**Juan Sebastián Bernal Jiménez - 1022434946  
Guillermo Alejandro Cano Rodriguez - 1121888697  
Jhon Edison Prieto Artunduaga - 1007163058  
Jonatan Javier Valero Comayan - 1007491800**  
 **Estructuras de Datos – Grupo 3**  
 **4 de mayo de 2023**

**PARCIAL EJERCICIOS ESTRUCTURAS SECUENCIALES - UNCODE**

1. **Robots:**

import java.util.LinkedList;

import java.util.ListIterator;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

LinkedList<Integer> robPos = new LinkedList<Integer>();

LinkedList<Integer> robNeg = new LinkedList<Integer>();

Scanner scanner = new Scanner(System.in);

String input = scanner.nextLine();

scanner= new Scanner(input);

int m;

while (scanner.hasNext()) {

m= scanner.nextInt();

if(m>0){

robPos.add(m);

} else if (m<0){

robNeg.add(m);

}

}

scanner.close();

String output="";

Integer pos, neg;

while(robPos.isEmpty()==false && robNeg.isEmpty()==false) {

pos = robPos.getLast();

neg = robNeg.getFirst();

if(pos>-neg){

robNeg.pollFirst();

} else if(-neg>pos) {

robPos.pollLast();

} else {

robNeg.pollFirst();

robPos.pollLast();

}

}

if (!robPos.isEmpty()) {

output += robPos.pollFirst();

while (!robPos.isEmpty()) {

output += " " + robPos.pollFirst();

}

} else if (!robNeg.isEmpty()) {

output += robNeg.pollFirst();

while (!robNeg.isEmpty()) {

output += " " + robNeg.pollFirst();

}

} else {

output = "No quedaron robots!";

}

System.out.print(output);

}

}

1. **Elementos Comunes:**

import java.util.\*;

public class Main {

public static class Node {

public int registro;

public Node next;

public Node prev;

public Node(int registro) {

this.registro = registro;

this.next = null;

this.prev = null;

}

}

public static class ListaDoubCirc {

public Node head;

public Node tail;

public int size = 0;

public ListaDoubCirc() {

this.head = null;

this.tail = null;

}

public boolean isVacia() {

if (this.head == null || this.tail == null) {

return true;

} else {

return false;

}

}

public void insertEnd(Node newNode) {

if (this.tail == null) {

this.head = newNode;

this.tail = newNode;

} else {

this.tail.next = newNode;

newNode.prev = this.tail;

this.tail = newNode;

}

this.size++;

}

public void insertFirst(Node newNode) {

if (this.head == null) {

this.head = newNode;

this.tail = newNode;

} else {

newNode.next = this.head;

this.head.prev = newNode;

this.head = newNode;

}

this.size++;

}

public void arregloListaTransform(String[] listaNumeros) {

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

Node nuevo = new Node(Integer.parseInt(listaNumeros[i]));

this.insertEnd(nuevo);

}

}

public Node getNodo(int index) {

if (this.isVacia() || index < 0 || index >= this.size) {

return null;

}

Node puntero = this.head;

int indice = 0;

while (indice < index && puntero != null) {

puntero = puntero.next;

indice++;

}

return puntero;

}

public void reverse() {

if (this.head == null || this.head == this.tail) {

return;

}

Node current = this.head;

Node prev = null;

Node next = null;

while (current != this.tail) {

next = current.next;

current.next = prev;

current.prev = next;

prev = current;

current = next;

}

current.next = prev;

current.prev = null;

this.tail = this.head;

this.head = current;

}

@Override

public String toString() {

String a = "";

if (this.isVacia() == true) {

return "";

} else {

for (int i = 0; i < this.size; i++) {

if (i == this.size - 1) {

a = a.concat(String.valueOf(getNodo(i).registro));

return a;

}

a = a.concat(String.valueOf(getNodo(i).registro) + " ");

}

return a;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String input1 = scanner.nextLine();

String input2 = scanner.nextLine();

String[] arr1 = input1.split(" ");

String[] arr2 = input2.split(" ");

ListaDoubCirc listaIni1 = new ListaDoubCirc();

ListaDoubCirc listaIni2 = new ListaDoubCirc();

listaIni1.arregloListaTransform(arr1);

listaIni2.arregloListaTransform(arr2);

ListaDoubCirc listaCopy = new ListaDoubCirc();

ListaDoubCirc listaCopy2 = new ListaDoubCirc();

int tamaño = listaIni1.size;

int tamaño1 = listaIni2.size;

if (tamaño != tamaño1) {

System.out.print("Las entradas deben ser de igual tamaño");

System.exit(0);

}

listaIni2.reverse();

Node node1 = listaIni1.head;

Node node2 = listaIni2.head;

while (node1 != null && node2 != null) {

if (node1.registro == node2.registro) {

node1 = node1.next;

node2 = node2.next;

} else {

if (node1 != null) {

listaCopy.insertEnd(node1);

node1 = node1.next;

}

if (node2 != null) {

listaCopy2.insertEnd(node2);

node2 = node2.next;

}

}

}

listaCopy2.reverse();

System.out.println(listaCopy);

System.out.print(listaCopy2);

}

}

1. **¿ Digan en qué casos de prueba fallaron, cuál creen que era la causa y cómo la solucionaron?**

Los casos de prueba que menos se generaron y corregimos más fácil fueron: ***presentation error - compilation error***. El primero era por algún espacio o salto de línea no deseado y el segundo por errores de indentación. Ya después, el de ***wrong answer*** salía un poco más, pero en sí el problema no era tanto las respuestas incorrectas sino que por apuntadores nulos, se dejaban de tener en cuenta elementos. El más recurrente fue el de ***Runtime Error***, que como mencioné anteriormente, fue debido al uso incorrecto, hasta ese momento, de apuntadores dentro de sentencias de control for