PLATAFORMA DE ORDENAMIENTO Y BÚSQUEDA EN CONSOLA

Solorzano Rodas, Julio Andres
UDV ESCTRUCTURA DE DATOS Ing. Brandon Chitay

1. Introducción

Este proyecto consiste en el desarrollo de una plataforma interactiva en consola que permite a los usuarios visualizar un menú con múltiples opciones. Entre sus funcionalidades, el sistema mostrará información sobre el desarrollador, permitirá cargar un conjunto de números desde un archivo CSV, y ejecutará algoritmos de ordenamiento y búsqueda según la elección del usuario.
El objetivo principal es ofrecer una herramienta educativa que permita explorar diferentes métodos de ordenamiento y búsqueda, proporcionando una experiencia práctica en estructuras de datos y algoritmos.

Relación con los algoritmos de ordenamiento y búsqueda en Java:
En el proyecto, uno de los principales objetivos es permitir que los usuarios **ejecuten algoritmos de ordenamiento** en un conjunto de números cargado desde un archivo CSV. Esto se relaciona con la implementación de los **algoritmos de ordenamiento en Java** como **Bubble Sort**, **Quick Sort**, **Insertion Sort**, y **Selection Sort**.

El proyecto permite que los usuarios **practiquen el flujo completo** desde cargar datos hasta ordenarlos y buscar elementos. La relación con los algoritmos de ordenamiento y búsqueda en Java se ve claramente cuando el sistema ordena un conjunto de números y luego permite al usuario buscar un número específico en el arreglo ordenado.

2. Explicación del Código (6-7 minutos)

Estructura del Menú:

Este menú permitirá al usuario navegar entre las distintas opciones del proyecto, como ordenar datos, buscar información o cargar un archivo CSV.

- **Creamos el método `mostrarMenu(Scanner scanner)`, que muestra un menú con las opciones del programa.**
  - Se ejecuta dentro de un bucle `do-while`, lo que permite que el menú se repita hasta que el usuario elija salir.
- **El usuario debe ingresar un número para seleccionar una opción.**
  - Usamos `scanner.nextInt()` para leer la entrada.
  - Luego, `scanner.nextLine()` se usa para limpiar el buffer y evitar errores al leer cadenas de texto.
- **Cada opción del menú está representada con un `case` dentro de un `switch`.**

  Cada opción representara acciones q realizan los sig.:

  - **Bubble Sort**
  - **Enhanced Bubble Sort**
  - **Quick Sort**
  - **Selection Sort**
  - **Merge Sort**

  Cargar CSV

- **Si el usuario elige la opción `9`, el programa finaliza con un mensaje de despedida.**
  - En cualquier otra opción, el usuario recibe un mensaje indicando que el algoritmo aún no ha sido implementado.

- **Algoritmos de Ordenamiento**:
    - Breve explicación de los algoritmos implementados:
        - **Bubble Sort**
        - **Enhanced Bubble Sort**
        - **Quick Sort**
        - **Selection Sort**
        - **Merge Sort**

Bubble Sort

- **Descripción**: El **Bubble Sort** es un algoritmo de ordenamiento simple y fácil de entender. Funciona comparando elementos adyacentes en el arreglo y, si están en el orden incorrecto, los intercambia. Este proceso se repite de manera iterativa hasta que el arreglo está completamente ordenado.

En cada pasada, el algoritmo "burbujeará" el mayor (o menor) elemento hacia el final del arreglo, y luego repetirá el proceso para el resto de los elementos.

2. Enhanced Bubble Sort (Bubble Sort Mejorado)

- **Descripción**: Es una versión optimizada del **Bubble Sort**. La mejora clave es que, después de cada pasada, si no se realizan intercambios en una iteración completa, el algoritmo se detiene antes de lo esperado, ya que eso indica que el arreglo ya está ordenado.
- **Cómo funciona**: Al igual que el **Bubble Sort** tradicional, compara elementos adyacentes y los intercambia si es necesario. Sin embargo, en cada pasada, verifica si ha hecho algún intercambio; si no lo ha hecho, termina el proceso.

3. Quick Sort

- **Descripción**: **Quick Sort** es un algoritmo de ordenamiento eficiente basado en el paradigma "divide y vencerás". Selecciona un **pivote** y divide el arreglo en dos subarreglos: uno con elementos menores al pivote y otro con elementos mayores. Luego, ordena recursivamente los subarreglos.
- **Cómo funciona**:
    1. Elige un pivote (puede ser el primer, último o medio elemento).

