

《 算法与数据结构 》

实验报告本

|  |  |
| --- | --- |
| 班 级： | **计211** |
| 学 号： | **19001531** |
| 姓 名： | **陈正江** |
| 指导教师： | **张 静** |

信息科学与工程学院

2023年 5月

**实 验 报 告 （ 1 ）**

|  |  |
| --- | --- |
| **实验名称**：栈和队列 | **实验地点**：信息楼318 |
| **所使用的工具软件及环境：Win10, Java** | |
| **一、实验目的：**  通过上机实验，要求掌握线性表、栈、队列和字符串的应用算法。 | |
| **二、实验内容描述：**（填写题目内容及输入输出要求）  （1）使用顺序表或链表实现多项式的加法运算。 输入：在一行输入一串数字，两两一组，分别为多项式系数和指数，中间用空格分隔，结束符为“0 0”。如："9 3 1 1 1 0 0 0"表示"多项式9x^3+x+1"。然后输入一个“+”号，继续输入一行多项式数据。  输出：多项式相加后的多项式。如"相加后多项式9x^3+x+1"  （2）假设某火车站采用后进先出的模式。现有n列火车，调度人员给出火车进站的序列，并给出火车出站的序列，判断在这个调度要求能否实现，如果能实现写出火车进站、出站的操作序列。（栈）  输入：第一行为一个正整数N代表火车数量；第二行为N个字母，中间用空格分开，代表N个火车的进站顺序；第三行为N个字母，中间用空格分开，代表N个火车的离站顺序。  输出：第一行输出0或1，0代表该调度无法实现，1代表可以实现；如果可以实现，请在第二行输出进站出站序列。表示进站时在字母后加上‘\_in’，出站加上‘\_out’。  第一题输入：  9 3 1 1 1 0 0 0 +  1 3 3 2 -1 0 0 0  第一题输出： 10 3 3 2 1 1  第二题输入1： 5 A B C D E B C E D A  第二题输出1： 1 A\_in B\_in B\_out C\_in C\_out D\_in E\_in E\_out D\_out A\_out  第二题输入2： 5 A B C D E D A B C E  第二题输出2： 0 | |
| **三、程序运行结果（说明设计思路，解释使用的数据结构，计算时间复杂度）**  **第1题**   1. 实验运行结果截图      1. 数据结构   采用链式存储结构存储一元多项式的非零系数与指数，构造类如下：  public static class PolymialNode {//建立一个多项式节点，包含系数和指数  private int coef;//系数  private int index;//指数  public PolymialNode next;//下一节点的引用  }  public static class PolyList {  PolymialNode head;//头节点  PolymialNode current; //当前单项式节点  }     1. 设计思路   自定义链表创建函数和多项式加法函数。  ·先创建链表表示一元多项式：先创建链表头结点，再利用循环语句将输入内容依次存入链表的数据结点中，实现创建；  while(true) {//链表创建函数  PolymialNode p1=new PolymialNode();  p1.coef=sc.nextInt();  p1.index=sc.nextInt();  if(p1.coef!=0&&p1.index==0) continue;  p.insert(p1);  if(p1.coef==0&&p1.index==0) break;  }  ·再对已经创建好的一元多项式从头结点开始，进行指数比较，指数相同时系数相加并保存到结果多项式result中，反之将把系数大的一项保存到结果多项式result中，并根据保存选定节点指向他们的下一节点；  public static PolyList addPoly(PolyList p, PolyList q) { //多项式加法函数  PolymialNode pnext = p.head.next;  PolymialNode qnext = q.head.next;  PolyList result =new PolyList();  while (pnext != null && qnext != null) {  int pindex = pnext.getindex();  int qindex = qnext.getindex();  int pcoef = pnext.getCoef();  int qcoef = qnext.getCoef();  if (pindex == qindex) {  if (pcoef+qcoef != 0) {//指数相同时系数相加并保存到结果多项式result中  PolymialNode node = new PolymialNode(pcoef + qcoef, pindex);  result.insert(node);  }  pnext = pnext.next;  qnext = qnext.next;  }  else if(pindex > qindex){ //把系数大的放进result  Polymial Nodenode=new PolymialNode(pnext.getCoef(), pnext.getindex());  result.insert(node);  pnext = pnext.next;  }  else{  PolymialNode node=new PolymialNode(qnext.getCoef(), qnext.getindex());  