Alvarado Valdivia Anderson | 21140131

Desarrollo de los Ejercicios del Examen parcial de Estructura de Datos

1) Se tiene dos arreglos, uno de los registros de estudiantes y el otro de los registros de empleados. Cada registro de un estudiante contiene campos para el apellido, primer nombre y una calificación (de 0 a 20). Cada registro de un empleado contiene campos para el apellido, primer nombre y salario. Ambos arreglos están ordenados en orden alfabético por el apellido y primer nombre. Dos registros con el mismo nombre y apellido no aparecen en el mismo arreglo (no hay registros repetidos en un arreglo)
   Escriba una acción, que de un aumento del 10% de salario para cada empleado que tenga un registro de estudiantes y cuya calificación sea mayor que 14.

PseudoCodigo:

```
Acción aplicarAumento(registrosEstudiantes, registrosEmpleados)
Inicio
        Para cada estudiante en registrosEstudiantes hacer
                Si estudiante.calificacion > 14 Entonces
                        apellido <- estudiante.apellido
                        nombre <- estudiante.nombre
                        Para cada empleado en registrosEmpleados hacer
                                Si empleado.apellido = apellido y
                                empleado.nombre = nombre Entonces
                                        salarioActual <- empleado.salario
                                        aumento <- salarioActual * 0.1
                                        nuevoSalario <- salarioActual + aumento
                                        empleado.salario <- nuevoSalario
                                        Salir
                                FinSi
                        FinPara
                FinSi
        FinPara
Fin
```

Codigo Java:

```
//Clase Registro de Estudiantes, con atributos Apellido , Nombre y Calificación.
class RegistroEstudiante {
   private String apellido;
   private String nombre;
   private int calificacion;

   //Constructor de la clase RegistroEstudiante.
   public RegistroEstudiante(String apellido, String nombre, int calificacion) {
```

```
        this.apellido = apellido;
        this.nombre = nombre;
        this.calificacion = calificacion;
    }

    // Método getter para obtener el Apellido.
    public String getApellido() {
        return apellido;
    }

    // Método getter para obtener el Nombre.
    public String getNombre() {
        return nombre;
    }

    // Método getter para obtener la Calificación.
    public int getCalificacion() {
        return calificacion;
    }
}

//Clase Registro de Empleados, con atributos Apellido , Nombre y Salario.
class RegistroEmpleado {
    private String apellido;
    private String nombre;
    private double salario;

    //Constructor de la clase RegistroEmpleado.
    public RegistroEmpleado(String apellido, String nombre, double salario) {
        this.apellido = apellido;
        this.nombre = nombre;
        this.salario = salario;
    }

    // Método getter para obtener el Apellido.
    public String getApellido() {
        return apellido;
    }

    // Método getter para obtener el Nombre.
    public String getNombre() {
        return nombre;
    }

    // Método getter para obtener el Salario.
    public double getSalario() {
        return salario;
    }

    // Método setter para establecer el Salario.
    public void setSalario(double salario) {
        this.salario = salario;
    }
```

```
    // Método toString para representar el registro de empleado como una cadena de texto
    public String toString() {
        return apellido + ", " + nombre + " -> Salario: $ " + salario;
    }
}

class AumentoSalario {

    // Método para aplicar el aumento de salario a los empleados que cumplen los criterios
    public void aplicarAumento(RegistroEstudiante[] registrosEstudiantes, RegistroEmpleado[]
registrosEmpleados) {
        for (RegistroEstudiante estudiante : registrosEstudiantes) {
            if (estudiante.getCalificacion() > 14) {
                String apellido = estudiante.getApellido();
                String nombre = estudiante.getNombre();
                for (RegistroEmpleado empleado : registrosEmpleados) {
                    if (empleado.getApellido().equals(apellido) && empleado.getNombre().equals(nombre)) {
                        double salarioActual = empleado.getSalario();
                        double aumento = salarioActual * 0.1;
                        double nuevoSalario = salarioActual + aumento;
                        empleado.setSalario(nuevoSalario);
                        break;
                    }
                }
            }
        }
    }
}

//Clase principal del programa.
public class App {
    public static void main(String[] args) {
        // Arreglos de registros de estudiantes y empleados
        RegistroEstudiante[] registrosEstudiantes = {
                new RegistroEstudiante("Alvarado", "Anderson", 18),
                new RegistroEstudiante("Capac", "Carlos", 17),
                new RegistroEstudiante("Gutierres", "Dario", 16)
        };

        RegistroEmpleado[] registrosEmpleados = {
                new RegistroEmpleado("Alvarado", "Anderson", 2000),
                new RegistroEmpleado("Barreto", "Cristian", 1500),
                new RegistroEmpleado("Gutierres", "Dario", 1800)
        };

        // Instancia de la clase AumentoSalario y llamada al método aplicarAumento
        AumentoSalario aumentoSalario = new AumentoSalario();
        aumentoSalario.aplicarAumento(registrosEstudiantes, registrosEmpleados);

        // Imprimir los registros actualizados de empleados
        System.out.println("Registro luego del aumento:");
        for (RegistroEmpleado empleado : registrosEmpleados) {
```

```
        System.out.println(empleado);
      }
    }
}
```

