**第2次作业**

***线性表栈、队列的应用***

* **班级** 软件52
* **学号** 2151601028
* **姓名** 李嘉伟
* **邮箱** 457157005@qq.com

**实验内容**

实验环境(选择在了学校的机房)

**编程语言** Java  
**处理器** 酷睿i4  
**内存** 4G

**1.四种数据结构实现与时间渐进性**  
  
**实现**  
题目限定了java中的接口,所以用java接口来实现，在子类中实现了接口中定义的抽象方法，从而实现这四种数据结构。第一个search函数调用了了另一个遍历取下标函数，使search函数得以实现。不光是这样，在后面使用前驱和后继的代码实现时，相当于是都在调用这个遍历取下标函数，节省了代码量，使相通的代码不至于重复。除此之外，其他的实现较为简单，不做过多的赘述。  
**java代码**  
**接口定义(这个接口基本上可以实现这四种数据结构，其中我还添加了一小部分必要的，比如实现中说的)：**

**//接口**

**interface List**

**{**

**boolean search(int x);**

**boolean insert(int x);**

**boolean isEmpty();**

**int delete(int x);**

**int successor(int x);**

**int predecessor(int x);**

**int minimum();**

**int maximum();**

**int KthElement(int k);**

**}**

**未排序的数组**

boolean search(int x); 遍历所有元素  
boolean insert(int x); 在最后插入  
boolean isEmpty(); 标记判断  
int delete(int x); 遍历所有元素，并将后面的元素移动到前面  
int successor(int x); 先找到指定元素取后继  
int predecessor(int x); 先找到指定元素取前驱  
int minimum(); 最小值需要遍历  
int maximum(); 最大值需要遍历  
int KthElement(int k); 取第k个元素，需要先排序



**//未排序数组**

**class Array implements List**

**{**

**static int INF = 100000000;**

**static int MAXSIZE = 1000;**

**private int[] arr;**

**private int size = 0;**

**Array()**

**{**

**arr = new int[MAXSIZE];**

**}**

**public boolean isEmpty()**

**{**

**return size == 0;**

**}**

**int search\_for\_index(int x)**

**{**

**for(int i = 0;i < size;i++)**

**{**

**if(arr[i] == x)**

**return i;**

**}**

**return -1;**

**}**

**public boolean search(int x)**

**{**

**return search\_for\_index(x) != -1;**

**}**

**public boolean insert(int x)**

**{**

**if(size == MAXSIZE)**

**return false;**

**arr[size++] = x;**

**return true;**

**}**

**public int delete(int x)**

**{**

**for(int i = 0;i < size;i++)**

**{**

**if(arr[i] == x)**

**{**

**for(int j = i+1;j < size;j++)**

**arr[j-1] = arr[j];**

**size--;**

**return i;**

**}**

**}**

**return INF;**

**}**

**public int successor(int x)**

**{**

**int index = search\_for\_index(x);**

**if(index == size-1 || index == -1)**

**return INF;**

**else**

**return arr[index + 1];**

**}**

**public int predecessor(int x)**

**{**

**int index = search\_for\_index(x);**

**if(index == 0 || index == -1)**

**return INF;**

**else**

**return arr[index - 1];**

**}**

**public int minimum()**

**{**

**int m = INF;**

**for(int i = 0 ;i < size;i++) m = m < arr[i] ? m : arr[i];**

**return m;**

**}**

**public int maximum()**

**{**

**int m = -INF;**

**for(int i = 0 ;i < size;i++) m = m > arr[i] ? m : arr[i];**

**return m;**

**}**

**public int KthElement(int k)**

**{**

**if(k > size || k <= 0)**

**return -INF;**

**int[] itemarr = arr.clone();**

**Arrays.sort(temarr,0,size);**

**return itemarr[size - k];**

**}**

**};**

**已排序的数组**

boolean search(int x); 二分搜索  
boolean insert(int x); 插入排序  
boolean isEmpty(); 标记判断  
int delete(int x); 先二分搜索，并将后面的元素移动到前面  
int successor(int x); 先找到指定元素取后继  
int predecessor(int x); 先找到指定元素取前驱  
int minimum(); 最小值数组第一个元素  
int maximum(); 最大值数组最后一个元素  
int KthElement(int k); 取第k个元素，直接读取