result.insert(node);  qnext = qnext.next; }  }  while (pnext != null) {  PolymialNode node = new PolymialNode(pnext.getCoef(), pnext.getindex());  result.insert(node);  pnext = pnext.next;  }  //p多项式已经完成统计  while (qnext != null) {  PolymialNode node = new PolymialNode(qnext.getCoef(), qnext.getindex());  result.insert(node);  qnext = qnext.next;  }  //q多项式已经完成统计  return result;  }  ·最后按照系数指数依次结果多项式。   1. 时间复杂度   创建多项式链表过程有一个while循环，时间复杂度为O(n)；多项式加法过程中有一个while循环，时间复杂度为O(n)，所以最终程序的时间复杂度为O(n)。  **第2题**   1. 实验运行结果截图      1. 数据结构   使用队列和栈的结构体，结构体如下：  Queue<String> queue = new LinkedList<>();  /\*由于Queue类是一个接口, 需要用其他类作为对象,使用LinkedList作为对象插入删除的时间复杂度为O(1).\*/  Stack<String> stack = new Stack<>();   1. 设计思路   自定义processOutQueue函数获取所有出站进站信息，并用cache保存，与要求进出站格式进行对比，检查调度是否能够实现。  ·先创建栈和链表队列，记录初始进站顺序和要求进出站顺序，传入processOutQueue函数中；  private static void processOutQueue(Queue<String> queue, Stack<String> stack, String outQueue,String string) {  // 如果队列和栈都为空，表示火车已全部进出完成。则保存出站信息  if (queue.isEmpty() && stack.isEmpty()) {  cache[n][0]=outQueue;//出站的情况  cache[n][1]=string; //实现此出站的进出站方式  n++;  outQueue=null;  return;  }  // 队列和栈有两种情况：出栈或进栈  // 一：出栈,已进站的需要更新出站信息outQueue  if (!stack.isEmpty()) {  // 先克隆，保证不会更改原本的队列和栈-----克隆很重要！！！！  Queue<String> tempQueue = new LinkedList<>(queue);  Stack<String> tempStack = (Stack<String>) stack.clone();  String tempstring=string;  String temp=tempStack.pop();//出站  String tempOutQueue = outQueue + (temp + " ");  tempstring+=temp+"\_out ";  processOutQueue(tempQueue, tempStack, tempOutQueue,tempstring);  }  // 二：队列进栈：没进站的继续进站  if (!queue.isEmpty()) {  // 先克隆  Queue<String> tempQueue = new LinkedList<>(queue);  Stack<String> tempStack = (Stack<String>) stack.clone();  String tempstring=string;  String tempString=tempQueue.poll();//进站  tempStack.push(tempString);//压栈  tempstring+=tempString+"\_in ";  processOutQueue(tempQueue, tempStack, outQueue,tempstring);  }  }  ·在函数中有两种情况：出栈或进栈（也可理解为出站或进站）当出栈时,已进站的需要更新出站信息outQueue，当队列进栈时，没进站的继续进站。这两种情况中需要注意先克隆当前栈和队列，保证不会更改原本的队列和栈；  ·最后如果队列和栈都为空，表示火车已全部进出完成，则保存出站信息，并用cache保存，与要求进出站格式进行对比，检查调度是否能够实现。  boolean flag=false; // 排序  for (String string[] : cache) {  if(string[0]==null)break;  if (string[0].equals(arr)) {  flag=true;  System.out.println("1");  System.out.println(string[1]);  }  }  if (flag==false) System.out.println("0");  }   1. 时间复杂度   创建链表队列，记录初始进站顺序和要求进出站顺序时用到一个for循环，时间复杂度为O(n)；processOutQueue函数过程中在检测队列和栈是否为空之后，分别调用递归函数，时间复杂度为O(n)；在用cache与要求进出站格式进行对比时，有for循环，时间复杂度为O(n)，所以最终程序的时间复杂度为O(n)。 | |

2023年5月15日