|  |  |
| --- | --- |
| **数据结构与算法 实验报告** | |
| 第二次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 宋郅博 |
| **班级** | 强基数学2102 |
| **学号** | 2213711457 |
| **电话** | 13561313919 |
| **Email** | 2105602479@qq.com |

实验1

在本次实验中，主要完成的任务是：

1、为指定的 List ADT（该 ADT 的具体要求请参见文件 List.java）实现三种数据结构：①使用顺序数组做为存储表示； ②使用单向链表做为存储表示； ③使用双向链表做为存储表示。不论使用哪种存储表示实现的数据结构，实验中需要处理的数据类型都是 Character 类型。

2、用给定的输入数据文件验证所实现的数据结构是否正确。

3、给出每种数据结构实现的每个方法在最坏情形下的时间复杂度表示。（使用表格列举结论）

1 .

顺序存储hw2/work\_1/ArrayList.java

以下是随机输入数值所得结果：

import java.io.PrintWriter;

public class ArrayList implements List<Character> {

    private Character[] array;

    private int capacity;

    private int length;

    private int cursor;

    public ArrayList() {

        this.capacity = 512;

        this.array = new Character[capacity];

        this.length = 0;

        this.cursor = -1;

    }

    public void insert(Character newElement) throws ListException {

        if (isFull()) {

            throw new ListException("List is full");

        }

        if (length == 0) {

            array[0] = newElement;

            cursor = 0;

        } else {

            for (int i = length; i > cursor + 1; i--) {

                array[i] = array[i - 1];

            }

            array[cursor + 1] = newElement;

            cursor++;

        }

        length++;

    }

    public void remove() {

        if (isEmpty()) return;

        for (int i = cursor; i < length - 1; i++) {

            array[i] = array[i + 1];

        }

        length--;

        if (length == 0) cursor = -1;

        else if (cursor == length) cursor = 0;

    }

    public void replace(Character newElement) {

        array[cursor] = newElement;

    }

    public void clear() {

        length = 0;

        cursor = -1;

    }

    public boolean isEmpty() {

        return length == 0;

    }

    public boolean isFull() {

        return length == capacity;

    }

    public boolean gotoBeginning() {

        if (isEmpty()) return false;

        cursor = 0;

        return true;

    }

    public boolean gotoEnd() {

        if (isEmpty()) return false;

        cursor = length - 1;

        return true;

    }

    public boolean gotoNext() {

        if (isEmpty() || cursor == length - 1) return false;

        cursor++;

        return true;

    }

    public boolean gotoPrev() {

        if (isEmpty() || cursor == 0) return false;

        cursor--;

        return true;

    }

    public Character getCursor() {

        return array[cursor];

    }

    public void showStructure(PrintWriter pw) {

        if (isEmpty()) {

            pw.println("Empty list {capacity = " + capacity + ", length = 0, cursor = -1}");

            return;

        }

        for (int i = 0; i < length; i++) {

            pw.print(array[i] + " ");

        }

        pw.println("{capacity = " + capacity + ", length = " + length + ", cursor = " + cursor + "}");

    }

    public void moveToNth(int n) {

        if (n < 0 || n >= length) return;

        Character temp = array[cursor];

        remove();

        for (int i = length; i > n; i--) {

            array[i] = array[i - 1];

        }

        array[n] = temp;

        length++;

        cursor = n;

    }

    public boolean find(Character searchElement) {

        if (isEmpty()) return false;

        for (int i = cursor; i < length; i++) {

            if (array[i].equals(searchElement)) {

                cursor = i;

                return true;

            }

        }

        cursor = length - 1;

        return false;

    }

}

单向链表与双向链表在附件hw2/work\_1中，分别为SinglyLinkedList.java, DoublyLinkedList.java

2 .

