Pregunta 1

0 pts

Yo, en mi calidad de estudiante de la ESPOL, declaro que he sido informado y conozco las normas que rigen a la ESPOL, en particular el Código de Ética y el Reglamento de Disciplina.

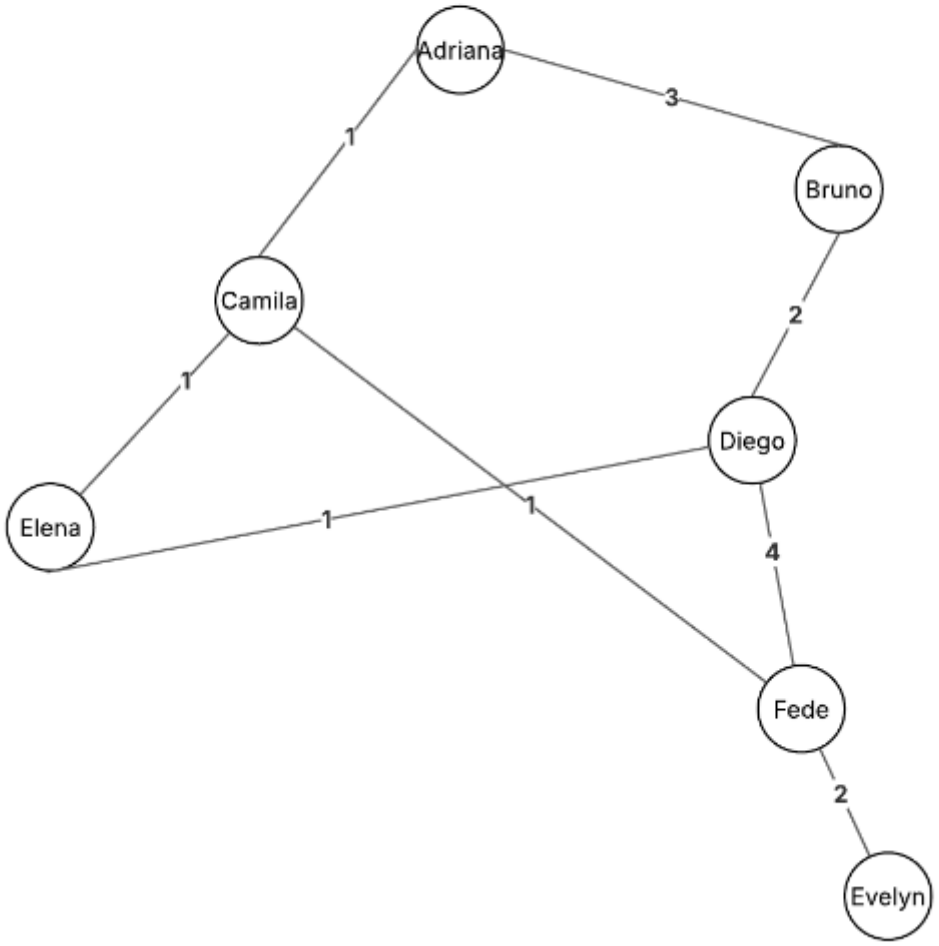
Acepto con pleno conocimiento este compromiso de honor por el cual reconozco y estoy consciente de que la presente evaluación está diseñada para ser resuelta de forma individual; que sólo puedo comunicarme con la persona responsable de la recepción de la evaluación; que no haré consultas indebidas o no autorizadas por el evaluador; ni usaré dispositivos electrónicos.

Acepto el presente compromiso, como constancia de haber leído y aceptar la declaración anterior y me comprometo a seguir fielmente las instrucciones que se indican para la realización de la presente evaluación.

- ○ Falso
- ○ Verdadero

Pregunta 2 — 35 pts

En las colaboraciones entre creadores de contenido de redes sociales como TikTok, YouTube. En que cada creador es un vértice. Dos creadores están conectados por un arco si han publicado al menos una colaboración como se muestra en la figura. El factor de peso del arco es el número de colaboraciones entre ellos en los últimos 12 meses. La metadata del arco (tipo E) guarda, el ID del reto o una URL del video más representativo de esa colaboración.



Implementar en la clase GraphAL<V> los siguientes métodos:

```
public List<V> shortestPathUnweighted(V origen, V destino): Devolver la ruta con menos saltos entre origen y destino, en una lista de origen a destino (in cluidos). Si no hay camino, devolver la lista vacía. (20 puntos)
```

```
public List<V> dfsOrder(V origen): Devolver la lista de V en el orden en que son visitados. (15 puntos)
```

Requerimientos de implementación:

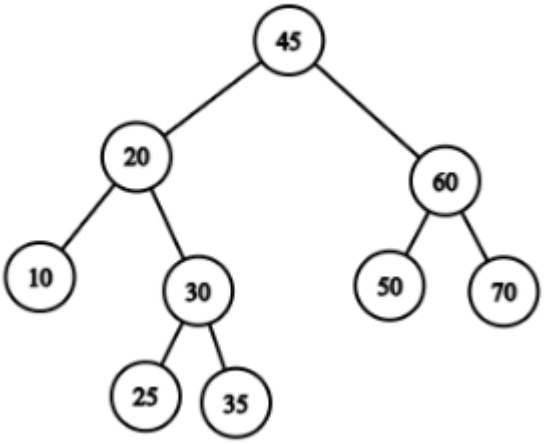
- Considera el grafo no dirigido.
- Evita vértices repetidos.
- Los factores de pesos no se usan en el camino con menos saltos.

