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：202200400053 | 姓名： 王宇涵 | | 班级： 22级2班 |
| 实验题目：链式描述线性表 | | | |
| 实验学时：2 | | 实验日期： 2023-10-11 | |
| 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。 | | | |
| 软件开发环境：  Vscode | | | |
| 1. 实验内容   (1) 第一行两个整数 N 和 Q。  第二行 N 个整数，作为节点的元素值，创建链表。  接下来 Q 行，执行各个操作，具体格式如下：  插入操作 : 1 idx val，在链表的 idx 位置插入元素 val;  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；  逆置操作 : 3，原地逆置链表；  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；  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  (2) 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。  Note: 给定序列是无序的，你需要首先得到一个有序的链表   1. 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) C++代码实现了一个基于链表的整数链表数据结构 Chain 和一系列操作，以及一个主程序，通过该程序可以执行不同的操作来修改链表和获取结果Chain 类  push\_back 方法：在链表末尾添加一个元素。  insert 方法：在指定索引位置插入一个元素。  findIndex 方法：查找元素在链表中的索引。  erase 方法：删除链表中的特定元素。  reverse 方法：反转链表。  outputXorSum 方法：计算链表中所有元素的异或和。  get 方法：获取链表中指定索引位置的元素。  output 方法：将链表中的元素输出到标准输出。  chainNode 结构体：表示链表节点，包括元素和指向下一个节点的指针。  **算法思路**  push\_back 方法用于在链表的末尾添加元素，这是一个常数时间操作。  insert 方法用于在指定索引位置插入元素，它通过遍历链表找到插入位置，然后插入新元素。在最坏情况下，时间复杂度为 O(n)，其中 n 是链表的大小。  findIndex 方法用于查找元素在链表中的索引，它需要遍历链表来查找，时间复杂度为 O(n)。  erase 方法用于删除链表中的特定元素，如果元素存在的话。它也需要遍历链表来查找和删除元素，时间复杂度为 O(n)。  reverse 方法用于反转链表，它通过遍历链表并更改节点的指针来实现链表的反转，时间复杂度为 O(n)。  outputXorSum 方法计算链表中所有元素的异或和，并输出结果，这需要线性时间，时间复杂度为 O(n)。  主程序 main 读取输入，并根据不同的操作类型执行相应的操作，包括插入元素、删除元素、反转链表、查找元素索引以及计算异或和.  (2)实现了一个基于链表数据结构（Chain），包括链表的常见操作和合并算法  chainNode 结构体：表示链表节点，包括元素和指向下一个节点的指针。  Chain 类：表示整数链表，包括链表的头节点、尾节点以及链表的大小。  chainBubbleSort 方法：使用冒泡排序对链表元素进行排序。  chainSelectionSort 方法：使用选择排序对链表元素进行排序。  chainInsertSort 方法：使用插入排序对链表元素进行排序。  findRightIndex 方法：辅助方法，查找元素在排序后的链表中应该插入的位置。  meld 函数：将两个链表合并为一个新的链表，并对新链表进行冒泡排序，然后返回合并后的链表。main 函数：从标准输入读取整数 n 和 m，然后创建两个链表 a 和 b，分别读取 n 和 m 个整数并存储在链表中。然后对 a 和 b 进行冒泡排序，最后使用 meld 函数将它们合并成链表 c，并输出 a、b 和 c 的异或和。   1. 测试结果（测试输入，测试输出）   (1)**测试输入**  10 10  6863 35084 11427 53377 34937 14116 5000 49692 70281 73704  4 6863  1 2 44199  5  4 21466  1 6 11483  5  4 34937  5  4 6863  1 10 18635  **输出**  0  398665  -1  410141  5  410141  0  (2)**测试输入**  3 0  3 1 2  **输出**  5 0 5   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   问题:实验二运行超时  影响因素  1. 排序算法  2. meld算法实现  发现  1. 冒泡排序最省时间,插入和选择慢  2. 直接插入a,b所有元素到c中,再进行冒泡排序最省时间,成功AC,其余的算法均慢  **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）**  **第一题代码**  #include<iostream>  int n,q;  using namespace std;  //结点和链表类  template <class T>  struct chainNode  {  T element;  chainNode<T>\* next;  //构造函数  chainNode() = default;  chainNode(const T& element)  {  this->element = element;  next = NULL;  }  chainNode(const T& element, chainNode<T>\* next)  {  this->element = element;  this->next = next;  }  };  template<class T>  class Chain  {  public:  //构造函数  Chain(){};  Chain(const Chain<T>&x)  {  firstNode=x.firstNode;  lastNode=x.lastNode;  listSize=x.listSize;  }  // 迭代器  class iterator  {  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  // 构造函数  iterator(chainNode<T>\* theNode = NULL)  {  node = theNode;  }  // 解引用操作符  T& operator\*() const { return node->element; }  T\* operator->() const { return &node->element; }  // 迭代器加法操作  iterator& operator++() {// 前置自增  node = node->next;  return \*this;  }  iterator operator++(int) {// 后置自增  iterator old = \*this;  node = node->next;  return old;  }  // 相等检验  bool operator!