2) Se dice que una matriz tiene un punto de silla si el valor de alguna posición (fila, columna) es el menor valor de su fila y a la vez el mayor de su columna. Escriba una acción que tenga como entrada una matriz A(M, N) y calcule la posición de un punto de silla (si es que existe).

PseudoCodigo:

```
Accion encontrarPuntoDeSilla(matriz[][])
Inicio
        filas <- longitud(matriz)
        columnas <- longitud(matriz)
        Para i desde 0 hasta filas hacer
                valorMinimoFila <- ValorMáximoEntero
                columnaMinimoFila <- -1
                Para j desde 0 hasta columnas hacer
                        Si matriz[i][j] < valorMinimoFila Entonces
                                valorMinimoFila <- matriz[i][j]
                                columnaMinimoFila <- j
                        FinSi
                FinPara

                esPuntoDeSilla <- Verdadero

                Para k desde 0 hasta filas hacer
                        Si matriz[k][columnaMinimoFila] > valorMinimoFila Entonces
                                esPuntoDeSilla <- Falso
                                Salir
                        FinSi
                FinPara

                Si esPuntoDeSilla Entonces
                        Retornar [i, columnaMinimoFila]
                FinSi
        FinPara
        Retornar nulo
Fin
```

Codigo Java:

```
//Método para buscar un punto de silla en una matriz.
class PuntoDeSilla {
   public int[] encontrarPuntoDeSilla(int[][] matriz) {
      int filas = matriz.length;
      int columnas = matriz[0].length;

      // Recorrer cada fila de la matriz
      for (int i = 0; i < filas; i++) {
         int valorMinimoFila = Integer.MAX_VALUE;
```

```
            int columnaMinimoFila = -1;

            // Encontrar el valor mínimo de la fila y su columna correspondiente
            for (int j = 0; j < columnas; j++) {
                if (matriz[i][j] < valorMinimoFila) {
                    valorMinimoFila = matriz[i][j];
                    columnaMinimoFila = j;
                }
            }

            boolean esPuntoDeSilla = true;

            // Verificar si el valor mínimo de la fila es el máximo en su columna
            for (int k = 0; k < filas; k++) {
                if (matriz[k][columnaMinimoFila] > valorMinimoFila) {
                    esPuntoDeSilla = false;
                    break;
                }
            }

            // Si se encontró un punto de silla, retornar su posición.
            if (esPuntoDeSilla) {
                return new int[]{i, columnaMinimoFila};
            }
        }

        // Si no se encontró un punto de silla, retornar null.
        return null;
    }
}

public class App {
    public static void main(String[] args) {
        int[][] matriz = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        PuntoDeSilla buscarPuntoDeSilla = new PuntoDeSilla();

        // Llamar al método encontrarPuntoDeSilla.
        int[] puntoDeSilla = buscarPuntoDeSilla.encontrarPuntoDeSilla(matriz);

        // Mostrar el resultado obtenido
        if (puntoDeSilla != null) {
            int fila = puntoDeSilla[0];
            int columna = puntoDeSilla[1];
            int valor = matriz[fila][columna];
            System.out.println("Punto de silla - posición (" + fila + ", " + columna + ")");
            System.out.println("Punto de silla - Valor: " + valor);
        } else {
            System.out.println("No se encontró un punto de silla en la matriz");
```

```
    }
  }
}
```

4) Una lista circular de cadenas está ordenada alfabéticamente, El único puntero (variable apuntador) apunta al nodo final de la lista, el cual tiene la dirección del nodo alfabéticamente mayor, este nodo final apunta al nodo alfabéticamente menor. Construya las acciones para realizar lo siguiente:
   a) Añadir una nueva palabra a la lista, en el orden que le corresponda.
   b) Eliminar una palabra dada de la lista.

