学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 崔田雨 | **学号** | 2021901365 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术（卓越工程师） |
| 学习情况简述 | | | |
| 根据视频理解练习代码  IMG_20220814_162037IMG_20220814_162044IMG_20220814_162047IMG_20220814_162053IMG_20220814_162056 1660465949813 | | | |
| 本周练习过的代码1 | | | |
| 代码1：带有尾指针的链表  public class LinkedListQueue <E> implements Queue<E>{  private class Node{  public E e;  public Node next;   public Node (E e,Node next){  this.e=e;  this.next=next;  }   public Node(E e){  this(e,null);  }   public Node(){  this(null,null);  }  @Override  public String toString(){  return e.toString();   }  }  private Node head,tail;  private int size;   public LinkedListQueue(){  head=null;  tail=null;  size=0;  }  @Override  public int getSize(){  return size;  }  @Override  public boolean isEmpty(){  return size==0;  }  @Override  public void enqueue(E e){  if(tail==null){  tail=new Node(e);  head=tail;  }  else{  tail.next=new Node(e);  tail=tail.next;  }  size++;  }  @Override  public E dequeue(){  if(isEmpty())  throw new IllegalArgumentException("Cannot dequeue from an empty queue.");  Node retNode=head;  head=head.next;  retNode.next=null;  if(head==null)  tail=null;  size--;  return retNode.e;  }  @Override  public E getFront(){  if(isEmpty())  throw new IllegalArgumentException("Queue is empty.");  return head.e;  }  @Override  public String toString(){  StringBuilder res=new StringBuilder();  res.append("Queue: front");   Node cur=head;  while(cur!=null){  res.append(cur+"->");  cur=cur.next;  }  res.append("NULL tail");  return res.toString();  }  public static void main(String[] args){  LinkedListQueue<Integer> queue=new LinkedListQueue<>();   for(int i=0;i<10;i++){  queue.enqueue(i);  System.*out*.println(queue);   if(i%3==2){  queue.dequeue();  System.*out*.println(queue);  }  }  } }  代码2：对三种链表的性能进行测试  import java.util.Random;  public class Main {  //测试使用q 运行opCount个enqueue和dequeue操作所需要的时间，单位：秒  private static double testQueue(Queue<Integer> q,int opCount){  long startTime=System.*nanoTime*();   Random random=new Random();  for(int i=0;i<opCount;i++)  q.enqueue(random.nextInt(Integer.*MAX\_VALUE*));//0-最大值的数，入队  for(int i=0;i<opCount;i++)  q.dequeue();//出队   long endTime=System.*nanoTime*();  return (endTime-startTime)/1000000000.0;   }  public static void main(String [] args){//测试  int opCount=100000;   ArrayQueue<Integer> arrayQueue=new ArrayQueue<>();  double time1=*testQueue*(arrayQueue,opCount);  System.*out*.println("ArrayQueue, time: "+ time1+" s");   LoopQueue<Integer> loopQueue=new LoopQueue<>();  double time2=*testQueue*(loopQueue,opCount);  System.*out*.println("LoopQueue, time: "+ time2+" s");   LinkedListQueue<Integer> linkedlistQueue=new LinkedListQueue<>();  double time3=*testQueue*(linkedlistQueue,opCount);  System.*out*.println("LinkedListQueue, time: "+ time3+" s");  }  } | | | |
| 本周练习过的代码2 | | | |
| 删除链表中的元素  代码1：ListNode类定义及其输出方式  public class ListNode {  public int val;  public ListNode next;   public ListNode(int x){  val=x;  }  //链表节点的构造函数  //使用arr为参数，创建一个链表，当前的ListNode为链表头结点  public ListNode(int[] arr){  if(arr==null && arr.length==0)  throw new IllegalArgumentException("arr can not be empty");  this.val=arr[0];  ListNode cur=this;  for(int i=1;i<arr.length;i++){  cur.next=new ListNode(arr[i]);  cur=cur.next;  }  }  //以当前节点为头结点的链表信息字符串  @Override  public String toString(){  StringBuilder res=new StringBuilder();  ListNode cur=this;  while(cur!=null) {  res.append(cur.val + "->");  cur = cur.next;  }  res.append("NULL");  return res.toString();  } }  }  代码2：删除元素解决方案1  class Solution {  public ListNode removeElements(ListNode head,int val){  while(head.val==val&& head!=null){  ListNode delNode=head;  head=head.next;  delNode.next=null;  }//删除头节点元素  if(head==null)  return head;  ListNode prev=head;  while(prev.next!=null){  if(prev.next.val==val){  ListNode delNode=prev.next;  prev.next=delNode.next;  delNode.next=null;  }  else  prev=prev.next;  }  return head;  }  public static void main(String[] args){  int [] nums={1,2,6,3,4,5,6};  ListNode head=new ListNode(nums);  System.*out*.println(head);   ListNode res=(new Solution()).removeElements(head,6);  System.*out*.println(res);  } }  代码3：删除元素解决方案2  class Solution2 {  public ListNode removeElements(ListNode head,int val){  ListNode dummyHead=new ListNode(-1);//虚拟头结点的取值无所谓  dummyHead.next=head;  /\* while(head.val==val&& head!=null){  ListNode delNode=head;  head=head.next;  delNode.next=null;  }//删除头节点元素  if(head==null)  return head;  ListNode prev=head;\*/  ListNode prev=dummyHead;  while(prev.next!=null){  if(prev.next.val==val){  /\* ListNode delNode=prev.next;  prev.next=delNode.next;  delNode.next=null;\*/  prev.next=prev.next.next;  }  else  prev=prev.next;  }  return head;  }  public static void main(String[] args){  int [] nums={1,2,6,3,4,5,6};  ListNode head=new ListNode(nums);  System.*out*.println(head);   ListNode res=(new Solution2()).removeElements(head,6);  System.*out*.println(res);  } } | | | |
| 本周练习过的代码3 | | | |
| 代码1：递归求和  public class Sum {//递归求和   public static int sum(int[] arr){  return *sum*(arr,0);  }  //计算arr[l...n]这个区间内所有数字的和  private static int sum(int[] arr, int l){  //当l==arr.length的时候，也就是数组为空  if (l == arr.length){  return 0;  }  return arr[l]+*sum*(arr, l+1);  }   public static void main(String[] args){  int[] nums = {1,2,3,4,5,6,7,8};  System.*out*.println(*sum*(nums));  } }  代码2：用递归解决删除链表的元素  public class Solution3 {  public ListNode removeElements(ListNode head,int val){//穿入一个链表头head  if (head == null)  return null;   ListNode res = removeElements(head.next,val);  if (head.val == val)  return head.next;  else {  head.next = res;  return head;  }  }    public static void main(String[] args) {  Integer[] nums = {1,2,6,3,4,5,6};  ListNode head = new ListNode(nums);//用数组生成的链表用一个head引用来表示，head指向头节点  System.*out*.println(head);   ListNode res = (new Solution3()).removeElements(head,6);  System.*out*.println(res);  } } | | | |