p先指向q, 否则q指向r之后,无法再找到p）

5.重复3.4步，直到r==NULL，完成

    node\* lin=head->next;

**while**(lin!=**nullptr**){

        cout<<lin->num<<" ";

        lin=lin->next;

    }输出转置后的链表

}

**void** listtwo::reversetwo(listtwo l) {

    node\* p = head->next;

    node\* q = p->next;

    tail=p;

    node\* r=**nullptr**;

**while** (q != head) {

        r = q;

        q = r->next;

        r->next=p;

        p=r;

    }

    head->next=r;

    tail->next=head;

    node\* lin=head->next;

**while**(lin!=head){

        cout<<lin->num<<" ";

        lin=lin->next;

    }同理单链表的转置逻辑，加了将链表连成循环链表的部分

    }

1. 主函数部分：

**int** main() {

    listone one;

    listtwo two;//定义了两个链表的长度

**int** tmp,lenone,i,lentwo;

    cout<<"input length of listone"<<endl;

    cin>>lenone;//输入单链表长度

    cout<<"input listone（not same one）:"<<endl;

**for**(i=1;i<=lenone;i++){

            cin>>tmp;

            one.insertone(tmp);

        }//输入单链表

    cout<<"input length of listtwo"<<endl;

    cin>>lentwo;//输入循环链表的长度

    cout<<"input listtwo（not same one）:"<<endl;

**for**(i=1;i<=lentwo;i++){

                cin>>tmp;

                two.inserttwo(tmp);

            }//输入循环链表

    cout<<"output reversed listone:";

    one.reverseone(one);转置单链表并输出

    cout<<endl;//换行

    cout<<"output reversed listtwo:";

    two.reversetwo(two);转置循环链表并输出

    cout << endl;//换行

    node\* temp1=two.head->next;

**for**(**int** i=0;i<lentwo;i++,temp1=temp1->next){

        one.removeone(temp1->num);

    }通过for循环从第一个元素遍历循环链表的数据，遇到相同则删除，得到目标单链表

    cout<<"output deleted listone:";

    node\* lin = one.head->next;

**while** (lin!=**nullptr**) {

        cout << lin->num << " ";

        lin = lin->next;

    }//输出目标单链表

    cout<<endl;

**int** lenonet=0;

    node\* temp0=one.head->next;

**while**(temp0!=**nullptr**){

        lenonet++;

        temp0=temp0->next;

}//对经过删除操作后的单链表进行长度统计，

**if**(lenonet==0){

        cout<<"deleted list is empty"<<endl;

**return** 0;

    }//判断长度是否为0，如果为0则输出提示然后结束程序

    cout<<"intput number of the num you want to find"<<endl;

**int** k;

    cin>>k;//输入要查找的倒数第几位

**if**(k>lenonet){

        cout<<"wrong input"<<endl;

    }//如果超范围也输出提示

**else**{

**int** m;

    m=lenonet-k+1;

    node\* temp3=one.head;

**for**(**int** n=0;n<m;n++){

        temp3=temp3->next;

    }//通过上文的算法正向遍历寻找目标结点

    cout<<"output the aim num"<<endl;

    cout<<temp3->num<<endl;

    }//输出目标结点

**return** 0;

}

经过多次🧪试验（10次+）可以验证程序可以成功实现功能，而且程序可以对一些特殊情况进行敏锐作出反应。

1. 复杂度分析

首先来说一下时间复杂度和空间复杂度的表示方法：算法的执行效率，粗略地讲，就是算法代码的执行时间。我们假设每行代码执行的时间都一样，都是1个单位时间，从而算出一段代码总的执行时间为多少个单位时间，然后将公式中的低阶、常量、系数这三个不左右增长趋势的部分忽略，只记录最大量级的表示法。例：T(n)=2n+2就可以记为T(n)=O(n)，T(n)=n2+2就可以记为T(n)=O(n2)。这就是大O时间复杂度表示法，大O时间复杂度实际上并不具体表示代码真正的执行时间，而是表示代码执行时间随数据规模增长的变化趋势，也叫做渐进时间复杂度，简称时间复杂度。

1. 时间复杂度分析：

对于结点和单链表和循环链表类，没有循环没有递归，时间复杂度是O（1）。对于链表的insert函数，也同样没有循环没有递归，时间复杂度同样是O（1）。remove函数有一个while函数，执行是单次递增一次，时间复杂度是O（n）。reverse函数有两个并列的while循环，都是单次递增循环，时间复杂度是O（n）。然后到主函数部分，输入两个链表的代码是两个while循环，单次递增，时间复杂度是O（n）。然后调用reverse函数，时间复杂度是O（n）。然后是在遍历循环链表的循环中调用remove函数，remove的时间复杂度是O（n），类似双重循环，这个地方的时间复杂度是O（n^2）。输出链表用了一个while循环，时间复杂度是O（n），然后是对进行删除操作后的单链表的长度计数循环，采用了一个递增的while循环，时间复杂度是O（n），然后是正向递增寻找所查找的结点，while循环，时间复杂度是O（n）。综上，这个代码的时间复杂度是O（n^2）。

1. 空间复杂度分析：

首先是insert插入函数，插入的话会执行两个操作，插入的节点指向要插入位置的节点，要插入位置的前节点指向插入节点，假设每个节点所占空间的大小为1，则空间复杂度为O(n)。然后是remove删除函数，删除中间某个节点只有一步操作，删除节点的前驱节点指向被删除节点的后驱节点，假设每个节点所占空间的大小为1,空间复杂度为O(n)。然后是reverse函数，在本次实验中，实现reverse功能的算法是三指针法，是在原链表的基础上就地转置，没有多余复制链表，故空间复杂度是O（n）。然后分析主函数里，新键了一些整形变量，空间复杂度都是O（1）。然后创建了两个链表，单链表和循环链表，空间复杂度是O（n），然后对输入的链表进行操作，其中reverse函数没有复制链表，是在原基础上进行操作，所以空间复杂度是O（n），然后是remove操作，也是在原链表的基础上进行删除操作，空间复杂度仍是O（n），然后是查找目标结点，在链表中遍历寻找结点，空间复杂度仍是O（n），新创建了一些node类的结点，都是单个创建，空间复杂度是O（1）。综上本程序的空间复杂度是O（n）。

五、实验总结和程序改进

1.首先是程序流畅度的思考，在上次实验中，发现我在面对一些特殊情况时候，程序无法作出及时的反应，后来在助教学长的提醒下，进行了程序的优化。在这次实验中，我有意的关注了程序应对特殊情况的能力，例如如果one链表的元素被two链表的元素覆盖的情况，以及如果输入要查找的数据以及超出了链表的范围的情况，和考虑到了如果在同一个链表中输入相同元素的处理情况。可以体会到大大提升了程序的流畅度和对代码的理解程度。

