Universidad de San Carlos de Guatemala
Facultad de Ingeniería

PRÁCTICA #2
MANUAL TÉCNICO
USAC DATA ANALYZER

Introducción a la Programación 1

Auxiliar Sebastian Alejandro Velasquez Bonilla

Guatemala, marzo de 2025

Introducción

El presente documento detalla los aspectos técnicos de la aplicación "USAC Processing Data", desarrollada como solución para la Práctica 2 del curso de Introducción a la Programación y Computación 1. Esta aplicación fue diseñada para procesar datos desde archivos CSV, visualizarlos en forma gráfica y aplicar diferentes algoritmos de ordenamiento, mostrando cada paso del proceso de manera visual y generando reportes con información relevante.

La aplicación fue desarrollada utilizando el lenguaje de programación Java, implementando interfaces gráficas y siguiendo el patrón de diseño MVC (Modelo-Vista-Controlador) para mantener una estructura organizada y modular del código.

Requerimientos del Sistema

Para el correcto funcionamiento de la aplicación, se requiere:

- JDK 11 o superior instalado
- Sistema operativo: Windows, macOS o Linux
- Memoria RAM: 2GB mínimo recomendado
- Espacio en disco: 100MB mínimo
- Resolución de pantalla mínima recomendada: 1024x768

Arquitectura del Sistema

La aplicación sigue el patrón arquitectónico Modelo-Vista-Controlador (MVC), separando claramente las responsabilidades:

- **Modelo**: Contiene las clases que representan y manipulan los datos.
- **Vista**: Comprende todas las interfaces gráficas que interactúan con el usuario.
- **Controlador**: Maneja la lógica de negocio y conecta el modelo con la vista.

Estructura del Proyecto

El proyecto está organizado en los siguientes paquetes:

- `modelo`: Contiene las clases que representan los datos y la lógica de negocio.
- `vista`: Contiene las clases que implementan las interfaces gráficas.
- `controlador`: Contiene las clases que conectan el modelo con la vista.
- `util`: Contiene clases de utilidad como los algoritmos de ordenamiento.

Descripción de Clases y Métodos

Paquete modelo

Clase `Dato.java`

Representa un elemento individual de datos que será procesado y graficado.

Atributos principales:

- `String categoria`: Almacena el nombre de la categoría.
- `int contador`: Almacena el valor numérico asociado a la categoría.

Métodos principales:

- `getCategoria()`: Devuelve la categoría.
- `getContador()`: Devuelve el contador.
- `setCategoria(String categoria)`: Establece la categoría.
- `setContador(int contador)`: Establece el contador.

Clase `Archivo.java`

Maneja la lectura del archivo CSV y la extracción de datos.

Métodos principales:

- `leerArchivo(String ruta)`: Lee el archivo CSV y extrae los datos.
- `validarExtension(String ruta, String extension)`: Valida que la extensión del archivo sea la correcta según la sección.
- `extraerEncabezados(String linea)`: Extrae los encabezados del archivo.
- `extraerDatos(String linea)`: Extrae los datos de una línea del archivo.

Paquete vista

Clase `VentanaPrincipal.java`

Implementa la interfaz gráfica principal de la aplicación.

Componentes principales:

- Campo para la ruta del archivo
- Botón para buscar el archivo
- Botón para aceptar y procesar el archivo
- Panel para mostrar la gráfica

- Panel para las opciones de ordenamiento

Métodos principales:

- `inicializarComponentes()`: Inicializa los componentes gráficos.
- `mostrarGrafica(List<Dato> datos)`: Muestra la gráfica con los datos cargados.
- `mostrarDialogoOrdenamiento()`: Muestra el diálogo con las opciones de ordenamiento.

Clase `VentanaOrdenamiento.java`

Implementa la interfaz gráfica para visualizar el proceso de ordenamiento.

Componentes principales:

- Panel para mostrar la gráfica durante el ordenamiento
- Etiquetas para mostrar información del ordenamiento (tiempo, pasos, etc.)

