山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：链式描述线性表		
实验学时：2	实验日期： 10.19	

实验目的：
掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。
掌握链表迭代器的实现与应用。

软件开发工具：
Vscode

1. 实验内容
要求封装链表类，链表迭代器类；
链表类需提供操作：在指定位置插入元素，删除指定元素，搜索链表中是否有指定元素，原地逆置链表，输出链表；
不得使用与链表实现相关的 STL。
题目描述：
（1）插入操作：1 idx val，在链表的 idx 位置插入元素 val;
（2）删除操作：2 val，删除链表中的 val 元素。若链表中存在多个该元素，仅删除第一个。若该元素不存在，输出 -1；
（3）逆置操作：3，原地逆置链表；
（4）查询操作：4 val，查询链表中的 val 元素，并输出其索引。若链表中存在多个该元素，仅输出第一个的索引。若不存在该元素，输出 -1；
（5）输出操作：5，使用 链表迭代器 ，输出当前链表索引与元素的异或和：f(chain)=$\sum_{i=0}^{n-1} i \oplus chain[i], n=len(chain)$；
（6）给定两组整数序列，你需要分别创建两个有序链表，使用链表迭代器实现链表的合并，并分别输出这三个有序链表的索引与元素的异或和。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）
（1）插入：从头指针开始找到要找的位置进行插入，若插入位置在头指针，需要改变头指针的地址。
（2）删除：找到元素相等的节点，将前一节点的 next 指向该节点的下一节点，再 delete 该节点
（3）逆置：定义两个指针，一个 pre，一个 temp，分别指向头指针和头指针的下一个，再把头指针指到 temp 的下一个，temp 的 next 指向 pre，再将三个节点往后移，进行循环。
（4）查询：遍历链表找到元素相等的节点，输出索引
（5）输出：用迭代器，计算异或和并输出
（6）合并：有序链表合并，两个指针分别指向两个链表的头节点，比较两个指针的元素大小，将小的放入合并后的链表，指针下移，以此类推。

3. 测试结果（测试输入，测试输出）

```
10 10
6863 35084 11427 53377 34937 14116 5000 49692 70281 73704
4 6863
0
1 2 44199
5
398665
4 21466
-1
1 6 11483
5
410141
4 34937
5
5
410141
4 6863
0
1 10 18635
PS D:\code repository\code>
```

```
3 0
3 1 2
5
0
5
PS D:\code repository\code>
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）
合并的时间复杂度为 0（n）
代码中合并没有用迭代器，用迭代器的话，就是元素为（*i），其他一样。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）
（1）

```
#include <iostream>
using namespace std;

template<class T>
struct chainnode{       //节点类
    T element;
    chainnode<T> *next;
    chainnode(){ }
    chainnode(const T& thelement){  //构造函数
        this->element=thelement;
    }
    chainnode(const T& element,chainnode<T>* next){
        this->element=element;
```

```
        this->next=next;
    }
};

template<class T>
class mylist{                          //链表类
public:
    mylist();
    mylist(const mylist<T>& thelist);
    void insert(int index,const T& thelement);
    void erase(const T& thelement);
    void reverse();
    void find(const T& thelement);

    class iterator{                    //链表成员类迭代器
    public:
        iterator(chainnode<T>* thenode){ node = thenode; }
        T& operator*() const { return node->element; }          //返回节点值
        T* operator->() const { return & node->element; }
        iterator& operator++(){                                 //前++
            node = node->next;
            return *this;
        }
        iterator operator++(int){                               //后++
            iterator last = *this;
            node=node->next;
            return last;
        }
        bool operator!=(const iterator theiter) const{
            return node!=theiter.node ;
        }
        bool operator==(const iterator theiter) const{
            return node==theiter.node;
        }
    private:
        chainnode<T>* node;
    };

    iterator begin(){
        return iterator(firstnode);
    }
    iterator end(){
        return NULL;
    }
    ~mylist(){
            while(firstnode!=NULL){
```

```
                chainnode<T>* temp=firstnode->next;
                delete firstnode;
                firstnode=temp;
            }
        }
private:
    chainnode<T>* firstnode;
    int size;
};

template<class T>
mylist<T>::mylist(){
    firstnode=NULL;
    size=0;
}

template<class T>
mylist<T>::mylist(const mylist<T>& thelist){            //复制构造函数
    size=thelist.size;
    if(size==0){
        firstnode=NULL;
        return ;
    }
    chainnode<T>* temp = thelist.firstnode;          //temp 指针指向被复制的链表
    firstnode = new chainnode<T> (temp->element);
    chainnode<T>* now = firstnode;                    //now 指针指向新链表
    temp = temp->next;
    while(temp!=NULL){
        now -> next = new chainnode<T> (temp->element);   //now 的 next 指向由 temp
的节点值构成的新节点
        now = now->next;
        temp = temp->next;
    }
    now->next = NULL;
}

template<class T>
void mylist<T>::insert(int index,const T& thelement){  //index 从 0 开始算
    size++;
    chainnode<T>* temp = firstnode;
    if(index==0){                                     //插入到头节点
        firstnode = new chainnode<T> (thelement);
        firstnode->next = temp;
        return ;
    }
    for(int i=0;i<index-1;i++){                  //中间或结尾插入
```

