Ingeniería del Software

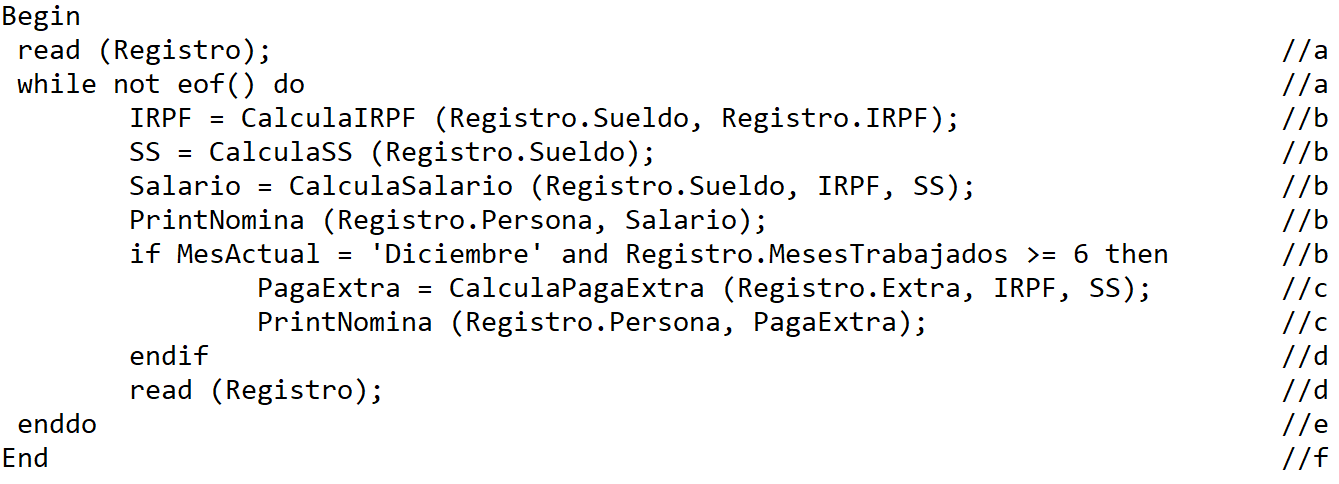
TEMA 6: PRUEBAS

Ejercicios Para Entregar (Trabajo Grupal)

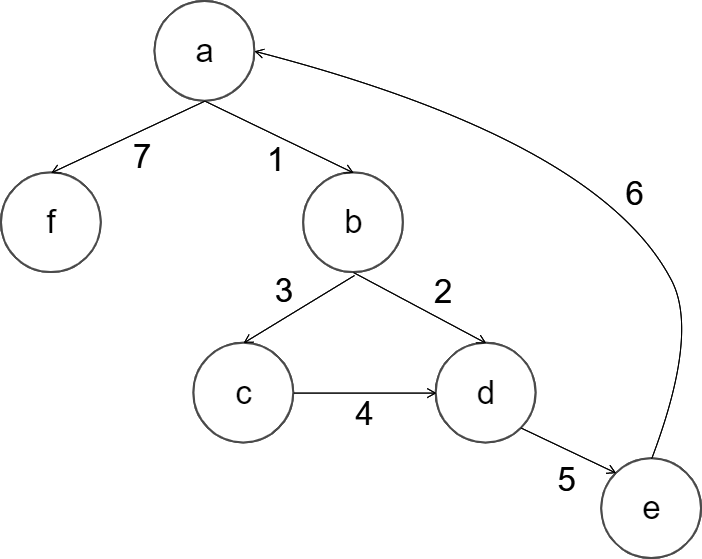
**Ejercicio 1:**

a) Representa el grafo de flujo para realizar pruebas de caja blanca. ¿Cuál es la complejidad ciclomática del fragmento de código?

Primero dividimos cada parte del código en nodos diferentes:



Y acto seguido representamos el grafo del flujo:



Complejidad ciclomática:

V(G)=Nº regiones cerradas+1 ⇒ 2+1=3

V(G)=Aristas–Nodos+2 ⇒ 7-6+2=3

Podemos decir entonces que su complejidad ciclomática es de 3.

b) Genera los casos de prueba necesarios para obtener una cobertura completa de decisión/condición. ¿Cuál es el mínimo número de casos de prueba que necesitaríamos?

Cada Condición debe tomar los valores verdadero y falso al menos una vez, garantizando a su vez la cobertura de Decisión (i.e., cada Decisión debe tomar los valores verdadero y falso al menos una vez). Se busca diseñar el menor número de casos de prueba que garanticen la cobertura de decisión/condición.