创建文件Test.java如下以测试：

import java.io.\*;

public class Test {

    public static void main(String[] args) throws Exception {

        List<Character> list = new ArrayList(); // 切换实现类测试

        BufferedReader br = new BufferedReader(new FileReader("list\_testcase.txt"));

        PrintWriter pw = new PrintWriter(System.out);

        String line;

        while ((line = br.readLine()) != null) {

            String[] commands = line.trim().split("\\s+");

            for (String cmd : commands) {

                processCommand(cmd, list, pw);

            }

            list.showStructure(pw);

            pw.flush();

        }

    }

    private static void processCommand(String cmd, List<Character> list, PrintWriter pw) {

        try {

            if (cmd.startsWith("+")) {

                list.insert(cmd.charAt(1));

            } else if (cmd.equals("<")) {

                list.gotoPrev();

            } else if (cmd.equals(">")) {

                list.gotoNext();

            } else if (cmd.startsWith("-")){

                list.remove();

            } else if (cmd.equals("=")) {

                list.replace(cmd.charAt(1));

            } else if (cmd.equals("#")){

                list.gotoBeginning();

            } else if (cmd.equals("\*")){

                list.gotoEnd();

            } else if (cmd.equals("~")){

                list.clear();

            }

        } catch (ListException e) {

            pw.println("Error: " + e.getMessage());

        }

    }

}

最初得到结果在result1.txt，发现虽然长度和光标都对，但是字符错误，经过修改后得到result2.txt，与目标结果相同。

3 .

以下是表格

时间复杂度表

| 方法 | 顺序数组 | 单向链表 | 双向链表 |
| --- | --- | --- | --- |
| insert | O(n) | O(1) | O(1) |
| remove | O(n) | O(n) | O(1) |
| replace | O(1) | O(1) | O(1) |
| clear | O(1) | O(1) | O(1) |
| isEmpty | O(1) | O(1) | O(1) |
| isFull | O(1) | O(1) | O(1) |
| gotoBeginning | O(1) | O(1) | O(1) |
| gotoEnd | O(1) | O(1) | O(1) |
| gotoNext | O(1) | O(1) | O(1) |
| gotoPrev | O(1) | O(n) | O(1) |
| getCursor | O(1) | O(1) | O(1) |
| moveToNth | O(n) | O(n) | O(n) |
| find | O(n) | O(n) | O(n) |

实验2

在本次实验中，完成双端队列抽象数据类型的实现。双端队列是普通队列的一个扩展，其允许在队头和队尾执行入队和出队操作，因此双端队列兼具了栈和队列的操作能力。

1、为指定的 DQueue ADT（该 ADT 的具体要求请参见文件 DQueue.java）实现两种数据结构：

①使用顺序数组做为存储表示； ②使用双向链表做为存储表示。不论使用哪种存储表示实现的数据结构，实验中需要处理的数据类型都是 Integer 类型，且对 ADT 中的每个行为要求在最坏情形下的时间复杂度都必须满足 O(1)。

2、借助合适的数据结构解决如下问题。

问题描述：给定一个包含 n（n>0）个数据的序列，和一个 K（0<K<=n）值，要求计算序列中每 K 个连续元素构成的子序列的最大值。

资源要求：最坏情形下的时间复杂度为 O(n)。

举例：当数据序列为：8、5、10、7、9、4、15、12、90、13 和 K=4，那么每 K 个连续的子序列的最大值分别为：10、10、10、15、15、90 和 90。

随机生成 10 组数据完成测试。

1.

