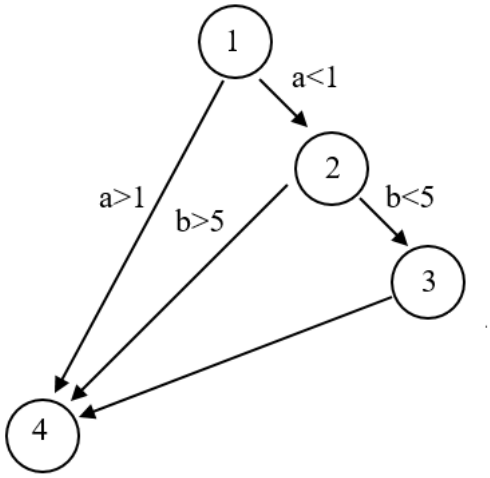
2021 Examen Mayo Resuelto

Fundamentos de Diseño de Software (Universidad Rey Juan Carlos)

Programar un algoritmo que, utilizando la técnica de Divide y Vencerás, decida si dos arrays son o no iguales (3 puntos)

```
public boolean iguales(int[]uno, int[] otro){
   return iguales (uno, otro, 0, otro.length-1);
 }

 public boolean iguales (int [] uno, int [] otro, int inicio, int fin){

    if (uno.length!=otro.length) return false;

    if (inicio == fin) {
          if (uno[inicio] == otro[inicio]) {
             return true;
          } else {
             return false;
          }

       } else {

          int mitad = (inicio + fin) / 2;
          return (iguales(uno, otro, inicio, mitad) && iguales (uno, otro, mitad + 1, fin));

       }

   }
```

Observando el siguiente diagrama de flujo, ¿Cuánto vale la complejidad ciclomática del código asociado?



Seleccione una:

- a. 3
- b. 4
- c.Sin el código no se puede calcular.
- d. 2

Regiones = 3 (no olvidar la externa).

Aristas - Nodos + 2 = 5 – 4 + 2= 3

Condiciones + 1 = 2 + 1 = 3

¿Cuál es la T(n) del siguiente fragmento de código? (2 puntos)
NOTA:- ESTÁ PERMITIDO USAR PAPEL Y LÁPIZ PARA HACER CÁLCULOS

```
FOR i = 1 TO n-1 DO
  FOR j = 1 TO n DO
     IF j == i THEN
        x=j
```

Seleccione una:

- a. 5n^2 + 6n + 3 (5 n cuadrado + 6n + 3)
- b. 6n^2 - 3 (6 n cuadrado menos 3)
- c. Prefiero no contestar
- d. 5n^2 + n - 3 (5 n cuadrado + n - 3)
- e. 6n^2 + 6n + 3 (6 n cuadrado + 6n + 3)

$T(n) = T_{for1}(n) = (n-1) * (3 + T_{for2}(n)) + 3$

$T_{for2}(n) = n * (3 + T_{if}(n)) + 3$

$T_{if}(n) = 1 + 1 = 2$

$T_{for2}(n) = n * (3 + 2) + 3 = 5n + 3$

$T(n) = T_{for1}(n) = (n-1) * (3 + 5n + 3) + 3 = (n-1) * (5n + 6) + 3 = 5n^2 - 5n + 6n - 3 =$

$5n^2 + n - 3$

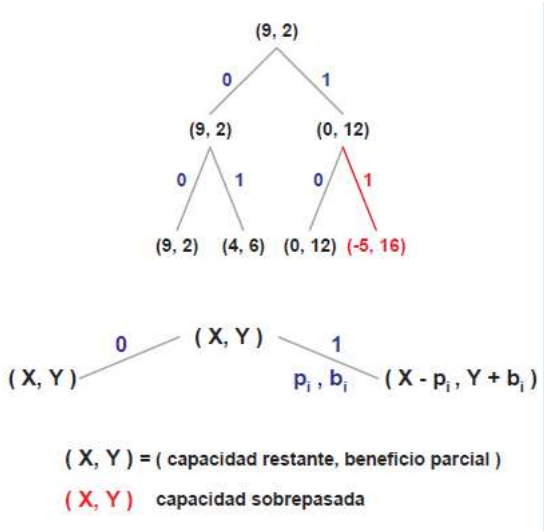
(3 puntos)

El problema de la mochila 0/1 consiste en obtener el máximo beneficio posible introduciendo N objetos en una mochila con capacidad C. Cada objeto tiene un peso y un beneficio asociado. Los objetos no se pueden trocear.