Métodos principales:

- `inicializarComponentes()`: Inicializa los componentes gráficos.
- `actualizarGrafica(List<Dato> datos)`: Actualiza la gráfica con los datos actuales durante el ordenamiento.
- `actualizarInformacion(String algoritmo, String velocidad, String tipo, long tiempo, int pasos)`: Actualiza la información mostrada sobre el ordenamiento.

Paquete controlador

Clase `ControladorPrincipal.java`

Maneja la lógica principal de la aplicación y conecta el modelo con la vista.

Métodos principales:

- `cargarArchivo(String ruta)`: Carga el archivo y extrae los datos.
- `iniciarOrdenamiento(String algoritmo, String velocidad, String tipo)`: Inicia el proceso de ordenamiento.
- `generarReporte(List<Dato> datosOriginales, List<Dato> datosOrdenados, String algoritmo, String velocidad, String tipo, long tiempo, int pasos)`: Genera el reporte PDF.

Paquete util

Clase `AlgoritmosOrdenamiento.java`

Implementa los diferentes algoritmos de ordenamiento utilizados en la aplicación.

Métodos principales:

- `bubbleSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Bubble Sort.
- `insertSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Insert Sort.
- `selectSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Select Sort.
- `mergeSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Merge Sort.
- `quickSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Quick Sort.
- `shellSort(List<Dato> datos, boolean ascendente, OrdenamientoObserver observer)`: Implementa el algoritmo Shell Sort.

Interfaz `OrdenamientoObserver.java`

Define los métodos que deben implementar las clases que deseen observar el proceso de ordenamiento.

Métodos:

- `actualizarOrdenamiento(List<Dato> datos, int pasos, long tiempo)`: Notifica a los observadores sobre el estado actual del ordenamiento.

Algoritmos de Ordenamiento

Bubble Sort

Compara pares adyacentes de elementos y los intercambia si están en el orden incorrecto. El proceso se repite hasta que no se requieren más intercambios.

```java
// Bubble Sort
private void bubbleSort(boolean ascendente) {
    int n = datosActuales.length;

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            // Verificar si debemos continuar
            if (cancelado)
                return;

            boolean debeIntercambiar;
            if (ascendente) {
                debeIntercambiar = datosActuales[j].getCount() > datosActuales[j + 1].getCount();
            } else {
                debeIntercambiar = datosActuales[j].getCount() < datosActuales[j + 1].getCount();
            }

            if (debeIntercambiar) {
                // Intercambiar elementos
                DataPoint temp = datosActuales[j];
                datosActuales[j] = datosActuales[j + 1];
                datosActuales[j + 1] = temp;

                // Actualizar contador y visualización
                contadorPasos++;
                SwingUtilities.invokeLater(this::actualizarVisualizacion);

                // Pausar para mostrar el progreso
                pausar();
            }
        }
    }
}
```