顺序数组(hw2/work\_2/ArrayDQueue.java)

public class ArrayDQueue implements DQueue<Integer> {

    private Integer[] array;

    private int front, rear, size, capacity;

    public ArrayDQueue() {

        capacity = 128;

        array = new Integer[capacity];

        front = -1;

        rear = -1;

        size = 0;

    }

    @Override

    public int size() {

        return size;

    }

    @Override

    public boolean isEmpty() {

        if (size==0) return false;

        return true;

    }

    @Override

    public boolean isFull() {

        if (size==capacity) return true;

        return false;

    }

    @Override

    public void enqueueToRear(Integer element) throws ListException {

        if (element==null) return;

        else if (isFull()==true) throw new ListException("Queue is full");

        else {

            if (isEmpty()==false) {

                front=0;

                rear=0;

            }else rear = (rear + 1) % capacity;//使用模运算，防止数组越界

            array[rear] = element;

            size++;

        }

    }

    @Override

    public void enqueueToFront(Integer element) throws ListException {

        if (element==null) return;

        else if (isFull()==true) throw new ListException("Queue is full");

        else {

            if (isEmpty()==false) {

                front=0;

                rear=0;

            }else front = (front - 1 + capacity) % capacity;

            array[front] = element;

            size++;

        }

    }

    @Override

    public Integer dequeueFromFront() {

        if (isEmpty()==false) return null;

        int tmp=array[front];

        if (front==rear) {

            front=-1;

            rear=-1;

            return null;

        }else front = (front + 1) % capacity;

        size--;

        return tmp;

    }

    @Override

    public Integer dequeueFromRear() {

        if (isEmpty()==false) return null;

        int tmp=array[rear];

        if (front==rear) {

            front=-1;

            rear=-1;

            return null;

        }else rear = (rear - 1 + capacity) % capacity;

        size--;

        return tmp;

    }

    @Override

    public Integer getFront() {

        if (isEmpty()==false) return null;

        return array[front];

    }

    @Override

    public Integer getRear() {

        if (isEmpty()==false) return null;

        return array[rear];

    }

}

双向链表在附件hw2/work\_2/LinkedListDQueue.java

可知，由于没有for或while循环，或者遍历整个数组或链表，所以以上操作的时间复杂度均为O(1)。

2.

运行SlidingWindowMax.java，所设置参数为长度20，K=5，得到以下结果（hw\_2\work\_2\result.txt）：

随机数组为：10 16 1 14 13 17 7 13 3 3 14 1 7 12 5 10 7 14 13 5

结果为：16 17 17 17 17 17 14 14 14 14 14 12 12 14 14 14

随机数组为：3 1 11 0 1 18 1 12 19 17 11 12 8 19 7 11 0 15 19 11

结果为：11 18 18 18 19 19 19 19 19 19 19 19 19 19 19 19

随机数组为：16 9 17 9 13 6 14 3 15 0 19 7 11 5 2 18 10 4 12 13

结果为：17 17 17 14 15 15 19 19 19 19 19 18 18 18 18 18

随机数组为：15 8 15 8 6 2 17 5 11 4 6 2 6 9 4 3 4 1 18 16

结果为：15 15 17 17 17 17 17 11 11 9 9 9 9 9 18 18

随机数组为：8 15 10 4 8 10 17 18 12 12 7 18 16 12 1 15 14 14 15 17

结果为：15 15 17 18 18 18 18 18 18 18 18 18 16 15 15 17

随机数组为：10 17 19 4 13 16 8 1 3 6 11 17 5 18 7 3 13 17 19 2

结果为：19 19 19 16 16 16 11 17 17 18 18 18 18 18 19 19

随机数组为：16 13 1 3 4 16 18 1 6 3 17 5 8 9 16 5 7 14 10 15

结果为：16 16 18 18 18 18 18 17 17 17 17 16 16 16 16 15

随机数组为：19 16 3 7 5 13 8 9 8 7 4 4 1 0 13 10 6 13 17 10

结果为：19 16 13 13 13 13 9 9 8 7 13 13 13 13 17 17

随机数组为：1 1 3 7 18 15 3 8 7 6 12 4 3 15 5 0 19 13 4 4

结果为：18 18 18 18 18 15 12 12 12 15 15 15 19 19 19 19

随机数组为：18 10 4 10 2 1 4 0 4 2 1 7 12 1 17 13 14 9 8 8

结果为：18 10 10 10 4 4 4 7 12 12 17 17 17 17 17 14

实验3

众所周知，栈虽然是一个操作受限的线性表，但是其用途却很广泛，栈的数据结构实现非常简单，因此我们只从应用层面熟悉栈。请完成下面三个子任务。

①递归是一种解决很多复杂问题最简单的思想方法，而任何编程语言对递归程序的支持都是通过栈实现的。请利用课堂上讲解的“Hanoi 塔”问题的非递归转化方法完成对递归快速排序的非递归转化。