=(const iterator right) const  {  return node != right.node;  }  bool operator==(const iterator right) const  {  return node == right.node;  }  protected:  chainNode<T>\* node;  };  iterator begin() { return iterator(firstNode); }  iterator end() { return iterator(NULL); }  //结束  //函数  void output(ostream &out)const ;  void push\_back(const T &x);  void insert(int theIndex,const T&x);  int findIndex(const T&x)const;  bool erase(const T& val);  void reverse();  void outputXorSum();  int findBiggerIndex(const T&x,int range)const;  void chainInsertSort();  int size(){return listSize;}  T get(int theIndex)const;  //变量  protected:  chainNode<T>\* firstNode=NULL;  chainNode<T>\* lastNode=NULL;  int listSize=0;  };  template<class T>  void Chain<T>::push\_back(const T &x)  {  chainNode<T>\* newNode=new chainNode<T>(x,NULL);  if(lastNode==NULL)  lastNode=firstNode=newNode;  else  {  lastNode->next=newNode;  lastNode=newNode;  }  listSize++;  }  template<class T>  void Chain<T>::insert(int theIndex,const T&x)  {  chainNode<T>\* currentNode=firstNode;  if(theIndex==0)  {  firstNode=new chainNode<T>(x,firstNode);  }  else  {  for(int i=0;i<theIndex-1;i++)  currentNode=currentNode->next;  currentNode->next=new chainNode<T>(x,currentNode->next);  }  listSize++;  //更新lastNode  currentNode=firstNode;  for(int i=0;i<listSize-1;i++)  {  currentNode=currentNode->next;  }  lastNode=currentNode;  return;  }  template<class T>  int Chain<T>::findIndex(const T&x)const  {  chainNode<T>\* currentNode=firstNode;  int index=0;  while(currentNode!=NULL)  {  if(currentNode->element==x)return index;  currentNode=currentNode->next;  index++;  }  return -1;  }  template<class T>  bool Chain<T>::erase(const T& val)  {  chainNode<T>\* currentNode=firstNode;  chainNode<T>\* deleteNode;  int index=findIndex(val);  if(index==-1)  return false;  else if(index==0)  {  deleteNode=firstNode;  firstNode=firstNode->next;  }  else  {  for(int i=0;i<index-1;i++)  currentNode=currentNode->next;  deleteNode=currentNode->next;  currentNode->next=deleteNode->next;  }  delete deleteNode;  listSize--;  //更新lastNode  currentNode=firstNode;  for(int i=0;i<listSize-1;i++)  {  currentNode=currentNode->next;  }  lastNode=currentNode;  return true;  }  template<class T>  void Chain<T>::reverse()  {  chainNode<T>\* currentNode=firstNode;  chainNode<T>\* previousNode=NULL;  chainNode<T>\* nextNode=firstNode;  lastNode=firstNode;  while(currentNode!=NULL)  {  nextNode=currentNode->next;  currentNode->next=previousNode;  previousNode=currentNode;  currentNode=nextNode;  }  firstNode=previousNode;  }  template<class T>  void Chain<T>::output(ostream &out)const  {  chainNode<T>\* currentNode=firstNode;  while(currentNode!=NULL)  {  out<<currentNode->element<<" ";  currentNode=currentNode->next;  }  }  template<class T>  ostream& operator<<(ostream&out,const Chain<T>x)  {  x.output(out);return out;  }  template<class T>  void Chain<T>::outputXorSum()  {  int res=0;  int index=0;  for(Chain<T>::iterator i=Chain<T>::begin();i!=Chain<T>::end();i++)  res+=(\*i)^(index++);  cout<<res<<endl;  }  template<class T>  T Chain<T>::get(int theIndex)const  {  chainNode<T>\* currentNode=firstNode;  for(int i=0;i<theIndex;i++)  currentNode=currentNode->next;  return currentNode->element;  }  int main()  {  cin>>n>>q;  Chain<int>s;  for(int i=0;i<n;i++)  {  int x;cin>>x;s.push\_back(x);  }  while(q--)  {  int op;cin>>op;  switch (op)  {  case 1:  {  int idx,val;cin>>idx>>val;  s.insert(idx,val);  break;  }  case 2:  {  int val;cin>>val;  if(!s.erase(val))cout<<-1<<endl;  