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（第一题） | | | |
| 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。  **输入输出格式**：  **输入**：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。；   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）   **第一题：**  构造一个结构体chainnode用来表示链表的一个节点，其中有两个成员，一个是element值，另一个是一个指针，指向下一个节点。  创建链表class类，成员包括一个指针，指向链表第一个节点，另一个表示线性表元素的个数。其中class类中还包括题目要求的操作方法：  Insert插入函数：  插入函数和删除函数的过程很相似，为了在链表中的索引为index的位置上的插入一个新的元素，需要首先找到索引为index-1的元素节点，然后再该节点之后插入新的元素节点。  用一个for循环，找到想要插入位置pos的前面一个节点，然后创造一个新的节点，让前面那个节点指向先创建的节点，县创建的节点的next指向之前那个前驱结点的下一个节点，最后说明线性表长度+1  Erase删除函数：  删除索引为index的元素，要考虑三种情况，①index<0或index>=listsize，这时，删除操作无效，因为没有这个位置上的元素，这种情况可能表示链表为空。②删除非空表的第0个元素节点。③删除其他元素节点  利用for循环遍历线性表，直到遍历到的节点的element的值等于要删除的节点的值，就将这个要删除的节点的前驱节点的next指向要删除节点的下一个节点，同时用delete释放被删除节点的空间，最后说明线性表长度-1.  Reverse逆置操作，维护三个临时节点，一次相连，从前到后依次是pre，cur，aft，这三个指针同时移动，从线性表的头遍历到尾，每次进行的操作是将cur的指向由af改为pre然后一次向右移动，即pre=cur，cur=aft，aft=aft->next，直到遍历完整个线性表。  Search查找操作，遍历数组，同时维护一个pos下标，表示当前遍历到线性表的第几个数了，如果遍历到的节点的element等于要查找到element，就返回pos下标。如果遍历完线性表还没有找到，就输出-1  最有一个操作要构建一个迭代器iteration，构造方法见附录源代码，有了迭代器之后输出索引与元素的异或和就只需要遍历线性表，维护一个pos下标，表示当前节点的索引是几，便于与元素做异或运算。   1. 测试结果（测试输入，测试输出）   **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     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）   **第一题：**  #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 erase(T \_element);           //删除函数      void reverse();                   //逆置函数      void search(T \_element);          //查询函数      void output();                    //输出函数      //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>  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>::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>  void chain<T>::erase(T \_element)  {      //要删除的节点      chainNode<T> \*deleteNode;      if (firstNode->element == \_element)      {          deleteNode = firstNode;          firstNode = firstNode->next;          delete deleteNode;          listSize--;          return;      }      for (chainNode<T> \*pre = firstNode; pre->next; pre = pre->next)      {          if (pre->next->element == \_element)          {              deleteNode = pre->next;              pre->next = deleteNode->next;              delete deleteNode;              listSize--;              return;          }      }      cout << -1 << endl;  }  template <class T>  void chain<T>::reverse()  {      chainNode<T> \*pre = firstNode;      firstNode = firstNode->next;      pre->next = NULL;      chainNode<T> \*aft = firstNode->next;      for (int i = 0; i < listSize - 3; i++)      {          firstNode->next = pre;          pre = firstNode;          firstNode = aft;          aft = aft->next;      }      firstNode->next = pre;      aft->next = firstNode;      firstNode = aft;  }  template <class T>  void chain<T>::search(T \_element)  {      //返回\_element首次出现的索引      //若该元素不存在，则返回-1      chainNode<T> \*currentNode = firstNode;      int pos = 0; //当前节点的索引      while (currentNode != NULL && currentNode->element != \_element)      {          //移向下一个节点          currentNode = currentNode->next;          pos++;      }      //确定是否找到所需的元素      if (currentNode == NULL)      {          cout << -1 << endl;      }      else          cout << pos << endl;  }  template <class T>  void chain<T>::output()  {      for (chainNode<T> \*currentNode = firstNode; currentNode != NULL; currentNode = currentNode->next)      {          cout << currentNode->element << " ";      }  }  int main()  {      int N, Q, temp;      cin >> N >> Q;      chain<int> c;      for (int i = 0; i < N; i++)      {          cin >> temp;          c.insert(i, temp);      }      int op, pos, element;      for (int i = 0; i < Q; i++)      {          cin >> op;          switch (op)          {          case 1:              cin >> pos >> element;              c.insert(pos, element);              break;          case 2:              cin >> element;              c.erase(element);          case 3:              c.reverse();              break;          case 4:              cin >> element;              c.search(element);              break;          case 5:              int res = 0, index = 0;              for (chain<int>::iterator iter = c.begin(); iter != c.end(); iter++)              {                  res += (\*iter) ^ (index++);              }              cout << res << endl;              break;          }      }      system("pause");      return 0;  } | | | |