Contestar a las siguientes preguntas, suponiendo:

que el problema se va a resolver mediante la técnica de "backtracking".

- que el resultado se va a ofrecer en un "array" de N componentes en el que el componente i-ésimo contendrá un 1 si el objeto i-ésimo se ha metido en la mochila, y un 0 en caso contrario.
- que se ofrece a continuación un trozo del árbol de exploración utilizado para un ejemplo dado.



¿Qué esquema de la técnica usarías para resolver el problema? Justifica la respuesta.

Mejor Solución. Es un problema de optimización.

¿Cuántos niveles tendrá el árbol de exploración?

N+1

¿Cuántos candidatos tendrá cada nodo? Justifica la respuesta.

Dos. Se corresponden con el hecho de incluir o no incluir el elemento en la mochila.

¿En qué consistirá la función de factibilidad? (basta explicarlo en lenguaje natural)

El problema será factible si el próximo candidato no desborda la mochila.

¿En qué consistirá la función solución? (basta explicarlo en lenguaje natural)

Se obtendrá una solución si se ha llegado a un nodo hoja (evidentemente, cumpliendo factibilidad). Por otra parte, se guardará en todo momento la mejor solución, debido a la naturaleza del problema.

¿El problema se puede resolver con una estrategia voraz? Justifica la respuesta.

El problema acepta una estrategia voraz, y con una función de selección como "beneficio decreciente" obtiene buenos resultados. Sin embargo, no alcanza el óptimo, es decir, existe un número no despreciable de casos en los que la solución ofrecida no es la mejor posible.

(1 punto)

En el algoritmo de Dijkstra resuelto a través de un algoritmo voraz, la función de factibilidad...

Seleccione una:

- a. selecciona el nodo pendiente que tenga la menor distancia desde el origen.
- b. Prefiero no contestar
- c. impide que el algoritmo continúe si se forma un ciclo.
- d. tiene una complejidad lineal O(n).
- e. siempre devuelve "true". ●

El algoritmo de Dijkstra siempre es factible. Errores frecuentes: confundir la función de factibilidad con la de selección; confundir el algoritmo de Dijkstra con el de Prim o Kruskal, en los que se pueden formar ciclos.

3 puntos

Reescribir el código facilitado a continuación para que cumpla el principio de inversión de dependencias.

Escribir después una clase adicional que permita que la Central de Datos, en lugar de la hora, pueda escribir la fecha (también como String, por simplificar).

Por último, escribir una clase Test que cree una Central de Datos que en primer lugar escriba la hora y que después escriba la fecha.

```
public class CentralDeDatos {
  public void mostrarInformacion() {
    Reloj reloj = new Reloj();
    String hora =reloj.mostrarHoraActual();
    System.out.println (hora);
    }
}
public class Reloj {
   private String hora;
   public String mostrarHoraActual() {
     return ("Hora exacta:" + hora); }
}
```

```
public class CentralDeDatos {

  Aparato a;

  public CentralDeDatos (Aparato a) {

    this.a=a;

  }

  public String mostrarInformacion() {

    return (a.mostrar());

    }

}

public class Reloj implements Aparato {

   private String hora;

   public String mostrar() {

     return ("Hora exacta:" + hora); }

}
```

```
public class Calendario implements Aparato {
    private String fecha;
    public String mostrar() {
       return (“Fecha exacta:” + fecha); }
 }
public interface aparato {
   String mostrar();
}
public class Test{

public static void main (String [] args){
 Reloj r = new Reloj();
 Calendario c = new Calendario();
 CentralDeDatos cdd = new CentralDeDatos (r);
 System.out.println (cdd.mostrarInformacion());
 cdd = new CentralDeDatos (c);
 System.out.println (cdd.mostrarInformacion());
  }
}
```

3 puntos

Dada la siguiente clase:

```
public class Misterio{
     Impresora impresora;
     public Misterio (Impresora i) {
      impresora = i;
     }
   public imprimeNormal (String texto) {
       impresora.setColor(true);
       impresora.setHoja("A4");
       impresora.setTipoDocumento("PDF");
       impresora.setTexto(texto);
       impresora.imprimirDocumento();
   }
   public imprimeBorrador (String texto) {
       impresora.setColor(false);
       impresora.setHoja("A4");
       impresora.setTipoDocumento("PDF");
       impresora.setTexto(texto);
       impresora.setBorrador (true);
       impresora.imprimirDocumento();
   }
}
 public class Cliente {
   public static void main(String[] args) {
       Impresora i = new Impresora ();
       Misterio misterio= new Misterio(i);
       misterio.imprimeNormal("hola");
       misterio.imprimeBorrador("adios");
       i.setColor(true);
       i.setHoja("A3");
       i.setTipoDocumento("TXT");
       i.imprimirDocumento();
}
```

Contestar a la siguiente pregunta: ¿Qué patrón de diseño se está utilizando?