```
        temp=temp->next;
    }
    chainnode<T>* p = new chainnode<T> (thelement);
    p->next = temp->next;
    temp->next = p;
}

template<class T>
void mylist<T>::erase(const T& thelement){    //删除节点
    chainnode<T>* temp = firstnode;
    size--;
    if(firstnode->element==thelement){     //删除的是头节点
        firstnode = firstnode->next;
        delete temp;
        return ;
    }
    while(temp->next!=NULL && temp->next->element!=thelement){
        temp=temp->next;
    }
    if(temp->next==NULL){               //没找到该节点
        cout<<-1<<endl;
        return ;
    }
    chainnode<T>* p = temp->next;
    temp->next = p->next;
    delete p;
}

template<class T>
void mylist<T>::reverse(){               //链表颠倒
    chainnode<T>* pre = firstnode;          //前一节点
    chainnode<T>* temp = firstnode->next;   //后一节点
    firstnode = firstnode->next;
    pre->next=NULL;
    while(firstnode->next!=NULL){
        firstnode = firstnode->next;       //firstnode 先移向 temp 的下一节点
        temp->next = pre;                  //temp 的 next 指针指向前一节点
        pre=temp;
        temp=firstnode;
    }
    temp->next=pre;
}

template<class T>
void mylist<T>::find(const T& thelement){           //查找
    chainnode<T>* temp = firstnode;
```

```
    int index=0;
    while(temp!=NULL && temp->element != thelement){
        temp=temp->next;
        index++;
    }
    if(temp==NULL){
        cout<<-1<<endl;
        return ;
    }
    cout<<index<<endl;
}

int main(){
    int n,q;
    cin>>n>>q;
    mylist<int> thelist;
    for(int i=0;i<n;i++){
        int t;
        cin>>t;
        thelist.insert(i,t);
    }
    for(int i=0;i<q;i++){
        int p;
        cin>>p;
        if(p==1){
            int idx,val;
            cin>>idx>>val;
            thelist.insert(idx,val);
        }
        else if(p==2){
            int val;
            cin>>val;
            thelist.erase(val);
        }
        else if(p==3){
            thelist.reverse();
        }
        else if(p==4){
            int val;
            cin>>val;
            thelist.find(val);
        }
        else if(p==5){
            int index=0,sum=0;
            for(mylist<int>::iterator i = thelist.begin();i!=thelist.end();i++){
                sum+=(index++)^(*i);
```

```
            }
            cout<<sum<<endl;
        }
    }
}


 (2)
 #include <iostream>
 using namespace std;

 template<class T>
 struct chainnode{
     T element;
     chainnode<T> *next;
     chainnode(){ }
     chainnode(const T& thelement){
         this->element=thelement;
     }
     chainnode(const T& element,chainnode<T>* next){
         this->element=element;
         this->next=next;
     }
 };

 template<class T>
 class mylist{
 public:
     mylist();
     mylist(const mylist<T>& thelist);
     void insert(int index,const T& thelement);
     void erase(const T& thelement);
     void reverse();
     void find(const T& thelement);
     void merge(const mylist<T>& b);
     void sort_insert(T thelement);
     void merge(mylist<T>& a,mylist<T>& b);
     chainnode<T>* head(){
         return firstnode;
     }

     class iterator{
     public:
         iterator(chainnode<T>* thenode){ node = thenode; }
         T& operator*() const { return node->element; }
         T* operator->() const { return & node->element; }
```

```
        iterator& operator++(){
            node = node->next;
            return *this;
        }
        iterator operator++(int){
            iterator last = *this;
            node=node->next;
            return last;
        }
        bool operator!=(const iterator theiter) const{
            return node!=theiter.node ;
        }
        bool operator==(const iterator theiter) const{
            return node==theiter.node;
        }
    private:
        chainnode<T>* node;
    };

    iterator begin(){
        return iterator(firstnode);
    }
    iterator end(){
        return NULL;
    }
    ~mylist(){
        while(firstnode!=NULL){
            chainnode<T>* temp=firstnode->next;
            delete firstnode;
            firstnode=temp;
        }
    }
private:
    chainnode<T>* firstnode;
    int size;
};

template<class T>
mylist<T>::mylist(){
    firstnode=NULL;
    size=0;
}

template<class T>
mylist<T>::mylist(const mylist<T>& thelist){
    size=thelist.size;
```