En el código proporcionado, solo hay dos condiciones:

| D1 | while not eof() do |
| --- | --- |
| D2 | if MesActual = 'Diciembre' and Registro.MesesTrabajados >= 6 then |

Por lo tanto, se debe tomar los valores verdadero y falso para dichas decisiones para garantizar que todos los posibles caminos se tomen al menos una vez. En el caso de que la condición D1 sea Falso, no es necesario considerar el valor de la condición dos, cubriendo ya todas las condiciones.

|  | Valor Verdadero | Valor Falso |
| --- | --- | --- |
| C1.1 | eof() = False | eof() = True |
| C2.1 | MesActual = 'Diciembre' | MesActual = 'Enero' |
| C2.2 | Registro.MesesTrabajados = 6 | Registro.MesesTrabajados = 1 |

| Caso de prueba 1: | |
| --- | --- |
| C1.1 = Verdadero  C2.1 = Verdadero  C2.2 = Verdadero | eof() = False  MesActual = 'Diciembre'  Registro.MesesTrabajados = 6 |

| Caso de prueba 2: | |
| --- | --- |
| C1.1 = Verdadero  C2.1 = Falso  C2.2 = Verdadero | eof() = False  MesActual = 'Enero'  Registro.MesesTrabajados = 6 |

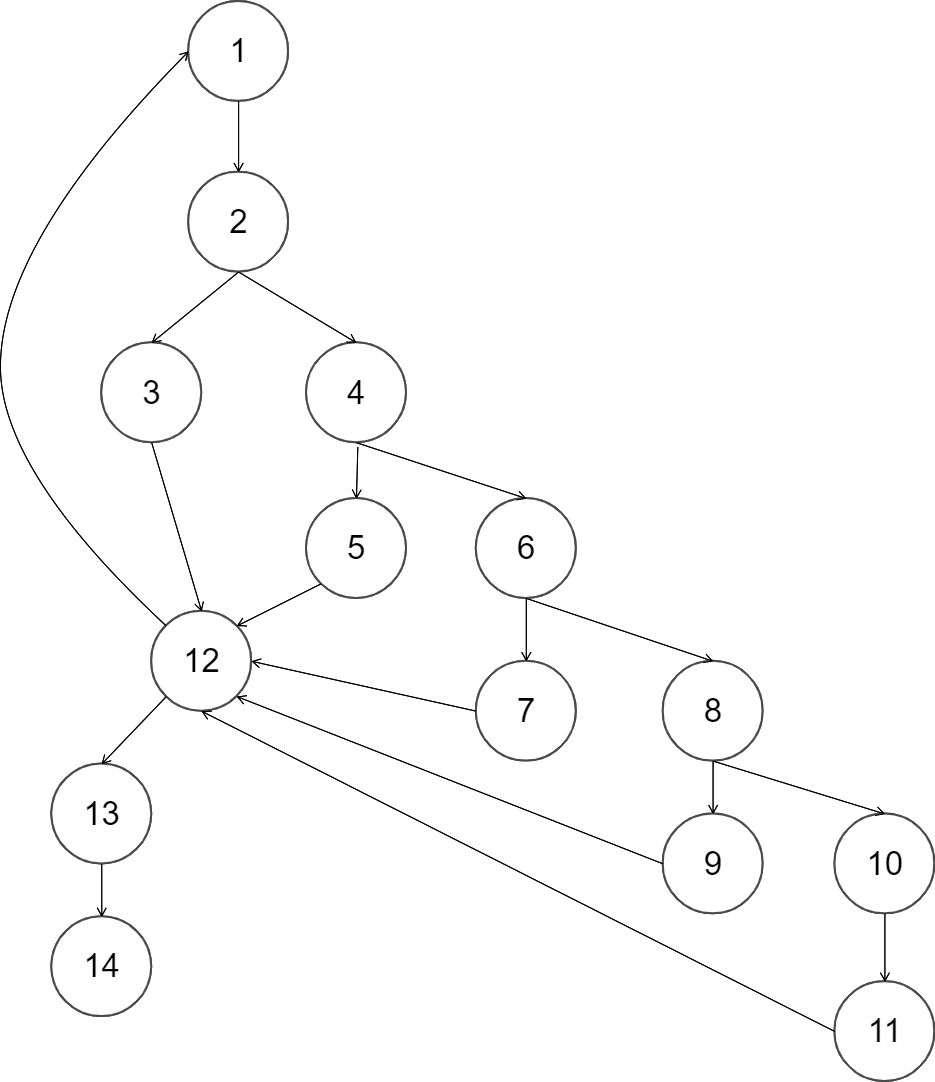
| Caso de prueba 3: | |
| --- | --- |
| C1.1 = Falso  C2 = Indiferente | eof() = True |

Por lo tanto, el mínimo de casos de prueba necesarios será 3 casos de prueba, ya que al ser la primera condición C1 Falso, no es necesario hacer un caso de prueba de la segunda condición ya que depende de la primera.

**Ejercicio 2:**

a) Respetando la numeración de líneas anterior, dibuja el grafo de flujo G. Calcula la complejidad ciclomática V(G) al menos de dos formas distintas.

