UD04.EXAME Práctico

DAM1-Programación 2024-25
12/02/2025

- Puedes utilizar apuntes y materiales que consideres pero deberás realizar los programas individualmente. En caso contrario se retirará el examen.
- Realiza programas bien estructurados, legibles, con comentarios, líneas en blanco, identificadores adecuados, etc.
- Cuida la interacción con el usuario, presentando la información de forma clara y ordenada.
- Utiliza los casos de prueba de ejemplo para probar tu programa y definir la salida por pantalla.

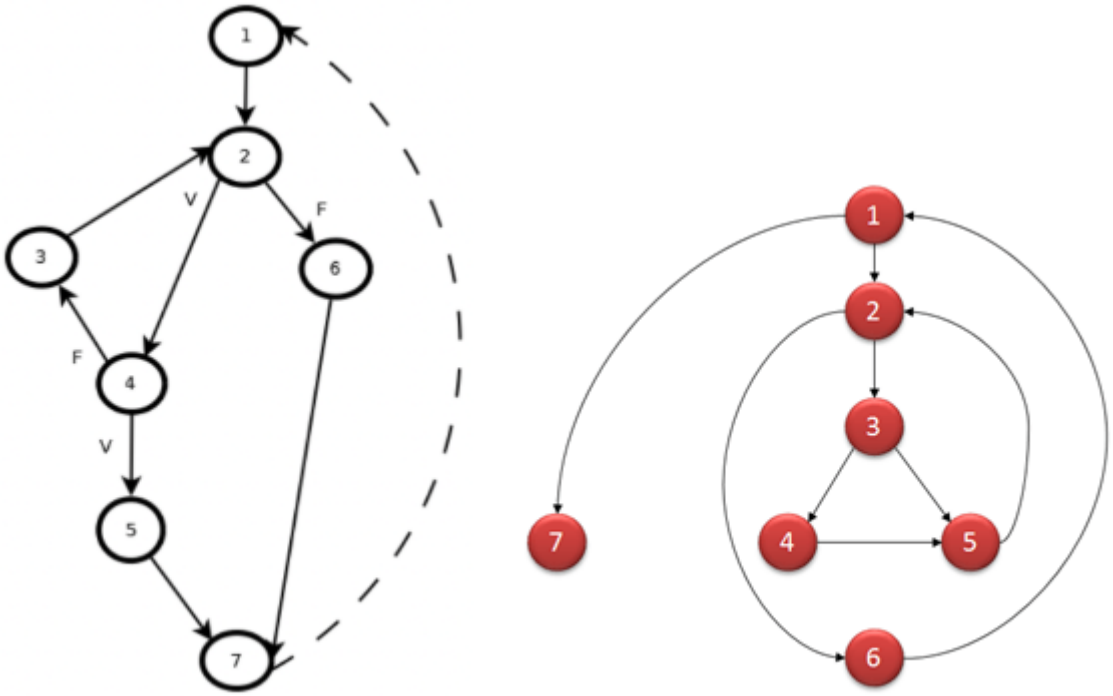
- Crea un **paquete** de nombre **ud4.xxxexamen** donde agrupar los ficheros de código fuente, donde **xxx** son las iniciales de tu nombre y apellidos. Si es necesario crea un nuevo proyecto/carpeta Java.
- Crea dentro del paquete ficheros **.java** por cada ejercicio con el nombre apropiado.
- **Incluye al inicio de cada programa un comentario con tu nombre y apellidos.**
- **Entrega la carpeta xxxexamen comprimida con los archivos de código fuente .java.**

- **Tiempo estimado:** 2:20 horas

1. Complejidad Ciclomática (2)

ComplejidadCiclomatica.java
La complejidad ciclomática es una métrica utilizada en ingeniería del software para medir cuantitativamente la complejidad lógica de un programa. Su cálculo se basa en el diagrama de flujo determinado por las estructuras de control usadas en el código y el resultado define el número de caminos independientes dentro de un fragmento de código.

Para calcular la complejidad ciclomática de un algoritmo con una única entrada y salida se confeccionará, en primer lugar, un grafo de flujo como los de las figuras, en los que cada nodo (N) representa una sentencia, o un bloque de sentencias secuenciales, o la evaluación de una condición se una sentencia alternativa o repetitiva. Estos últimos nodos también se llaman Nodos Predicado (NP). Las aristas (A) por su parte representan los flujos posibles del programa, de un nodo a otro u otros.