A continuación, realizar todos los cambios necesarios en el código para que la clase Misterio se ajuste al patrón Singleton (escribir en la caja de texto el código completo modificado).

NOTA:- La clase Impresora se presupone correctamente programada, compilada y accesible en todo momento.

Respuesta: Patrón Facade

```
public class Misterio{
      Impresora impresora;
/*    public Misterio (Impresora i) {
       impresora = i;
     }  */
   private static final Singleton uniqueInstance;
   private Misterio () { }
    public static Misterio getInstance(){
       if (uniqueInstance == null)
              uniqueInstance = new Misterio();
      return uniqueInstance;
     }
    public setImpresora (Impresora i){
      impresora = i;
    }
     public imprimeNormal (String texto) {
       impresora.setColor(true);
       impresora.setHoja("A4");
       impresora.setTipoDocumento("PDF");
       impresora.setTexto(texto);
       impresora.imprimirDocumento();
    }
```

```
    public imprimeBorrador (String texto) {
        impresora.setColor(false);
        impresora.setHoja("A4");
        impresora.setTipoDocumento("PDF");
        impresora.setTexto(texto);
        impresora.setBorrador (true);
        impresora.imprimirDocumento();
    }
}
public class Cliente {
    public static void main(String[] args) {
        Impresora i = new Impresora ();
        //Misterio Misterio= new Misterio (i);
        misterio = Misterio.getInstance();
        misterio.setImpresora (i);
        misterio.imprimeNormal("hola");
        misterio.imprimeBorrador("adios");
        i.setColor(true);
        i.setHoja("A3");
        i.setTipoDocumento("TXT");
        i.imprimirDocumento();
}
```

4 puntos

Aplicar el patrón Decorator al siguiente escenario: queremos poder incorporar equipamientos extra a los coches, y poder comprobar en todo momento el precio actualizado de los coches.

En el ejercicio bastará con considerar el extra "PotenciaAdicional", que tendrá un precio de 500 Euros y el extra "CamaraTrasera", que tendrá un precio de 1000 Euros. Se programará también una clase Test que en su método "main" incorpore a un coche una cámara trasera y le proporcione potencia adicional dos veces. El programa ofrecerá el precio del coche antes y después de incorporar cada extra.

```
public  abstract class Vehiculo{
    protected double precio;

    public  abstract double getPrecio ();
    public  void setPrecio (int precio) {this.precio=precio;}
    }
public class Coche extends Vehiculo{
    public double getPrecio () {return precio;}
}
public abstract class Extra extends Vehiculo{
    public  abstract double getPrecio ();}

public class CamaraTrasera extends Extra{
   private final double PRECIO_EXTRA = 1000;
   private Vehiculo v;

    public CamaraTrasera (Vehiculo v){
        this.v = v;
        this.precio =v.precio+PRECIO_EXTRA;
     }

    public  double getPrecio() {
         return precio;
    }
 }
```

```
public class PotenciaAdicional extends Extra{
   private final double PRECIO_EXTRA = 500;
   private Vehiculo v;

    public PotenciaAdicional (Vehiculo v){
        this.v = v;
        this.precio =v.precio+PRECIO_EXTRA;
     }

    public  double getPrecio() {
         return precio;
    }
 }

public class Test{

   public static void main (String [] args){
       Coche c = new Coche();
       c.setPrecio(10000);
       System.out.println (c.getPrecio());
       CamaraTrasera ct = new CamaraTrasera (c);
       System.out.println (ct.getPrecio());
        PotenciaAdicional pa1 = new PotenciaAdicional (ct);
        System.out.println (pa1.getPrecio());
        PotenciaAdicional pa2 = new PotenciaAdicional (pa1);
       System.out.println (pa2.getPrecio());
    }
}
```