Trabajo Práctico Final

Santino Fuentes FAI-3666

3 de julio de 2023

Desarrollo de Algoritmos

Índice

1. Estructuras Utilizadas

- **Alumno:** Clase que representa a un alumno con atributos como apellido, nombre, legajo, grado y promedio.
- **int[] legajosRepitentes:** Arreglo para almacenar los legajos de alumnos repitentes, es decir, no van a pasar de grado.
- **double[] promedios:** Arreglo para almacenar los promedios de cada grado.
- **Alumno[][] infoAlumnos:** Matriz bidimensional para almacenar la información de los alumnos organizados por grados.
- **Alumno[] egresados:** Arreglo para almacenar los alumnos egresados de cada promoción. Cada año se actualizan.

2. Módulos y su Funcionalidad

- **linea():** Imprime una línea divisoria por pantalla.
- **mostrarMenu():** Muestra el menú principal del programa y solicita al usuario que ingrese una opción válida.
- **cargarAlumnos():** Lee la información de los alumnos desde un archivo y carga la matriz *infoAlumnos*.
- **asignarAlumno():** Asigna un objeto *Alumno* a la matriz *alumnos* en el grado correspondiente.
- **cargarAlumnosDes():** Carga los legajos de alumnos desaprobados desde un archivo al arreglo *desaprobados*.
- **verificarMatrizCargada():** Verifica si la matriz *unaMatriz* tiene algún valor cargado.
- **intercambiar():** Intercambia dos posiciones en un arreglo de enteros.
- **ordenarRepitentes():** Ordena un arreglo de legajos de manera ascendente utilizando el algoritmo de ordenamiento de burbuja mejorado.
- **ordenarPorNombre():** Ordena un arreglo de alumnos de un grado en orden ascendente según el orden lexicográfico de la concatenación *nombre+apellido*.
- **buscarRepitente():** Realiza una búsqueda binaria en el arreglo de repitentes para determinar si un alumno es repitente.
- **pasarDeGrado():** Actualiza la matriz de alumnos para pasarlos de grado y los asigna a la matriz de egresados si corresponde.
- **contarAlumnosGrado():** Cuenta la cantidad de alumnos en un grado, es un módulo auxiliar para calcular el promedio del grado.
- **calcularPromedioGrado():** Calcula el promedio del grado, es un módulo auxiliar para calcular los promedios generales.
- **calcularPromedios():** Calcula los promedios de cada grado y los almacena en el arreglo *promedios*.
- **mostrarPromedios():** Muestra los promedios calculados por grado por pantalla.

- **Alumno.promover():** Propio de la clase *Alumno*, modifica el atributo *grado* en función del valor actual. Si el grado es menor o igual a 6, entonces el valor del grado se incrementa en 1. Si el grado es igual a 7, entonces no se incrementa pues se trata de un egresado, entonces su valor cambia a -1.
- **Alumno.toString():** Propio de la clase *Alumno*, su función es la de concatenar el los atributos de la clase en una sola cadena.

3. Pasar de grado

El proceso de pasar de grado se implementó en un método *pasarDeGrado()* el cual tiene por parámetros formales una matriz de alumnos, la cual almacena la información sobre los alumnos, un arreglo de enteros, el cual contiene los legajos de todos los alumnos repitentes, y otra matriz de alumnos, la cual almacena los alumnos egresados. Cuando se invoca el método desde el algoritmo principal, se pasan como parámetros actuales las estructuras mencionadas (matrices y arreglo) y éste crea una nueva estructura (matriz de la misma dimensión que la pasada por parámetro, 7 x 4) vacía, la cual va a utilizar luego. Se comienza iterando por fila y columna en la matriz de entrada, obteniendo así un alumno, el cual se copia a un nuevo alumno tambien creado en éste método, luego se invoca otro método *buscarRepitente()* con parámetro actual el alumno de la actual iteración, en el cual se analiza la existencia de un legajo igual al del alumno, de ser encontrado, se aplica el método *promover()* de la clase Alumno sobre el alumno actual, por lo que su atributo *grado* se incrementa. ¿Qué sucede si el alumno está en séptimo grado y se promueve? El método propio *promover()* verifica cuando esta condición sucede y en lugar de incrementarse, el grado pasa a ser -1. A continuación, se verifica si luego de aplicar el método anterior, al alumno le corresponde egresar, de ser así, el alumno se asigna a una nueva posición en la matriz *egresados*. De no corresponderle al alumno egresar, se asigna a una nueva posición en la matriz *actualizados*. Este proceso se repite en cada fila de la matriz, siempre que no haya elementos nulos.
Al finalizar el proceso, la matriz *actualizados* se retorna, para que en *main()* sea asignada a la matriz original.

4. Manejo de los Valores Nulos

Los valores nulos, no son ningún problema utilizando el método *asignarAlumno()*, en el, lo que se hace es ir *encolando* los elementos en la matriz, primeramente obteniendo el grado del alumno con el método *getGrado()*, para así saber sobre que fila trabajar. Además, al retornar la referencia de la nueva matriz en el método *asignarAlumno()*, en *main()*, se reasigna la matriz retornada a la matriz anterior.
Quiero dejar claro que en ningún momento algún método "elimina" elementos de las matrices, sino que siempre se itera por todos los alumnos previa invocación del método *promover()*, es por ello que nunca quedan "huecos" en las estructuras.

5. Promoción

Comparación de QuickSort y BubbleSort Mejorado El método *BubbleSort Mejorado* compara cada elemento de un arreglo con su sucesor y los intercambia entre si de ser necesario, cada pasada va a "hundir" el elemento más grande (o más pequeño, si así se desea) hasta la última posición menos la anterior. En el caso de haber hecho una pasada y la bandera no ha cambiado su valor, ya no realiza más pasadas, es lo que lo hace mejor que el *BubbleSort* tradicional.
El método *QuickSort* utiliza la táctica "divide y vencerás", teniendo en cada iteración un arreglo más pequeño y ubicado correctamente en un valor llamado "pivote".

Complejidad Temporal de los Algoritmos			
Algoritmo	**Promedio**	**Mejor Caso**	**Peor Caso**
BubbleSort	$O(n^2)$	$(O(N))$	$O(n^2)$
QuickSort	$O(n * log(n))$	$O(n * log(n))$	$O(n^2)$

Observando los órdenes, podemos notar que el método *QuickSort* es más rápido en promedio que el *BubbleSort*, porque el método burbuja.

Pruebas Empíricas

Tiempo en segundos		
Tamaño	**BubbleSort**	**QuickSort**
0	0,000002	0,000001
10	0,000002	0,000002
20	0,000004	0,000004
30	0,000008	0,000005
40	0,000011	0,000006
50	0,000026	0,000007
60	0,000023	0,000012
70	0,000046	0,000009
80	0,000037	0,000014

Comparativas entre Algoritmos de Ordenamiento