**//已排序数组**

**class SortedArray implements List**

**{**

**static int INF = 10000000;**

**static int MAXSIZE = 1000;**

**private int[] arr;**

**private int size = 0;**

**SortedArray()**

**{**

**arr = new int[MAXSIZE];**

**}**

**public boolean isEmpty()**

**{**

**return size == 0;**

**}**

**int search\_for\_index(int x)**

**{**

**//此处使用二分搜索**

**int l = 0;**

**int r = size-1;**

**int mid = l + (r - l)/2;**

**while(l <= r)**

**{**

**if(arr[mid] == x)**

**return mid;**

**else if(arr[mid] > x)**

**r = mid - 1;**

**else**

**l = mid + 1;**

**mid = l + (r - l)/2;**

**}**

**if(arr[l] == x) return l;**

**return -1;**

**}**

**public boolean search(int x)**

**{**

**return search\_for\_index(x) != -1;**

**}**

**public boolean insert(int x)**

**{**

**if(size == MAXSIZE)**

**return false;**

**if(size == 0)**

**{**

**arr[size++] = x;**

**return true;**

**}**

**for(int i = size-1;i >= 0;i--)**

**{**

**if(arr[i] > x) arr[i + 1] = arr[i];**

**else**

**{**

**arr[i + 1] = x;**

**break;**

**}**

**}**

**size++;**

**return true;**

**}**

**public int delete(int x)**

**{**

**int index = search\_for\_index(x);**

**if(index == -1)**

**return INF;**

**size--;**

**for(int i = index + 1;i < size;i++) arr[i-1] = arr[i];**

**return index;**

**}**

**public int successor(int x)**

**{**

**int index = search\_for\_index(x);**

**if(index == size-1 || index == -1)**

**return INF;**

**else**

**return arr[index + 1];**

**}**

**public int predecessor(int x)**

**{**

**int index = search\_for\_index(x);**

**if(index == 0 || index == -1)**

**return INF;**

**else**

**return arr[index - 1];**

**}**

**public int minimum()**

**{**

**if(size == 0) return INF;**

**return arr[0];**

**}**

**public int maximum()**

**{**

**if(size == 0) return -INF;**

**return arr[size-1];**

**}**

**public int KthElement(int k)**

**{**

**if(k > size || k <= 0)**

**return -INF;**

**return arr[size - k];**

**}**

**};**

**未排序的单循环链表**

boolean search(int x); 遍历所有元素  
boolean insert(int x); 在最后插入  
boolean isEmpty(); 标记判断  
int delete(int x); 遍历所有元素找到制定元素,然后拆掉这一段，将剩下两段拼接起来  
int successor(int x); 找到指定元素，取后继元素  
int predecessor(int x); 找到指定元素，取前驱元素  
int minimum(); 最小值需要遍历  
int maximum(); 最大值需要遍历  
int KthElement(int k); 先将链表的数据输出到数组中,取第k个元素，需要先排序，否则没法取



**class LinkList implements List**

**{**

**static int INF = 100000000;**

**class Element**

**{**

**public int value = 0;**

**public Element next = null;**

**};**

**private Element header = null;**

**private int size = 0;**

**public void initLinkList()**

**{**

**header = new Element();**

**header.value = 0;**

**header.next = header;**

**}**

**Element search\_for\_cursor(int x)**

**{**

**Element cur = header;**

**while(cur.next != header)**

**{**

**if(cur.next.value == x)**

**return cur;**

**cur = cur.next;**

**}**

**return cur;**

**}**

**public boolean search(int x)**

**{**

**return search\_for\_cursor(x).next != header;**

**}**

**public boolean insert(int x)**

**{**

**Element cur = search\_for\_cursor(INF);**

**Element exe = new Element();**

**exe.value = x;**

**exe.next = header;**

**cur.next = exe;**

**size ++;**

**return true;**

**}**

**public boolean isEmpty()**

**{**

**return header.next == header;**

**}**

**public int delete(int x)**

**{**

**Element cur = search\_for\_cursor(x);**

**if(cur.next == header)**

**return INF;**

**cur.next = cur.next.next;**

**size --;**

**return x;**

**}**

**public int predecessor(int x)**

**{**

**Element cur = search\_for\_cursor(x);**

**if(cur.next == header || (cur == header && cur.next.value == x))**

**return INF;**

**return cur.value;**

**}**

**public int successor(int x)**

**{**

**Element cur = search\_for\_cursor(x);**

**if(cur.next == header || cur.next.next == header)**

**return INF;**

**return cur.next.next.value;**

**}**

**public int minimum()**

**{**

**int min = INF;**

**Element cur = header;**

**while(cur.next != header)**

**{**

**if(cur.next.value < min)**

**min = cur.next.value;**

**cur = cur.next;**

**}**

**return min;**

**}**

**public int maximum()**

**{**

**int max = -INF;**

**Element cur = header;**

**while(cur.next != header)**

**{**

**if(cur.next.value > max)**

**max = cur.next.value;**

**cur = cur.next;**

**}**

**return max;**

**}**

**public int KthElement(int k)**

**{**

**if(k <=0 || k > size)**

**return INF;**

**int[] arr = new int[size];**

**int i = 0;**

**Element cur = header;**

**while(cur.next != header)**

**{**

**arr[i++] = cur.next.value;**

**cur = cur.next;**

**}**

**Arrays.sort(arr);**

**return arr[size - k];**

**}**

**};**

**已排序的单循环链表**  
由于已排序的单循环链表与未排序的单循环链表在行为上基本一样吧，只有insert函数与KthElement函进行了重写。

boolean insert(int x); 在适当的位置插入， 时间复杂度O(n)  
int KthElement(int k); 直接去链表的第k个元素， O(k)