②算术混合运算表达式的计算。表达式不仅能处理整数，还需要处理小数。表达式中涉及的运算符包括+、-、\*、/、^(指数)。表达式可以包含括号（只包含圆括号）嵌套，因此要处理括号匹配失败的情形。

③寻找连续的股票最大走势区间。在股票市场中，需要识别股价走势中的连续时间段，这些时间段内股价表现出特定的行为模式，如连续上涨、连续下跌或是横盘整理等。这种分析对于技术分析尤为重要，它帮助交易者和投资者理解市场的动向，判断趋势的持续性或反转的可能性。给定一个记录股票价格的数组 prices，需要生成对应的 s 数组，s[i]中记录的值为 i-j+1，其中i 和 j 必须要满足 prices[j..i-1]中的每一个值都符合prices[j]<=prices[i]。s 数组中的最大值即为连续上涨的最大区间，如图 1 所示。

设计并实现一个算法完成对给定的 prices 数组上求最大连续上涨时间区间值。如果仅使用朴实的设计思想，该算法的时间复杂和空间复杂度是多少？如果使用栈设计的算法，其时间复杂度和空间复杂度是多少？

1. Hanoi塔

代码在hw\_2\work\_3\Hanoi.java

输出如下：  
请输入递归层数: 3

从A移到C

从A移到B

从C移到B

从A移到C

从B移到A

从B移到C

从A移到C

1. 算术混合运算表达式

代码如下：

package work\_3;

import java.util.\*;

public class arithmetic {

    public static double evaluate(String expression) {

        Stack<Double> values = new Stack<>();//操作数栈

        Stack<Character> ops = new Stack<>();//操作符栈

        for (int i = 0; i < expression.length(); i++) {

            char c = expression.charAt(i);

            if (Character.isDigit(c) || c == '.') {//如果是数字或小数点，则直接入栈

                StringBuilder sb = new StringBuilder();

                while (i < expression.length() && (Character.isDigit(expression.charAt(i)) || expression.charAt(i) == '.')) {

                    sb.append(expression.charAt(i++));

                }

                i--;

                values.push(Double.parseDouble(sb.toString()));

            }else if (c == '(') {

                ops.push(c);

            } else if (c == ')') {

                while (ops.peek() != '(') {

                    values.push(applyOp(ops.pop(), values.pop(), values.pop()));

                }

                ops.pop(); // 弹出 '('

            } else if (isOperator(c)) {

                while (!ops.isEmpty() && hasPrecedence(c, ops.peek())) {

                    values.push(applyOp(ops.pop(), values.pop(), values.pop()));

                }

                ops.push(c);

            }

        }

        while (!ops.isEmpty()) {

            values.push(applyOp(ops.pop(), values.pop(), values.pop()));

        }

        return values.pop();

    }

    private static boolean isOperator(char c) {

        return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

    }

    private static boolean hasPrecedence(char op1, char op2) {

        if (op2 == '(' || op2 == ')') {

            return false;

        }

        if ((op1 == '^') && (op2 == '^')) {

            return false; // 右结合

        }

        if (op1 == '^') {

            return false; // ^的优先级最高，除了右结合的情况

        }

        if ((op1 == '\*' || op1 == '/') && (op2 == '+' || op2 == '-')) {

            return true;

        }

        if ((op1 == '+' || op1 == '-') && (op2 == '\*' || op2 == '/')) {

            return false;