PseudoCodigo:

A)

```
Acción añadirPalabra(palabra)
Inicio
        nuevoNodo <- nuevo Nodo(palabra)
        Si vacia Entonces
                nuevoNodo.siguiente <- nuevoNodo
                nuntero <- nuevoNodo
        Sino Si nuevaPalabra > puntero.palabra Entonces
                nuevoNodo.siguiente <- puntero.siguiente
                nuntero.siguiente <- nuevoNodo
                nuntero <- nuevoNodo
        Sino
                actual <- puntero.siguiente
                anterior <- Nulo

                Mientras actual != puntero && nuevaPalabra > actual.palabra Hacer
                anterior <- actual
                actual <- actual.siguiente
                MinMientras

                nuevoNodo.siguiente <- actual
                anterior.siguiente <- nuevoNodo
        FinSi
Fin
```

B)

```
Acción eliminarPalabra(palabra)
Inicio
        Si vacia Entonces
                Retornar
        FinSi

        actual <- puntero.siguiente
        anterior <- puntero

        Hacer
                Si actual.palabra = palabraEliminar Entonces
```

```
                anterior.siguiente <- actual.siguiente

                Si actual = puntero Entonces
                        puntero <- anterior
                FinSi

                Retornar
                FinSi

                anterior <- actual
                actual <- actual.siguiente
        Mientras actual diferente de puntero.siguiente
Fin
```

Codigo Java:

```
//Clase que representa un nodo en una lista circular.
class Nodo {
   String palabra;
   Nodo siguiente;

   //Constructor de la clase Nodo.
   public Nodo(String palabra) {
      this.palabra = palabra;
      this.siguiente = null;
   }
}

//Clase que representa una lista circular de palabras.
class ListaCircular {
   Nodo puntero;

   //Constructor de la clase ListaCircular.
   public ListaCircular() {
      this.puntero = null;
   }

   //Verifica si la lista está vacía.
   public boolean vacia() {
      return puntero == null;
   }

   //Metodo para añadir una nueva palabra a la lista en orden alfabético.
   public void añadirPalabra(String nuevaPalabra) {
      Nodo nuevoNodo = new Nodo(nuevaPalabra);

      if (vacia()) {
         nuevoNodo.siguiente = nuevoNodo;
         puntero = nuevoNodo;
      } else if (nuevaPalabra.compareTo(puntero.palabra) > 0) {
         nuevoNodo.siguiente = puntero.siguiente;
         puntero.siguiente = nuevoNodo;
         puntero = nuevoNodo;
      } else {
```

```
            Nodo actual = puntero.siguiente;
            Nodo anterior = null;

            while (actual != puntero && nuevaPalabra.compareTo(actual.palabra) > 0) {
                anterior = actual;
                actual = actual.siguiente;
            }

            nuevoNodo.siguiente = actual;
            anterior.siguiente = nuevoNodo;
        }
    }

    //Metodo para eliminar una palabra de la lista.
    public void eliminarPalabra(String palabraEliminar) {
        if (vacia()) {
            return;
        }

        Nodo actual = puntero.siguiente;
        Nodo anterior = puntero;

        do {
            if (actual.palabra.equals(palabraEliminar)) {
                anterior.siguiente = actual.siguiente;

                if (actual == puntero) {
                    puntero = anterior;
                }

                return;
            }

            anterior = actual;
            actual = actual.siguiente;
        } while (actual != puntero.siguiente);
    }

    //Metodo para imprimir los elementos de la lista en orden.
    public void imprimirLista() {
        if (vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        Nodo actual = puntero.siguiente;

        do {
            System.out.println(actual.palabra);
            actual = actual.siguiente;
        } while (actual != puntero.siguiente);
    }
}
```

```
//Clase principal que contiene el método main para ejecutar el programa.
public class App {
   public static void main(String[] args) throws Exception {
      ListaCircular lista = new ListaCircular();

      lista.añadirPalabra("ComputaciónCientífica");
      lista.añadirPalabra("Estadística");
      lista.añadirPalabra("InvestigaciónOperativa");
      lista.añadirPalabra("Sistemas");

      System.out.println("Lista original:");
      lista.imprimirLista();

      System.out.println("Añadimos una nueva palabra:");
      lista.añadirPalabra("Matemática");
      lista.imprimirLista();

      System.out.println("Eliminamos una palabra:");
      lista.eliminarPalabra("Estadística");

      System.out.println("Lista después de eliminar 'Estadística':");
      lista.imprimirLista();
   }
}
```

5) Escriba una acción recursiva que invierta el orden de los elementos de una pila.