```
    if(size==0){
        firstnode=NULL;
        return ;
    }
    chainnode<T>* temp = thelist.firstnode;
    firstnode = new chainnode<T> (temp->element);
    chainnode<T>* now = firstnode;
    temp = temp->next;
    while(temp!=NULL){
        now -> next = new chainnode<T> (temp->element);
        now = now->next;
        temp = temp->next;
    }
    now->next = NULL;
}

template<class T>
void mylist<T>::insert(int index,const T& thelement){  //index 从 0 开始算
    size++;
    chainnode<T>* temp = firstnode;
    if(index==0){
        firstnode = new chainnode<T> (thelement);
        firstnode->next = temp;
        return ;
    }
    for(int i=0;i<index-1;i++){
        temp=temp->next;
    }
    chainnode<T>* p = new chainnode<T> (thelement);
    p->next = temp->next;
    temp->next = p;
}

template<class T>
void mylist<T>::erase(const T& thelement){
    chainnode<T>* temp = firstnode;
    size--;
    if(firstnode->element==thelement){
        firstnode = firstnode->next;
        delete temp;
        return ;
    }
    while(temp->next!=NULL && temp->next->element!=thelement){
        temp=temp->next;
    }
    if(temp->next==NULL){
```

```
        cout<<-1<<endl;
        return ;
    }
    chainnode<T>* p = temp->next;
    temp->next = p->next;
    delete p;
}

template<class T>
void mylist<T>::reverse(){
    chainnode<T>* pre = firstnode;
    chainnode<T>* temp = firstnode->next;
    firstnode = firstnode->next;
    pre->next=NULL;
    while(firstnode->next!=NULL){
        firstnode = firstnode->next;
        temp->next = pre;
        pre=temp;
        temp=firstnode;
    }
    temp->next=pre;
}

template<class T>
void mylist<T>::find(const T& thelement){
    chainnode<T>* temp = firstnode;
    int index=0;
    while(temp!=NULL && temp->element != thelement){
        temp=temp->next;
        index++;
    }
    if(temp==NULL){
        cout<<-1<<endl;
        return ;
    }
    cout<<index<<endl;
}

template<class T>
void mylist<T>::sort_insert(T thelement){
    size++;
    chainnode<T>* temp = firstnode;
    if(firstnode==NULL || thelement < temp->element){
        firstnode = new chainnode<T> (thelement);
        firstnode->next = temp;
        return ;
```

```
        }
        while(temp->next!=NULL && thelement>temp->next->element){
            temp=temp->next;
        }
        chainnode<T>* p = new chainnode<T> (thelement);
        p->next = temp->next;
        temp->next = p;
    }

    template<class T>
    void mylist<T>::merge(mylist<T>& a,mylist<T>& b){
        chainnode<T>* temp_a = a.firstnode;
        chainnode<T>* temp_b = b.firstnode;
        int num=0;
        while(temp_a!=NULL && temp_b!=NULL){
            if(temp_a->element<temp_b->element){
                insert(num,temp_a->element);
                temp_a=temp_a->next;
                num++;
            }
            else{
                insert(num,temp_b->element);
                temp_b=temp_b->next;
                num++;
            }
        }
        while(temp_a!=NULL){
            insert(num,temp_a->element);
            temp_a=temp_a->next;
            num++;
        }
        while(temp_b!=NULL){
            insert(num,temp_b->element);
            temp_b=temp_b->next;
            num++;
        }
    }

    int main(){
        int n,m;
        cin>>n>>m;

        mylist<int> list_n;
        mylist<int> list_m;

        for(int i=0;i<n;i++) {
```

```
        int p;
        cin>>p;
        list_n.sort_insert(p);
    }
    for(int i=0;i<m;i++){
        int p;
        cin>>p;
        list_m.sort_insert(p);
    }

    int index=0,sum=0;
    for(mylist<int>::iterator i = list_n.begin();i!=list_n.end();i++){
        sum+=(index++)^(*i);
    }
    cout<<sum<<endl;

    index=0,sum=0;
    for(mylist<int>::iterator i = list_m.begin();i!=list_m.end();i++){
        sum+=(index++)^(*i);
    }
    cout<<sum<<endl;

    mylist<int> list_merge;
    list_merge.merge(list_n,list_m);

    index=0,sum=0;
    for(mylist<int>::iterator i = list_merge.begin();i!=list_merge.end();i++){
        sum+=(index++)^(*i);
    }
    cout<<sum<<endl;
}
```