        }

        return true;

    }

    private static double applyOp(char op, double b, double a) {

        switch (op) {

        case '+': return a + b;

        case '-': return a - b;

        case '\*': return a \* b;

        case '/': return a / b;

        case '^': return Math.pow(a, b);

        default: throw new IllegalArgumentException("Unknown operator: " + op);

        }

    }

}

1. 连续时间序列

朴素的算法是对于每个i，找到最大的j，使得prices[j] <= prices[j+1] <= ... <= prices[i]。时间复杂度，由于只需要设置一个与序列等长的空数列，因此空间复杂度为。(文件在hw\_2\work\_3\stock\_normal.java)

我们可以使用单调栈优化时间复杂度至：

1. 维护一个单调递减栈，栈中保存索引。
2. 对于每个元素i，找到左边第一个比prices[i]大的元素的位置k，则区间起始位置为j = k + 1。
3. 若栈为空，说明所有左侧元素均小于等于当前元素，区间起始位置为0。
4. 计算s[i] = i - j + 1，最终返回s数组的最大值。

由于只需要创建一个与prices等长的空列，空间复杂度为

代码如下：

package work\_3;

import java.util.Queue;

import java.util.ArrayDeque;

import java.util.Deque;

public class StockIntervalFinder {

    public static int findMaxContinuousRise(double[] prices) {

        int n = prices.length;

        if (n == 0) return 0;

        int[] s = new int[n];

        Deque<Integer> stack = new ArrayDeque<>();

        for (int i = 0; i < n; i++) {

            while (!stack.isEmpty() && prices[stack.peek()] <= prices[i]) {

                stack.pop();

            }

            int j = stack.isEmpty() ? 0 : stack.peek() + 1;

            s[i] = i - j + 1;

            stack.push(i);

        }

        int max = 0;

        for (int length : s) {

            if (length > max) {

                max = length;

            }

        }

        return max;

    }

    public static void main(String[] args) {

        double[] prices = {8, 5, 10, 7, 9, 4, 15, 12, 90, 13};

        int maxLength = findMaxContinuousRise(prices);

        System.out.println("最大连续上涨区间长度: " + maxLength); // 输出: 9

    }

}

实验4

使用自定义的队列数据结构实现对某一个数据序列的排序(采用基数排序)，其中对待排序数据有如下的要求:

①当数据序列是整数类型的数据的时候，数据序列中每个数据的位数不要求等宽，比如: 1、21、12、322、44、123、2312、765、56

②当数据序列是字符串类型的数据的时候，数据序列中每个字符串都是等宽的，比如:"abc","bde","fad","abd","bef","fdd","abe"

注：radixsort1.txt 和 radixsort2.txt 是为上面两个数据序列提供的测试数据。

创建RadixSort.java,代码如下：

package work\_4;

import java.util.Arrays;

import java.util.LinkedList;

import java.util.Queue;

public class RadixSort {

    public static void sort(int[] arr) {

        int max = Arrays.stream(arr).max().orElse(0);

        for (int exp = 1; max / exp > 0; exp \*= 10)

            countSort(arr, exp);

    }

    private static void countSort(int[] arr, int exp) {

        Queue<Integer>[] buckets = new LinkedList[10];

        for (int i = 0; i < 10; i++)

            buckets[i] = new LinkedList<>();

        for (int num : arr)

            buckets[(num / exp) % 10].offer(num);

        int idx = 0;

        for (Queue<Integer> bucket : buckets)

            while (!bucket.isEmpty())

                arr[idx++] = bucket.poll();

    }

}

1. 整数队列

创建hw2/work\_4/MainStart1.java来使用radixSort1.txt并排序

所得结果保存在sort1\_result.txt

1. 字符串

创建hw2/work\_4/MainStart22.java来使用radixSort2.txt并排序

所得结果保存在sort2\_result.txt