Una vez confeccionado el grafo, la complejidad ciclomática (V(G)) se puede calcular se 3 formas:

1. **V(G) = número de regiones**. Entendiendo por región cada área distinta encerrada por notods y aristas e incluyendo también como una región más el área abierta exterior.
2. **V(G) = A - N + 2**
3. **V(G) = NP + 1**

Para los grafos de la imagen anterior la complejidad ciclomática sería de **3** para el de la izquierda y **4** para el de la derecha.

Podemos codificar un grafo como una matriz irregular, modificando la numeración de los nodos para que comience en cero, y en la que cada fila representa un nodo y contiene los números de los nodos a los que apunta. Por ejemplo, los grafos de la figura anterior se podrían codificar así:

```
int[][] grafo1 = {
    {1},
    {3, 5},
    {1},
    {2, 4},
    {6},
    {6},
    {}
};
```

```
int[][] grafo2 = {
    {1, 6},
    {2, 5},
    {3, 4},
    {4},
    {1},
    {0},
    {}
};
```

Implementa el siguiente método que recibe un grafo en el formato anterior y devuelve la complejidad ciclomática correspondiente:

```
static int complejidadCiclomatica(int[][] grafo);
```

2. Busca do Tesouro (4)

BuscaTesouro.java

Implementa un xogo no que o xogador terá que localizar un tesouro agachado nunha casilla oculta dun mapa bidimensional.

Ao inicio o programa solicitará ao usuario o número de filas e columnas do mapa e a continuación agachará o tesouro nunha casilla elexida ao chou.

A partir de ahí o programa solicitará repetidamente ao usuario o número de fila e columna no que pensa que está o tesouro ata que acerte.
Se o usuario non acerta a ubicación do tesouro, o programa daralle unha pista indicando en que dirección se atopa: “Máis ao {norte|nordés|leste|sueste|sur|suroeste|oeste|noroeste}

O usuario terá un número máximo de intentos para atopar o tesouro que se calculará como un 10% de casillas do mapa. Por exemplo, para un mapa de 10x15, é dicir de 150 casillas, o usuario terá un maximo de 15 intentos.

O programa rematará cando se esgoten o número de intentos ou se atope o tesouro, informando ao usuario do resultado e do número total de intentos requeridos.

O programa deberá ser robusto e soportar entradas inválidas do usuario, tanto por introducir caracteres alfanuméricos cando se agardan números como por introducir números de fila ou columna fora de rango.

3. Codificando Morse (4)

CodificandoMorse.java (Inspirado en El telegrama más corto - ¡Acepta el reto!)

El código Morse codifica cada carácter se codifica como una sucesión de puntos y rayas según una tabla como la siguiente:

Letra	Código		Letra	Código		Letra	Código		Letra	Código
A	.-		N	-.		H	....		U	..-
B	-...		O	---		I	..		V	...-
C	-.-.		P	.--.		J	.---		W	.--
D	-..		Q	--.-		K	-.-		X	-..-
E	.		R	.-.		L	.-..		Y	-.--
F	..-.		S	...		M	--		Z	--..
G	--.		T	-		!	-.-.--		?	..--..

En realidad, el mensaje se enviaba como una serie de pitidos. Según el estándar de 1922, un punto se envía como un pitido, una raya como tres pitidos (o un pitido de tres puntos de duración). Además, entre símbolo y símbolo (raya o punto) de una misma letra hay que esperar un punto, entre letras de una misma palabra se esperan tres puntos, y entre palabras cinco puntos.

Implementa el siguiente método que recibe una frase de no más de 80 letras mayúsculas del alfabeto inglés, signos de admiración, interrogación y espacios, y devuelve la duración del mensaje codificado en número de puntos.

```
static int numPuntosMorse(String frase);
```

Recurso: podes utilizar as seguintes estructuras que codifican as táboas do código Morse.

```
String letras = "ABCDEFGHIJKLMNOPQRSTUVWXYZ!?";
String[] letrasMorse = { ".-", "-...", "-.-.", "-..", ".", "..-.", "--.",
"....", "..", ".---", "-.-", ".-..",
        "--", "-.", "---", ".--.", "--.-", ".-.", "...", "_", "..-", "...-",
".--", "-..-", "-.--", "--..",
        "-.--", ".----.." };
```

Casos de Prueba

?	15
!	19
SI	11
YA NACIO	73