break;  }  case 3:  {  s.reverse();  break;  }  case 4:  {  int val;cin>>val;  int index=s.findIndex(val);  if(index==-1)  cout<<-1<<endl;  else cout<<index<<endl;  break;  }  case 5:  {  s.outputXorSum();  break;  }  default:  break;  }  }  return 0;  }    第二题代码  #include<iostream>  using namespace std;  int n,m;  template <class T>  struct chainNode {  T element;  chainNode<T>\* next;  //构造函数  chainNode() = default;  chainNode(const T& element)  {  this->element = element;  next = NULL;  }  chainNode(const T& element, chainNode<T>\* next)  {  this->element = element;  this->next = next;  }  };  template<class T>  class Chain  {  public:  //构造函数  Chain(){};  Chain(const Chain<T>&x)  {  firstNode=x.firstNode;  lastNode=x.lastNode;  listSize=x.listSize;  }  // 迭代器  class iterator  {  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  // 构造函数  iterator(chainNode<T>\* theNode = NULL)  {  node = theNode;  }  // 解引用操作符  T& operator\*() const { return node->element; }  T\* operator->() const { return &node->element; }  // 迭代器加法操作  iterator& operator++() {// 前置自增  node = node->next;  return \*this;  }  iterator operator++(int) {// 后置自增  iterator old = \*this;  node = node->next;  return old;  }  // 相等检验  bool operator!=(const iterator right) const  {  return node != right.node;  }  bool operator==(const iterator right) const  {  return node == right.node;  }  protected:  chainNode<T>\* node;  };  iterator begin() { return iterator(firstNode); }  iterator end() { return iterator(NULL); }  //结束  //函数  void output(ostream &out)const ;  void push\_back(const T &x);  void insert(int theIndex,const T&x);  int findIndex(const T&x)const;  bool erase(const T& val);  void reverse();  void outputXorSum();  int findBiggerIndex(const T&x,int range)const;  void chainInsertSort();  int size(){return listSize;}  T get(int theIndex)const;  void chainBubbleSort() ;  //变量  public:  chainNode<T>\* firstNode=NULL;  chainNode<T>\* lastNode=NULL;  int listSize=0;  };  template<class T>  void Chain<T>::push\_back(const T &x)  {  chainNode<T>\* newNode=new chainNode<T>(x,NULL);  if(lastNode==NULL)  lastNode=firstNode=newNode;  else  {  lastNode->next=newNode;  lastNode=newNode;  }  listSize++;  }  template<class T>  void Chain<T>::insert(int theIndex,const T&x)  {  chainNode<T>\* currentNode=firstNode;  if(theIndex==0)  {  firstNode=new chainNode<T>(x,firstNode);  }  else  {  for(int i=0;i<theIndex-1;i++)  currentNode=currentNode->next;  currentNode->next=new chainNode<T>(x,currentNode->next);  }  listSize++;  //更新lastNode  currentNode=firstNode;  for(int i=0;i<listSize-1;i++)  {  currentNode=currentNode->next;  }  lastNode=currentNode;  return;  }  template<class T>  int Chain<T>::findIndex(const T&x)const  {  chainNode<T>\* currentNode=firstNode;  int index=0;  while(currentNode!=NULL)  {  if(currentNode->element==x)return index;  currentNode=currentNode->next;  index++;  }  return -1;  }  template<class T>  bool Chain<T>::erase(const T& val)  {  chainNode<T>\* currentNode=firstNode;  chainNode<T>\* deleteNode;  int index=findIndex(val);  if(index==-1)return false;  else if(index==0)  {  deleteNode=firstNode;  firstNode=firstNode->next;  }  else  {  for(int i=0;i<index-1;i++)  currentNode=currentNode->next;  deleteNode=currentNode->next;  currentNode->next=deleteNode->next;  }  delete deleteNode;  listSize--;  //更新lastNode  currentNode=firstNode;  for(int i=0;i<listSize-1;i++)  {  currentNode=currentNode->next;  }  lastNode=currentNode;  return true;  }  template<class T>  void Chain<T>::reverse()  {  chainNode<T>\* currentNode=firstNode;  chainNode<T>\* previousNode=NULL;  chainNode<T>\* nextNode=firstNode;  lastNode=firstNode;  while(currentNode!