Pregunta 3 — 35 pts

La empresa de mensajería MensajesEC desea organizar sus rutas de entrega basándose en la prioridad de entrega de cada paquete, para esto, cada paquete tiene un código numérico entre 1 y 100 que indica su prioridad (menor número = mayor prioridad), es decir la empresa debe entregar primero los de mayor prioridad.

Para resolver el problema, deberá construir un Árbol Binario de Búsqueda **(BST)** donde cada nodo representa un paquete y su valor es el código de prioridad. Analice la construcción del siguiente ejemplo para que así pueda desarrollar los requerimientos solicitados:

- Use estos datos de prueba: 45, 20, 60, 10, 30, 50, 70, 25, 35
- Este es el árbol binario de búsqueda **(BST)** que se espera:



A continuación, los requerimientos a desarrollar, junto con un ejemplo:

1. Desarrolle un método que permita insertar nuevos paquetes **(15 puntos)**
   - `public void insertar (int codigo):`

     Inserte un nuevo paquete en el BST respetando las reglas del árbol binario de búsqueda. Recibe el código del paquete a insertar en el árbol. Para los valores duplicados solo ignore su ingreso.
2. Desarrolle un método que permita conocer en qué nivel hay más paquetes y la cantidad. **(20 puntos)**
   - `public String nivelMasCongestionado():`

     Calcula el nivel que contiene la mayor cantidad de paquetes en el BST (nivel más congestionado) y devuelve un mensaje que indica, que nivel es el más congestionado y cuantos paquetes tiene ese nivel. Utilice recorrido de anchura **(BFS) (nivel por nivel)**. No recibe argumentos. Devuelve un mensaje indicando el nivel más congestionado e indica la cantidad de paquetes en dicho nivel. En caso de que dos niveles tengan la misma cantidad de paquetes, devuelva el nivel más cercano a la raíz y que sea el último vértice agregado.

     - Use el BST planteado, se visitan los nodos nivel por nivel, de izquierda a derecha
     - 45 → 20 → 60 → 10 → 30 → 50 → 70 → 25 → 35

- Ejemplo con el BST planteado, el resultado del nivel más congestionado es el Nivel 2, y cuenta con 4 paquetes:
  - **Nivel 0:** Vértice [45] → 1 paquete
  - **Nivel 1:** Vértices [20, 60] → 2 paquetes
  - **Nivel 2:** Vértices [10, 30, 50, 70] → 4 paquetes
  - **Nivel 3:** Vértices [25, 35] → 2 paquetes

  Nota: Considere que el nivel de la raíz es el nivel 0.

Pregunta 4 — 30 pts

En una plataforma de streaming se quiere identificar las películas más rentables en términos de ingresos. Una película se considera valiosa cuando su precio de alquiler es alto y ha sido vista muchas veces.

Debe implementar un método, el cual recibe un primer mapa que asocia códigos de las películas (String) con valores enteros como conteos u ocurrencias de veces que han visto la película. Un segundo mapa que asocia códigos de las películas (String) con valores numéricos (Precio de alquiler) y un entero n.

Debe retornar una List<String> con los códigos de los n elementos más "rentables", ordenados de mayor a menor según el puntaje calculado como producto de los valores de ambos mapas para cada código. En caso de empate, se evaluará que el orden se resuelva de forma determinista, por ejemplo usando el código en orden ascendente.

```
public static List<String> obtenerPeliculasMasRentables( Map<String, Integer> ma
```

Código	Vistas
M001	120
M002	80
M003	150
M004	60
M005	200

Código	Precio
M001	2.50
M002	4.00
M003	1.75
M004	4.00
M005	1.15

Ejemplo: El cálculo de ingresos.

- M001 → 120 × 2.50 = 300.0
- M002 → 80 × 4.00 = 320.0
- M003 → 150 × 1.75 = 262.5
- M004 → 60 × 4.00 = 240.0
- M005 → 200 × 1.15 = 230.0

Ordenados de mayor a menor:

M002 (320.0), M001 (300.0), M003 (262.5), M004 (240.0), M005 (230.0)

Para **n = 3**, el programa retorna: Top 3 películas más rentables: [M002, M001, M003]

El esquema general de la clase película y consideraciones se detallan a continuación:

```java
public class Pelicula {
    // Atributos
    private String codigo;   // Código único de la película
    private String titulo;   // Título de la película
    private double precio;   // Precio de alquiler
    private int vistas;      // Número de visualizaciones

    // Constructor
    public Pelicula(String codigo, String titulo,
                    double precio, int vistas) { ... }

    // Getters
    public String getCodigo() { ... }   // Retorna el código de la película
    public String getTitulo() { ... }   // Retorna el título de la película
    public double getPrecio() { ... }   // Retorna el precio de alquiler
    public int getVistas() { ... }      // Retorna la cantidad de vistas

    // Setters
    public void setPrecio(double precio) { ... }
        // No retorna nada (void)
    public void setVistas(int vistas) { ... }
        // No retorna nada (void)

    // Método para calcular ingresos
    public double calcularIngresos() { ... }
        // Retorna los ingresos (precio * vistas)
}
```

Para la implementación se espera que el estudiante seleccione estructuras de datos que permitan manejar eficientemente la unión de claves, el almacenamiento temporal de pares código–valor y el ordenamiento según los criterios establecidos. Asimismo, se debe validar el manejo de casos extremos o inválidos, tales como valores nulos en los mapas o cuando el parámetro **n** sea menor o igual a cero.