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：201900161140 | 姓名： 张文浩 | | 班级： 19级人工智能 |
| 实验题目：链式描述线性表（第二题） | | | |
| 实验学时：2 | | 实验日期： 9.26 | |
| 实验目的：   1. 掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。 2. 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   **题目描述**：  要求使用题目一中实现的链表类，迭代器类完成本题；  不得使用与题目实现相关的STL；  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。  注：给定序列是无序的，你需要首先得到一个有序的链表。  **输入输出格式**：  **输入**：  第一行两个整数 N 和 M；  第二行 N 个整数，代表第一组整数序列；  第三行 M 个整数，代表第二组整数序列。  **输出**：  三行整数。分别代表第一组数、第二组数对应的有序链表与合并后有序链表的索引与元素的异或和。   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）   第二题的结构体、链表以及迭代器的构建方法与第一题一样。  第二题用到的方式：  Sort排序函数：定义一个左边界，左边界从头结点开始，每次再用一个for循环从左边界开始遍历整个线性表，一旦发现遍历到的节点的值小于左边界节点的值，就把这两个节点的值进行交换，这样每次保证左边界都是左边界开始到最后中最小的元素，等到左边界遍历到最后，就完成了排序过程。  Merge合并两个有序线性表操作：假设要合并的两个线性表已经排好序，用双指针算法，两个指针，分别从两个要合并的线性表的头节点开始遍历，每次将连个节点中小的哪一个节点加入到新的线性表中，直到有一个指针遍历完对应的数组，这个时候如果另一个指针还没有遍历完，就需要把没遍历完的元素直接加到新的线性表中。这样新的线性表也一是有序的。   1. 测试结果（测试输入，测试输出）     两个线性表分别是4 2 3和1 7 3  排好序后是2 3 4和1 3 7  合并的表示1 2 3 3 4 7  2 3 4的异或和是10  1 3 7的异或和是8  1 2 3 3 4 7的异或和是7   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   通过本次实验，我掌握线性表结构、链式描述方法（链式存储结构）、链表的实现。  以及链表迭代器的实现与应用。  链表的实现思路较为简单，但需要注意特殊情况的出现，分类实现功能，例如：删除的元素为第一个元素、插入的元素为第一个元素等。本次题目中，逆置的实现方法稍显复杂，也是最需要仔细的地方。取出第一个节点作为链表终点，第二第三个节点改变指针指向，之后进行循环即可逆置完成。这里的关键在于想出循环的内容，三个节点进行操作之后需要保证相对位置不变，这样才能继续进行下去，最后再将firstNode节点更改即可。  此外，在编写iterator的时候要注意语法规范，否则会出现程序无法运行的情况。编写迭代器也要考虑到所有可能进行的操作，重载尽可能多的运算符，以免在之后的使用中出现编译器报错。   1. 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）   **第二题**  #include <iostream>  using namespace std;  template <class T>  struct chainNode  {      //数据成员      T element;      chainNode<T> \*next;      //方法      chainNode() {}      chainNode(const T &element) { this->element = element; }      chainNode(const T &element, chainNode<T> \*next)      {          this->element = element;          this->next = next;      }  };  template <class T>  class chain  {  public:      chain()      {          firstNode = NULL;          listSize = 0;      }                                 //构造函数      ~chain();                         //析构函数      void insert(int pos, T \_element); //插入函数      void sort();                      //排序函数      void merge(chain &a, chain &b);   //合并函数      void output();      //迭代器      class iterator;      iterator begin() { return iterator(firstNode); }      iterator end() { return iterator(NULL); }      // iterator for chain      class iterator      {      public:          //构造函数          iterator(chainNode<T> \*theNode = NULL)          {              node = theNode;          }          //解引用操作          T &operator\*() const { return node->element; }          T \*operator->() const { return &node->element; }          //迭代器加法操作          iterator &operator++() //前加          {              node = node->next;              return \*this;          }          iterator operator++(int) //后加          {              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;      };  protected:      chainNode<T> \*firstNode; //指向链表第一个节点的指针      int listSize;            //线性表的元素个数  };  template <class T>  void chain<T>::insert(int pos, T \_element)  {      //在pos位置上插入一个\_element元素      if (pos < 0 || pos > listSize)      { //无效索引          cout << "无效索引" << endl;          return;      }      if (pos == 0)      { //在链表头插入          firstNode = new chainNode<T>(\_element, firstNode);      }      else      {          //寻找新元素前驱          chainNode<T> \*p = firstNode;          for (int i = 0; i < pos - 1; i++)          {              p = p->next;          }          //在p之后插入          p->next = new chainNode<T>(\_element, p->next);      }      listSize++;  }  template <class T>  chain<T>::~chain()  {      //链表析构函数，删除链表的所有节点      while (firstNode != NULL)      {          //删除首节点          chainNode<T> \*nextNode = firstNode->next;          delete firstNode;          firstNode = nextNode;      }  }  template <class T>  void chain<T>::output()  {      for (chainNode<T> \*currentNode = firstNode; currentNode != NULL; currentNode = currentNode->next)      {          cout << currentNode->element << " ";      }      cout << endl;  }  template <class T>  void chain<T>::sort()  {      //排序方法是每次把最小的数移动到最左边,左边界a不断左移      if (listSize < 1)          return;      chainNode<T> \*a = NULL;      chainNode<T> \*b = NULL;      for (a = firstNode; a != NULL; a = a->next)      {          for (b = a->next; b != NULL; b = b->next)          {              if (a->element > b->element)              {                  T tmp = b->element;                  b->element = a->element;                  a->element = tmp;              }          }      }  }  template <class T>  void chain<T>::merge(chain &a, chain &b)  {      //使用迭代器实现两个有序链表的合并      int i = 0;      chain::iterator iter1 = a.begin(), iter2 = b.begin();      while (iter1 != NULL && iter2 != NULL)      {          if (\*iter1 < \*iter2)          {              insert(i++, \*iter1);              iter1++;          }          else if (\*iter1 > \*iter2)          {              insert(i++, \*iter2);              iter2++;          }          else          {              insert(i++, \*iter1);              insert(i++, \*iter2);              iter1++;              iter2++;          }      }      if (iter1 != NULL)      {          while (iter1 != NULL)          {              insert(i++, \*iter1);              iter1++;          }      }      if (iter2 != NULL)      {          while (iter2 != NULL)          {              insert(i++, \*iter2);              iter2++;          }      }  }  int main()  {      int N, M, temp;      cin >> N >> M;      chain<int> a, b, c;      for (int i = 0; i < N; i++)      {          cin >> temp;          a.insert(i, temp);      }      for (int i = 0; i < M; i++)      {          cin >> temp;          b.insert(i, temp);      }      a.sort();      b.sort();      c.merge(a, b);      int res = 0, index = 0;      for (chain<int>::iterator iter = a.begin(); iter != a.end(); iter++)      {          res += (\*iter) ^ (index++);      }      cout << res << endl;      res = 0;      index = 0;      for (chain<int>::iterator iter = b.begin(); iter != b.end(); iter++)      {          res += (\*iter) ^ (index++);      }      cout << res << endl;      res = 0;      index = 0;      for (chain<int>::iterator iter = c.begin(); iter != c.end(); iter++)      {          res += (\*iter) ^ (index++);      }      cout << res << endl;      system("pause");      return 0;  } | | | |