=NULL)  {  nextNode=currentNode->next;  currentNode->next=previousNode;  previousNode=currentNode;  currentNode=nextNode;  }  firstNode=previousNode;  }  template<class T>  void Chain<T>::output(ostream &out)const  {  chainNode<T>\* currentNode=firstNode;  while(currentNode!=NULL)  {  out<<currentNode->element<<" ";  currentNode=currentNode->next;  }  }  template<class T>  ostream& operator<<(ostream&out,const Chain<T>x)  {  x.output(out);return out;  }  template<class T>  void Chain<T>::outputXorSum()  {  if(Chain<T>::size()==0)  {  cout<<0<<endl;  return;  }  int res=0;  int index=0;  for(Chain<T>::iterator i=Chain<T>::begin();i!=Chain<T>::end();i++)  res+=(\*i)^(index++);  cout<<res<<endl;  }  template<class T>  T Chain<T>::get(int theIndex)const  {  chainNode<T>\* currentNode=firstNode;  for(int i=0;i<theIndex;i++)  currentNode=currentNode->next;  return currentNode->element;  }  template<class T>  void Chain<T>::chainBubbleSort()  {  // 对链表元素进行冒泡排序，使链表变为有序链表  if (listSize < 1) return;  chainNode<T> \*p = NULL;  chainNode<T> \*q = NULL;  for (p = firstNode; p != NULL; p = p->next)  {  for (q = p->next; q != NULL; q = q->next)  {  if (p->element > q->element)  {  T tmp = q->element;  q->element = p->element;  p->element = tmp;  }  }  }  }  template<class T>  void Chain<T>::chainInsertSort()  {  //对某一个链表中的元素进行排序  Chain<T> tmp;  for(int i=0;i<listSize;i++)  { //判断每个元素应该在的位置  if(!i)  {  tmp.insert(0,get(0));  continue;  }  int index=tmp.findBiggerIndex(get(i),i);  tmp.insert(index,get(i));  }  firstNode=tmp.firstNode;  lastNode=tmp.lastNode;  }  template<class T>  int Chain<T>::findBiggerIndex(const T&x,int range)const  {  int index=0;  for(;index<range;index++)  if (get(index)>x)break;  return index;  }  // Chain<int> meld(Chain<int>a, Chain<int>b)  // {  // chainNode<int>\* newFirstNode=new chainNode<int>;  // chainNode<int>\* currentNode=newFirstNode;  // chainNode<int>\*nodeA=a.firstNode;  // chainNode<int>\*nodeB=b.firstNode;  // while(nodeA!=NULL&&nodeB!=NULL)  // {  // if(nodeA->element<=nodeB->element)  // {  // currentNode->next=nodeA;  // nodeA=nodeA->next;  // }  // else  // {  // currentNode->next=nodeB;  // nodeB=nodeB->next;  // }  // currentNode=currentNode->next;  // }  // currentNode->next=(nodeA==NULL?nodeB:nodeA);  // Chain<int>c;  // c.firstNode=newFirstNode->next;  // c.listSize=a.size()+b.size();  // return c;  // }  // Chain<int> meld(Chain<int>a,Chain<int>b)  // {  // Chain<int>c(a);  // Chain<int>::iterator i;  // for(i=b.begin();i!=b.end();i++)  // {  // //在c中找到插入的地方  // int index=c.findBiggerIndex((\*i),c.size());  // c.insert(index,(\*i));  // }  // return c;  // }  Chain<int> meld(Chain<int>a,Chain<int>b)  {  Chain<int>c;  for(Chain<int>::iterator it=a.begin();it!=a.end();it++)  {  c.insert(0,(\*it));  }  for(Chain<int>::iterator it=b.begin();it!=b.end();it++)  {  c.insert(0,(\*it));  }  c.chainInsertSort();  return c;  }  int main()  {  cin>>n>>m;  Chain<int>a,b,c;  while(n--)  {  int x;cin>>x;  a.push\_back(x);  }  while(m--)  {  int x;cin>>x;  b.push\_back(x);  }  a.chainInsertSort();  b.chainInsertSort();  //实现c的合并  c=meld(a,b);  a.outputXorSum();  b.outputXorSum();  c.outputXorSum();  } | | | |
|  | | | |