Implementación de Hilos

La visualización del proceso de ordenamiento se realiza utilizando hilos para permitir que la interfaz gráfica se mantenga responsiva mientras se ejecuta el algoritmo de ordenamiento.

```java
private void iniciarOrdenamiento() {
    // Reiniciar datos
    clonarDatos();
    contadorPasos = 0;
    cancelado = false;
    ordenando = true;

    // Estimar el número total de pasos basado en el algoritmo seleccionado
    String algoritmo = (String) comboAlgoritmo.getSelectedItem();
    int n = datosActuales.length;

    // Estimar pasos según algoritmo
    switch (algoritmo) {
        case "Bubble Sort":
            totalPasosEstimados = (n * n) / 2; // O(n²)
            break;
        case "Insert Sort":
            totalPasosEstimados = (n * n) / 4; // O(n²) pero generalmente más eficiente
            break;
        case "Select Sort":
            totalPasosEstimados = (n * n) / 2; // O(n²)
            break;
        case "Merge Sort":
        case "Quick Sort":
        case "Shell Sort":
            totalPasosEstimados = (int) (n * Math.log(n)); // O(n log n)
            break;
        default:
            totalPasosEstimados = n * n;
    }
```

```java
// Configurar la barra de progreso
progressBar.setMinimum(n:0);
progressBar.setMaximum(totalPasosEstimados);
progressBar.setValue(n:0);

// Capturar la imagen inicial antes de ordenar
imagenInicial = capturarGrafica(panelVisualizacion);

// Configurar UI
btnOrdenar.setEnabled(b:false);
btnCancelar.setEnabled(b:true);
comboAlgoritmo.setEnabled(b:false);
comboVelocidad.setEnabled(b:false);
radioAscendente.setEnabled(b:false);
radioDescendente.setEnabled(b:false);

// Iniciar temporizador
tiempoInicio = System.currentTimeMillis();
timerActualizacion.start();
```

```
// Iniciar hilo de ordenamiento
hiloOrdenamiento = new Thread(() -> {
    try {
        String algoritmoSeleccionado = (String) comboAlgoritmo.getSelectedItem();
        boolean ascendente = radioAscendente.isSelected();

        switch (algoritmoSeleccionado) {
            case "Bubble Sort":
                bubbleSort(ascendente);
                break;
            case "Insert Sort":
                insertSort(ascendente);
                break;
            case "Select Sort":
                selectSort(ascendente);
                break;
            case "Merge Sort":
                // El merge sort requiere un array auxiliar
                DataPoint[] aux = new DataPoint[datosActuales.length];
                mergeSort(izq:0, datosActuales.length - 1, aux, ascendente);
                break;
            case "Quick Sort":
                quickSort(bajo:0, datosActuales.length - 1, ascendente);
                break;
            case "Shell Sort":
                shellSort(ascendente);
                break;
        }

        // Actualizar UI una última vez después de completar
        SwingUtilities.invokeLater(() -> {
            actualizarVisualizacion();
            finalizarOrdenamiento(completado:true);
        });
```

```
    } catch (Exception ex) {
        ex.printStackTrace();
        SwingUtilities.invokeLater(() -> {
            finalizarOrdenamiento(completado:false);
            JOptionPane.showMessageDialog(this,
                    "Error durante el ordenamiento: " + ex.getMessage(),
                    title:"Error",
                    JOptionPane.ERROR_MESSAGE);
        });
    }
});
```