2. Reorganiza el arreglo de manera que los elementos menores al pivote queden a su izquierda y los mayores a su derecha.
3. Recursivamente ordena los subarreglos izquierdo y derecho.

4. Selection Sort

- **Descripción**: **Selection Sort** es un algoritmo de ordenamiento que funciona seleccionando el menor (o mayor) elemento en cada paso y colocándolo en la posición correcta del arreglo.
- **Cómo funciona**:
    1. Recorre el arreglo buscando el menor (o mayor) elemento.
    2. Intercambia ese elemento con el primer elemento no ordenado.
    3. Repite el proceso con el subarreglo restante.

5. Merge Sort

- **Descripción**: **Merge Sort** es un algoritmo de ordenamiento basado en el enfoque "divide y vencerás". Divide el arreglo en dos mitades, las ordena recursivamente y luego las combina en un arreglo ordenado.
- **Cómo funciona**:
    1. Divide el arreglo en dos mitades.
    2. Ordena cada mitad recursivamente.
    3. Combina las dos mitades ordenadas para formar el arreglo final ordenado.

- **Binary Search**:

La **búsqueda binaria** es un algoritmo eficiente para encontrar un elemento en un arreglo **ordenado**. El principio de la búsqueda binaria es dividir el conjunto de datos en mitades repetidamente, reduciendo así a la mitad la cantidad de elementos que deben ser examinados en cada paso, hasta que se encuentre el elemento o se determine que no está presente, funciona solo en listas ordenadas. Busca el valor en la lista dividiendo el rango de búsqueda a la mitad repetidamente hasta encontrar el valor o determinar que no está presente.

- **Cargar Datos desde un Archivo CSV**:

Cargar datos desde un archivo CSV (Comma-Separated Values, o valores separados por comas) y convertirlos en una lista de números es un proceso común en muchos programas. Para hacerlo en Java, se pueden seguir una serie de pasos utilizando clases estándar como FileReader, BufferedReader, y métodos como split() para separar los datos por comas.

3. Recomendaciones y Mejoras (1-2 minutos)

- **Mejoras y Expansión**:

Optimización de algoritmos de ordenamiento:

- **Mejorar la eficiencia de Bubble Sort**: Aunque el **Enhanced Bubble Sort** es más eficiente que el **Bubble Sort** básico, aún es ineficiente para listas grandes. Para listas más grandes, se podría cambiar a algoritmos más eficientes como **Quick Sort**, **Merge Sort** o **Heap Sort**.

Agregar Nuevos Algoritmos

- **Algoritmos de ordenamiento adicionales**:
    - **Heap Sort**: Este algoritmo es útil si necesitamos un algoritmo de ordenamiento con un consumo de memoria más eficiente que **Merge Sort**.

- **Resumen Final**:

Este proyecto ha permitido el desarrollo de una plataforma interactiva en consola que ofrece a los usuarios una experiencia educativa y práctica sobre los conceptos fundamentales de los **algoritmos de ordenamiento** y **búsqueda**. A través de un menú interactivo, los usuarios pueden:

- Visualizar información sobre el desarrollador.
- Cargar un conjunto de números desde un archivo CSV.
- Ejecutar varios algoritmos de **ordenamiento** (como **Bubble Sort**, **Quick Sort**, **Merge Sort**, **Selection Sort**, y **Enhanced Bubble Sort**) y **búsqueda** (como **Búsqueda Binaria**) según la selección del usuario.

El proyecto cumple su objetivo principal de proporcionar una herramienta educativa que permita explorar y experimentar con diferentes métodos de **ordenamiento** y **búsqueda**. Además, ofrece una **interfaz de consola interactiva**, lo que permite a los usuarios ver de manera práctica cómo funcionan estos algoritmos en tiempo real.

Durante el desarrollo del proyecto, se abordaron diversas tareas técnicas, como la **lectura y procesamiento de archivos CSV**, la **implementación de algoritmos eficientes** de ordenamiento y búsqueda, y la creación de una estructura interactiva para que los usuarios pudieran interactuar fácilmente con la plataforma. Además, se implementaron mejoras para optimizar el rendimiento, asegurando que el sistema pudiera manejar de manera eficiente tanto archivos pequeños como grandes.

Gracias por el apoyo recibido

Gracias por escucharme, al profesor y compañeros.

Enlaces

Youtube

https://youtu.be/x9RYrKhT1zo

github

https://github.com/julio462/PRIMER-PARCIAL-ED1.git