**class SortedLinkList extends LinkList**

**{**

**public boolean insert(int x)**

**{**

**size++;**

**Element cur = header;**

**while(cur.next != header && cur.next.value < x)**

**cur = cur.next;**

**Element exe = new Element();**

**exe.value = x;**

**exe.next = cur.next;**

**cur.next = exe;**

**return true;**

**}**

**public int KthElement(int k)**

**{**

**if(k <=0 || k > size)**

**return INF;**

**Element cur = header;**

**for(int pos = 0;pos <= size - k;pos++)**

**cur = cur.next;**

**return cur.value;**

**}**

**};**

**2.创建布尔表达式计算器**

**分析**  
对于一个布尔表达式，其计算方式与一个代数表达式的计算方法是相同的，基本思路是将其转换为后缀式后进行计算。但是实际上在转换为后缀的过程中就可以直接计算出结果。方法是：用一个操作数栈来存取操作数，用一个操作符栈存取操作符，遇到操作数的时候直接压栈，遇到操作符的时候，做判断：如果操作符栈为空，那么直接将操作符压入，否则，去除栈顶操作符op2，与op1进行优先级比较，如果op1的优先级要高，那么将op1压栈，不然就将op2弹出栈，并从操作数栈中弹出两个元素进行计算，得出新的结果再重新压入栈中，并且将新的操作符压栈。如果遇到(那么直接压栈，如果遇到)要将()之间的所有操作符依次出栈进行计算，计算完结果压栈。并且在最后一定将操作符栈清空处理。

由于此题目做起来不是那么得心应手，我参考了一些书籍，咨询了一些同学。这个题要有一个清晰的思路，就是先把操作符栈和操作数栈都在纸上自己先构思写一下，然后在想着如何进行代码实现，前前后后进行了多次，也失败了不少次，后来与同学讨论，参考书籍，得到下列java代码。