Generación de Reportes

La aplicación genera reportes en formato PDF utilizando la biblioteca iText.

```
private void generarReporte() {
    try {
        // Crear directorio principal de reportes si no existe
        File dirReportes = new File(pathname:"Reportes");
        if (!dirReportes.exists()) {
            dirReportes.mkdirs();
        }

        // Obtener nombre del algoritmo y crear subcarpeta
        String algoritmo = (String) comboAlgoritmo.getSelectedItem();
        File dirAlgoritmo = new File(dirReportes, algoritmo);
        if (!dirAlgoritmo.exists()) {
            dirAlgoritmo.mkdirs();
        }

        // Nombre del archivo
        SimpleDateFormat sdf = new SimpleDateFormat(pattern:"yyyyMMdd_HHmmss");
        String fechaHora = sdf.format(new Date());
        String direccion = radioAscendente.isSelected() ? "Asc" : "Desc";
        String nombreArchivo = "Reporte_" + algoritmo.replace(target:" ", replacement:"") + "_" + direccion + "_" + fechaHora + ".pdf";

        File archivoReporte = new File(dirAlgoritmo, nombreArchivo);

        // Crear documento PDF (iText 5)
        Document document = new Document(PageSize.A4);
        PdfWriter.getInstance(document, new FileOutputStream(archivoReporte));
        document.open();

        // Establecer márgenes
        document.setMargins(marginLeft:36, marginRight:36, marginTop:36, marginBott…36);

        // Estilo de fuentes
        Font fontTitle = FontFactory.getFont(FontFactory.HELVETICA_BOLD, size:16);
        Font fontSubtitle = FontFactory.getFont(FontFactory.HELVETICA_BOLD, size:14);
        Font fontNormal = FontFactory.getFont(FontFactory.HELVETICA, size:12);
```

```
        // Título del reporte
        Paragraph title = new Paragraph(string:"REPORTE DE ORDENAMIENTO", fontTitle);
        title.setAlignment(Element.ALIGN_CENTER);
        document.add(title);

        // Información del estudiante
        document.add(new Paragraph(string:"\nNombre: Christian Javier Rivas Arreaga", fontNormal));
        document.add(new Paragraph(string:"Carné: 202303204", fontNormal));

        // Fecha y hora
        document.add(new Paragraph("Fecha y hora: " +
                new SimpleDateFormat(pattern:"dd/MM/yyyy HH:mm:ss").format(new Date()), fontNormal));

        // Separador
        document.add(new Paragraph(string:"\n"));

        // Información del ordenamiento (tabla)
        PdfPTable tableInfo = new PdfPTable(numColumns:2);
        tableInfo.setWidthPercentage(widthPercentage:100);
        try {
            tableInfo.setWidths(new float[] { 1f, 2f });
        } catch (DocumentException e) {
            e.printStackTrace();
        }

        // Agregar filas a la tabla de información
        addRowToTable(tableInfo, label:"Algoritmo:", algoritmo, fontNormal);
        addRowToTable(tableInfo, label:"Dirección:", radioAscendente.isSelected() ? "Ascendente" : "Descendente",
                fontNormal);
        addRowToTable(tableInfo, label:"Velocidad:", (String) comboVelocidad.getSelectedItem(), fontNormal);
        addRowToTable(tableInfo, label:"Tiempo total:", obtenerTiempoFormateado(), fontNormal);
        addRowToTable(tableInfo, label:"Total de pasos:", String.valueOf(contadorPasos), fontNormal);

        document.add(tableInfo);
```

```java
// Datos mínimo y máximo
int min = Integer.MAX_VALUE;
int max = Integer.MIN_VALUE;
String catMin = "";
String catMax = "";

for (DataPoint dp : datosActuales) {
    if (dp.getCount() < min) {
        min = dp.getCount();
        catMin = dp.getCategory();
    }
    if (dp.getCount() > max) {
        max = dp.getCount();
        catMax = dp.getCategory();
    }
}

document.add(new Paragraph("\nDato mínimo: " + catMin + " (" + min + ")", fontNormal));
document.add(new Paragraph("Dato máximo: " + catMax + " (" + max + ")", fontNormal));

// Título para los datos originales
Paragraph originalTitle = new Paragraph(string:"DATOS ORIGINALES", fontSubtitle);
originalTitle.setAlignment(Element.ALIGN_CENTER);
originalTitle.setSpacingBefore(spacing:20);
originalTitle.setSpacingAfter(spacing:15);
document.add(originalTitle);

// Tabla con datos originales
PdfPTable tableOriginal = new PdfPTable(numColumns:2);