PseudoCodigo:

```
Acción invertirPila(Pila)
Inicio
   Si pilaVacia(pila) Entonces
      Retornar
   FinSi

   elemento <- desempilar(pila)   //Guardar el elemento superior de la pila

   invertirPila(pila)  // Llamada recursiva para invertir el resto de la pila

   insertarAlFondo(pila, elemento)  //Insertar el elemento al fondo de la pila
FinAcción

Acción insertarAlFondo(Pila, elemento)
Inicio
   Si pilaVacia(pila) Entonces
      empilar(pila, elemento)  //Si la pila está vacía, simplemente apilamos el elemento
   Sino
      cima <- desempilar(pila)  //Desapilar el elemento superior
      insertarAlFondo(pila, elemento)  //Llamada recursiva para insertar en el fondo

      empilar(pila, cima)  //Apilar nuevamente el elemento desapilado
   FinSi
```

Fin

Codigo Java:

```
//Clase Nodo, con atributos valor y siguiente.
//Valor: almacena el valor del nodo.
//Siguiente: indica el siguiente nodo en la lista.
class Nodo{
   int valor;
   Nodo siguiente;

   //Constructor de la clase Nodo.
   public Nodo(int valor){
      this.valor = valor;
      this.siguiente = null;
   }
}

//Clase Pila, con atributos tamanio y cima.
//Tamanio: tamaño de la pila.
//Cima: nodo superior de la pila.
class Pila{
   int tamanio;
   Nodo cima;

   //Constructor de la clase Pila.
   public Pila(){
      this.tamanio = 0;
      this.cima = null;
   }

   //Verifica si la pila esta vacia.
   public boolean vacia(){
      return cima == null;
   }

   //Metodo para empilar un elemento a la cima de la pila.
   //Crea un nuevo nodo con el valor asignado.
   public void empilar(int valor){
      Nodo nuevoNodo = new Nodo(valor);
      nuevoNodo.siguiente = cima;
      cima = nuevoNodo;
      tamanio++;
   }

   ////Metodo para desempilar un elemento de la cima de la pila.
   public int desempilar(){
      if(vacia()){
         System.out.print("La pila esta vacia.");
         return -1; //Valor invalido.
      }

      int auxiliar = cima.valor;
```

```
    cima = cima.siguiente;
    tamanio--;
    return auxiliar;
}

//Retorna el valor del elemento en cima de la pila.
public int cima(){
    if(vacia()){
        System.out.print("La pila esta vacia.");
        return -1; //Valor invalido.
    }

    return cima.valor;
}

//Retorna el tamaño de la pila.
public int tamanio(){
    if(tamanio > 0){
        return tamanio;
    }else{
        return 0;
    }
}

//Metodo para invertir el orden de los elementos en la pila.
//Este metodo usa la recursion para invertir la pila.
public void invertirPila(){
    if(vacia()){
        return;
    }

    int elemento = desempilar();
    invertirPila();
    insertarAlFondo(elemento);
}

//Ingresa un elemeton al fondo de la pila.
//Este metodo desempila la Pila para ingresar el elemento al fondo.
public void insertarAlFondo(int elemento){
    if(vacia()){
        empilar(elemento);
    }else{
        int cima = desempilar();
        insertarAlFondo(elemento);
        empilar(cima);
    }
}

//Metodo para mostrar en pantalla los elementos de la pila.
//Imprime la pila en orden, desde la cima hasta el fondo.
public void imprimir(){
    if(vacia()){
        System.out.println("La pila esta vacia.");
```

```
        }

        Nodo nodoActual = cima;
        while(nodoActual != null){
            System.out.print(nodoActual.valor + " ");
            nodoActual = nodoActual.siguiente;
        }

        System.out.println();
    }
}

//Clase principal del programa.
public class App {
    public static void main(String[] args) throws Exception {
        //Creamos la pila.
        Pila pila = new Pila();

        //Se agregan valores a la pila.
        pila.empilar(1);
        pila.empilar(2);
        pila.empilar(3);
        pila.empilar(4);
        pila.empilar(5);

        //Se imprime la pila inicialmente.
        System.out.println("Pila: ");
        pila.imprimir();

        //Se invierte la pila.
        pila.invertirPila();

        //Se imprime la pila invertida.
        System.out.println("Pila invertida: ");
        pila.imprimir();
    }
}
```