**使用了java代码实现：**

**Java代码实现**

**public class Computer**

**{**

**static void alert(String message)**

**{**

**System.out.println(message);**

**}**

**static boolean compare(char op1,char op2)**

**{**

**String ops = "!&|(";**

**return ops.indexOf(op1) <= ops.indexOf(op2);**

**}**

**static void calc(char op,Stack<Boolean> stknum)**

**{**

**if(op == '&' || op == '|')**

**{**

**if(stknum.size() < 2)**

**{**

**alert("nums exception");**

**}**

**else**

**{**

**boolean num1,num2;**

**num1 = stknum.pop();**

**num2 = stknum.pop();**

**if(op == '&')**

**{**

**stknum.push(num1 && num2);**

**}**

**else**

**{**

**stknum.push(num1 || num2);**

**}**

**}**

**}**

**else if(op == '!')**

**{**

**if(stknum.size() < 1)**

**{**

**alert("nums exception");**

**}**

**else**

**{**

**stknum.push(!stknum.pop());**

**}**

**}**

**}**

**static boolean parse(String expression)**

**{**

**Stack<Boolean> stknum = new Stack<Boolean>();**

**Stack<Character> stkop = new Stack<Character>();**

**int len = expression.length();**

**for(int i = 0;i < len;i++)**

**{**

**char c = expression.charAt(i);**

**switch(c)**

**{**

**case 'T':**

**stknum.push(true);**

**break;**

**case 'F':**

**stknum.push(false);**

**break;**

**case ')':**

**assert(stkop.isEmpty() == false);**

**char preop = stkop.pop();**

**while(preop != '(')**

**{**

**calc(preop,stknum);**

**assert(stkop.isEmpty() == false);**

**preop = stkop.pop();**

**}**

**break;**

**case '(':**

**stkop.push('(');**

**break;**

**case '&':**

**case '|':**

**case '!':**

**if(stkop.isEmpty())**

**{**

**stkop.push(c);**

**break;**

**}**

**preop = stkop.pop();**

**if(compare(c,preop))**

**{**

**stkop.push(preop);**

**stkop.push(c);**

**}**

**else**

**{**

**calc(preop,stknum);**

**stkop.push(c);**

**}**

**break;**

**default:**

**alert("undefined char");**

**break;**

**}**

**}**

**while(!stkop.isEmpty())**

**{**

**char preop = stkop.pop();**

**calc(preop,stknum);**

**}**

**return stknum.pop();**

**}**

**public static void main(String[] args) {**

**//System.out.println(compare('&','!'));**

**Scanner scan = new Scanner(System.in);**

**while(true)**

**{**

**String expression = scan.next();**

**if(parse(expression))**

**{**

**alert("V");**

**}**

**else**

**{**

**alert("F");**

**}**

**}**

**}**

**};**

**3.利用队列实现基数排序**

**分析**  
一、 实现循环队列部分，(front + 1) mod MAXSIZE back 的时候代表队列为空，而front back的时候代表队列满。这样front实际上指向的是待读取的元素的前一个位置。  
二、实现基数排序部分，这里要分element的类型是字符串或者是整数型，字符串要求等宽，整数型不要求等宽。但是基数排序部分代码是可以复用的，于是我把基数排序部分抽象出来形成一个模块，在java里可以做成接口，在子类中实现那些抽象方法就好了。这样让代码清晰了不少。

**Java代码实现**  
为了增强可扩展性，为元素设计了一个接口，只要能实现该接口的元素类型，都可以使用bucket\_sort函数进行基数排序。

**interface Element**

**{**

**public abstract int numat(int pos);**

**public abstract int length();**

**}**

**//循环队列代码**

**class Queue**

**{**

**int MAXSIZE = 0;**

**Element[] es;**

**int front = 0;**

**int back = 1;**

**Queue()**

**{**

**MAXSIZE = 100;**

**es = new Element[MAXSIZE];**

**}**

**boolean isEmpty()**

**{**

**return (front+1)%MAXSIZE == back;**

**}**

**boolean isFull()**

**{**

**return front == back;**

**}**

**boolean push(Element e)**

**{**

**if(isFull())**

**{**

**return false;**

**}**

**else**

**{**

**es[back] = e;**

**back = (back+1)%MAXSIZE;**

**}**

**return true;**

**}**

**Element pop()**

**{**

**if(isEmpty())**

**{**

**return null;**

**}**

**else**

**{**

**front = (front+1)%MAXSIZE;**

**return es[front];**

**}**

**}**

**Element front()**

**{**

**if(isEmpty())**

**return null;**

**else**

**return es[(front+1)%MAXSIZE];**

**}**

**};**

**//整数型实现了Element接口**

**class MyInteger implements Element**

**{**

**int num;**

**MyInteger(int num)**

**{**

**this.num = num;**

**}**

**public int numat(int pos)**

**{**

**int tem = num;**

**while(pos > 0)**

**{**

**tem /=10;**

**pos--;**

**}**

**return tem % 10;**

**}**

**public int length()**

**{**

**int cnt = 0;**

**int tem = num;**

**while(tem > 0)**

**{**

**cnt ++;**

**tem /= 10;**

**}**

**return cnt;**

**}**

**};**

**//字符串型实现了Element接口**

**class MyString implements Element**

**{**

**String str;**

**MyString(String str)**

**{**

**this.str = str;**

**}**

**public int numat(int pos)**

**{**

**int len = str.length();**

**if(pos >= len)**

**return 0;**

**else**

**return str.charAt(len-1-pos) - 'a';**

**}**

**public int length()**

**{**

**return str.length();**

**}**

**}**

**//对实现了Element接口的元素进行基数排序**

**static void bucket\_sort(Element[] es,int type)**

**{**

**int len = es.length;**

**Queue[] qs;**

**int max\_qs = 0;**

**int max\_size = 0;**

**if(type == 0)**

**max\_qs = 10;**

**else**

**max\_qs = 26;**

**qs = new Queue[max\_qs];**

**for(int i = 0;i < max\_qs;i++) qs[i] = new Queue();**

**for(int i = 0;i < len;i++) max\_size = max\_size > es[i].length()?max\_size:es[i].length();**

**for(int pos = 0;pos < max\_size;pos++)**

**{**

**for(int eid = 0;eid < len;eid++)**

**qs[es[eid].numat(pos)].push(es[eid]);**

**int cur = 0;**

**for(int qid = 0;qid < max\_qs;qid++)**

**while(!qs[qid].isEmpty()) es[cur++] = qs[qid].pop();**

**}**

**}**

**总结：**

总的来说，这次的作业整体难度不大，就是在实现的时候容易出各种各样的问题，接触到了java的接口，实际上大大方便了这次的代码实现，与此同时，也让我温习了一下编写代码的技巧，还有咨询同学和阅读书籍增长了阅历，希望以后可以越来越好吧，也谢谢原盛老师细心地讲解！