tableOriginal.setWidthPercentage(widthPercentage:80);
tableOriginal.setHorizontalAlignment(Element.ALIGN_CENTER);

// Encabezados de tabla
PdfPCell headerCell1 = new PdfPCell(new Phrase(string:"Categoría", fontNormal));
headerCell1.setHorizontalAlignment(Element.ALIGN_CENTER);
tableOriginal.addCell(headerCell1);
```

```java
PdfPCell headerCell2 = new PdfPCell(new Phrase(string:"Valor", fontNormal));
headerCell2.setHorizontalAlignment(Element.ALIGN_CENTER);
tableOriginal.addCell(headerCell2);

// Datos originales
for (DataPoint dp : datosOriginales) {
    PdfPCell cell1 = new PdfPCell(new Phrase(dp.getCategory(), fontNormal));
    tableOriginal.addCell(cell1);

    PdfPCell cell2 = new PdfPCell(new Phrase(String.valueOf(dp.getCount()), fontNormal));
    cell2.setHorizontalAlignment(Element.ALIGN_RIGHT);
    tableOriginal.addCell(cell2);
}

document.add(tableOriginal);

// Agregar la imagen original
try {
    if (imagenInicial != null) {
        document.add(new Paragraph(string:"\n"));
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ImageIO.write(imagenInicial, formatName:"png", baos);
        Image img = Image.getInstance(baos.toByteArray());

        // Escalar imagen para que se ajuste a la página
        float width = document.getPageSize().getWidth() * 0.8f;
        img.scaleToFit(width, fitHeight:300);
        img.setAlignment(Element.ALIGN_CENTER);

        document.add(img);
    }
} catch (Exception e) {
    document.add(new Paragraph("\nNo se pudo incluir la gráfica original: " + e.getMessage(), fontNormal));
    e.printStackTrace();
}
```

```java
// Título para los datos ordenados
Paragraph orderedTitle = new Paragraph(string:"DATOS ORDENADOS", fontSubtitle);
orderedTitle.setAlignment(Element.ALIGN_CENTER);
orderedTitle.setSpacingBefore(spacing:30);
orderedTitle.setSpacingAfter(spacing:15);
document.add(orderedTitle);

// Tabla con datos ordenados
PdfPTable tableOrdered = new PdfPTable(numColumns:2);
tableOrdered.setWidthPercentage(widthPercentage:80);
tableOrdered.setHorizontalAlignment(Element.ALIGN_CENTER);

// Encabezados de tabla
PdfPCell headerCell1Copy = new PdfPCell(new Phrase(string:"Categoría", fontNormal));
headerCell1Copy.setHorizontalAlignment(Element.ALIGN_CENTER);

PdfPCell headerCell2Copy = new PdfPCell(new Phrase(string:"Valor", fontNormal));
headerCell2Copy.setHorizontalAlignment(Element.ALIGN_CENTER);

tableOrdered.addCell(headerCell1Copy);
tableOrdered.addCell(headerCell2Copy);

// Datos ordenados
for (DataPoint dp : datosActuales) {
    PdfPCell cell1 = new PdfPCell(new Phrase(dp.getCategory(), fontNormal));
    tableOrdered.addCell(cell1);

    PdfPCell cell2 = new PdfPCell(new Phrase(String.valueOf(dp.getCount()), fontNormal));
    cell2.setHorizontalAlignment(Element.ALIGN_RIGHT);
    tableOrdered.addCell(cell2);
}

document.add(tableOrdered);
```

```
    // Agregar la imagen final (actual)
    try {
        document.add(new Paragraph(string:"\n"));
        BufferedImage imagenFinal = capturarGrafica(panelVisualizacion);
        if (imagenFinal != null) {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(imagenFinal, formatName:"png", baos);
            Image img = Image.getInstance(baos.toByteArray());

            // Escalar imagen para que se ajuste a la página
            float width = document.getPageSize().getWidth() * 0.8f;
            img.scaleToFit(width, fitHeight:300);
            img.setAlignment(Element.ALIGN_CENTER);

            document.add(img);
        }
    } catch (Exception e) {
        document.add(new Paragraph("\nNo se pudo incluir la gráfica ordenada: " + e.getMessage(), fontNormal));
        e.printStackTrace();
    }

    // Cerrar el documento
    document.close();

    // Mostrar mensaje de éxito
    JOptionPane.showMessageDialog(this,
            "Reporte generado exitosamente en:\n" + archivoReporte.getAbsolutePath(),
            title:"Reporte Generado",
            JOptionPane.INFORMATION_MESSAGE);
```

```
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this,
                "Error al generar reporte: " + e.getMessage(),
                title:"Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```