Práctica 4

Parcial 2. Temas 5 y 6

Objetivo

Implementar en **parejas** el código **original** para resolver la tarea planteada, demostrando que se alcanza el siguiente resultado de aprendizaje de la asignatura:

- **RAE2.** Entender estrategias algorítmicas clásicas (divide y vencerás, avance rápido y vuelta atrás), y ser capaz de utilizarlas a la hora de implementar soluciones a problemas concretos.
- **RAE5.** Comprender las principales estructuras de datos relacionales (grafos), su utilidad, su funcionamiento y su implementación. Ser capaz de emplearlas para la resolución de problemas concretos y como mecanismo para plantear soluciones más óptimas.

Descripción

Resolver los ejercicios propuestos, implementando para cada uno de ellos una función, y utilizando para ellos las clases proporcionadas a través del Campus Virtual, que implementan un grafo dirigido:

- Grafo.java
- Lista.java
- Par.java

Deberá crearse una clase llamada “Practica4.java”, que incluirá una función por cada ejercicio, además de la función main con las pruebas que los estudiantes estimen oportunas. No debe declararse esta clase dentro de ningún **paquete** de java.

Si se considera necesario, se pueden crear funciones auxiliares como privadas, que posteriormente se justificarán en la documentación.

Ejercicio 1. Codificar una función que calcule el grado del grafo que se le pase como entrada:

```
public int gradoGrafo (Grafo<Clave,InfoV,Coste> grafo)
```

Ejercicio 2. Escribir una función que reciba como entrada un grafo y un vértice v, y que imprima por pantalla todos los vértices alcanzables desde v aplicando un recorrido en anchura. Para simular el contenedor cola necesario para este tipo de recorrido, se debe utilizar el TAD Lista implementado mediante la clase Lista.java, pero utilizándolo

con las restricciones asociadas a una cola, con accesos únicamente a través de sus extremos.

```
public void recorridoAnchura (Grafo<Clave,InfoV,Coste> grafo, Clave v) {
```

Ejercicio 3. Implementar una función que reciba como entrada un grafo, y devuelva como resultado el número de componentes conexas que lo componen, basándose para ello en la aplicación de un recorrido en profundidad.

```
public int componentesConexas (Grafo<Clave,InfoV,Coste> grafo) {
```

Ejercicio 4. Un camión necesita llegar desde una ciudad A hasta otra B siguiendo una ruta predefinida. Teniendo en cuenta que el vehículo tiene una autonomía máxima de X kilómetros, el objetivo es alcanzar el punto B realizando el menor número posible de paradas/repostajes intermedios.

Se pide aplicar la estrategia algorítmica de **avance rápido** para codificar la función "calcularViaje", que escriba por pantalla en orden de parada los nombres de las ciudades donde habrá que detenerse para alcanzar el destino.

```
public void calcularViaje (String[] ciudades, int[] distancias,
int autonomia) {
```

Los parámetros de entrada representan lo siguiente:

- **`String[] ciudades:`** Un array de tamaño N con los nombres de las N ciudades de la ruta. En este array no se incluye la ciudad de origen A, sino todas las posibles paradas. Así, la primera ciudad (índice 0) será siempre la ciudad más cercana a A, y la última (índice N-1) será la ciudad de destino B.
  `Ejemplo:` ciudades = {"Toledo", "Ciudad Real", "Jaén", "Alcalá la Real", "Granada"}

- **`int [] distancias:`** será el array de tamaño N que almacena la distancia en kilómetros que hay desde una ciudad a la siguiente. Por tanto, el número en la primera posición representa los kilómetros desde la ciudad A hasta el primer destino. La cantidad en la segunda posición indica la distancia desde la primera potencial parada hasta la segunda, etc. En el ejemplo, 75 indica la distancia desde el origen hasta Toledo, y el 120 la distancia desde Toledo hasta Ciudad Real.
  `Ejemplo:` ciudades ={75, 120, 173, 70, 55}

- **`int autonomía:`** Representa los kilómetros máximos que el camión puede recorrer sin repostar.
  `Ejemplo:` 200

No se permite la entrega de código ni documentación que no hayan sido realizados totalmente y de manera original por los estudiantes. No se permite el uso de sistemas

de inteligencia artificial o "copilotos". Tras la entrega, la práctica podrá incluir un examen oral a cualquiera de los autores.

Adicionalmente, para aprobar la práctica es condición necesaria:

- Respetar la interfaz de clases y funciones/métodos especificada.
- Que el código compile.
- No salir de los bucles utilizando sentencias que modifiquen el flujo lógico del programa: break, continue, return, etc.
- Adjuntar los ficheros pedidos siguiendo las normas de entrega.

Documentación

El código debe estar convenientemente documentado (sin excesos):

- Cada **fichero de clase** desarrollado por los estudiantes contendrá una cabecera identificando a los autores (por orden alfabético), fecha y versión del código.
  Ejemplo:

```
/**
 * @author1 Apellido1 Apellido2, Nombre
 * @expediente1 12345678
 * @author2 Apellido3 Apellido4, Nombre
 * @expediente2 87654321
 * @date 2024-05-10
 * @version 0.8
 */
```

- Las funciones y métodos tendrán una breve documentación en Javadoc.
  Ejemplo:

```
/**
 * Función que revuelve el dato máximo de un array de números enteros
 * @param lista: array de números enteros
 */
      public int maximo(int[] lista){
      ...
```

Se adjuntará un documento en formato **PDF** que deberá nombrarse como se indica más adelante. Esta documentación debe incluir:

- Portada
- Índice
- Código en formato texto convenientemente formateado y tabulado de las funciones implementadas para la(s) clase(s) entregadas. No es necesario incluir la función main. Cada función debe documentarse en un apartado diferente, redactando antes muy brevemente, en menos de media página, la idea o estrategia seguida. Las funciones adicionales deben justificarse aquí.
- Dibujo o gráfico, en formato digital (no manuscrito ni fotografía adjunta) de las estructuras utilizados para las pruebas, junto con el código utilizado para su creación.

- Tabla esquemática de las pruebas realizadas: qué se quiere probar, entrada suministrada, resultado esperado, resultado obtenidos.
- Bibliografía

Normas de entrega

En la entrega deben adjuntarse **2 ficheros por separado**, no debe enviarse ningún archivo comprimido:

1. Practica4.java, que incluirá las funciones:
   - gradoGrafo
   - recorridoAnchura
   - componentesConexas
   - calcularViaje
   - main

2. Apellido1_Apellido2.Apellido3_Apellido4.**P4**-M2X-GYY.**PDF**, donde:
   - Los estudiantes se ordenarán alfabéticamente.
   - Apellido1_Apellido2 indican los apellidos del primer estudiante.
   - Apellido3_Apellido4 indican los apellidos del segundo estudiante.
   - **X** se sustituirá para indicar el número de grupo de la asignatura (M21, M22, M23, etc.) e **YY** será para el número de la pareja de trabajo.

   - **Ejemplo:**
     "Castro_Suárez.Serrano_Martínez.**P4**-M23-G05.PDF"

La entrega debe hacerse antes de las **23:55h** del día marcado como fecha límite.