El gráfico del flujo G queda de la siguiente forma:



Y sabiendo que:

Complejidad ciclomática:

| V(G)=Nº regiones cerradas+1 ⇒ **5+1=6** |
| --- |

| V(G)=Aristas–Nodos+2 ⇒ **18-14+2=6** |
| --- |

Podemos entonces confirmar que la complejidad ciclomática de G es de 6.

b) Se desea diseñar un conjunto de casos de prueba que garantice una cobertura completa de decisión/condición del fragmento de código que va de la línea 2 hasta la 11 del método Java anterior. Dibuja una tabla representando el número mínimo de casos de prueba necesario para ello, identificando decisiones, condiciones, entradas y salidas, de forma que se garantice la cobertura completa de decisión/condición de dicho fragmento de código.

Cada Condición debe tomar los valores verdadero y falso al menos una vez, garantizando a su vez la cobertura de Decisión (i.e., cada Decisión debe tomar los valores verdadero y falso al menos una vez). Se busca diseñar el menor número de casos de prueba que garanticen la cobertura de decisión/condición.

En el código proporcionado, solo hay cinco condiciones:

| D1 | if (a<0 & b<0 & c<0) |
| --- | --- |
| D2 | else if (a<0) |
| D3 | else if (b<0) |
| D4 | else if (c<0) |
| D5 | else |

Por lo tanto, se debe tomar los valores verdadero y falso para dichas decisiones para garantizar que todos los posibles caminos se tomen al menos una vez. Al ser todas las condiciones solo comprobadas si las anteriores son falsas, solo hará falta comprobar una vez por cada condición, y como las condiciones se repiten entre los if y elses, podemos reducir todo a las siguientes tres condiciones:

|  | Valor Verdadero | Valor Falso |
| --- | --- | --- |
| C1.1 | a=-1 | a=1 |
| C1.2 | b=-1 | b=1 |
| C1.3 | c=-1 | c=1 |

| Caso de prueba 1: | |
| --- | --- |
| C1.1 = Verdadero  C1.2 = Verdadero  C1.3 = Verdadero | a=-1  b=-1  c=-1 |

| Caso de prueba 2: | |
| --- | --- |
| C1.1 = Verdadero  C1.2 = Falso  C1.3 = Falso | a= -1  b= 1  c= 1 |

| Caso de prueba 3: | |
| --- | --- |
| C1.1 = Falso  C1.2 = Verdadero  C1.3 = Falso | a=1  b=-1  c=1 |

| Caso de prueba 4: | |
| --- | --- |
| C1.1 = Falso  C1.2 = Falso  C1.3 = Verdadero | a=1  b=1  c= -1 |

| Caso de prueba 5: | |
| --- | --- |
| C1.1 = Falso  C1.2 = Falso  C1.3 = Falso | a=1  b=1  c=1 |

c) Supongamos que queremos además diseñar el número mínimo de casos de pruebas que garantice una cobertura completa de decisión del código comprendido entre las líneas 2-11. ¿Coincidiría con el número de casos de prueba calculado en el apartado anterior (para la cobertura completa de decisión/condición)? ¿Por qué? Justifica adecuadamente tu respuesta

Cada Condición debe tomar los valores verdadero y falso al menos una vez, garantizando a su vez la cobertura de Decisión (i.e., cada Decisión debe tomar los valores verdadero y falso al menos una vez). Se busca diseñar el menor número de casos de prueba que garanticen la cobertura de decisión/condición.

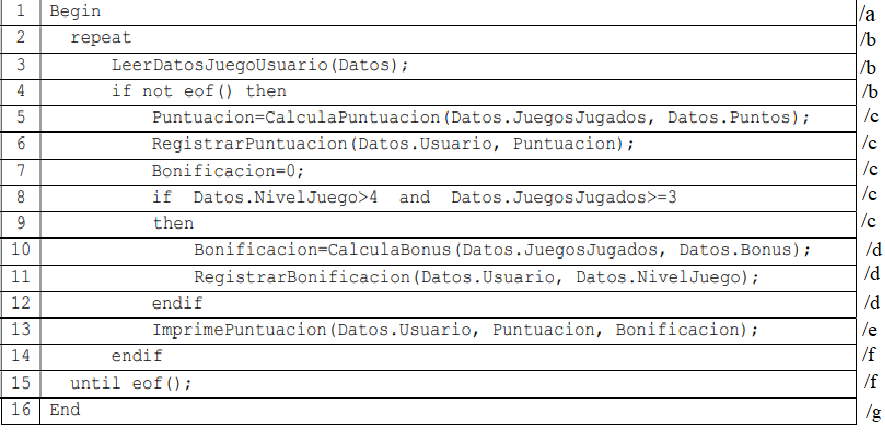
En el código proporcionado hay las siguientes condiciones:

| D1 | if (a<0 & b<0 & c<0) |
| --- | --- |
| D2 | else if (a<0) |
| D3 | else if (b<0) |
| D4 | else if (c<0) |

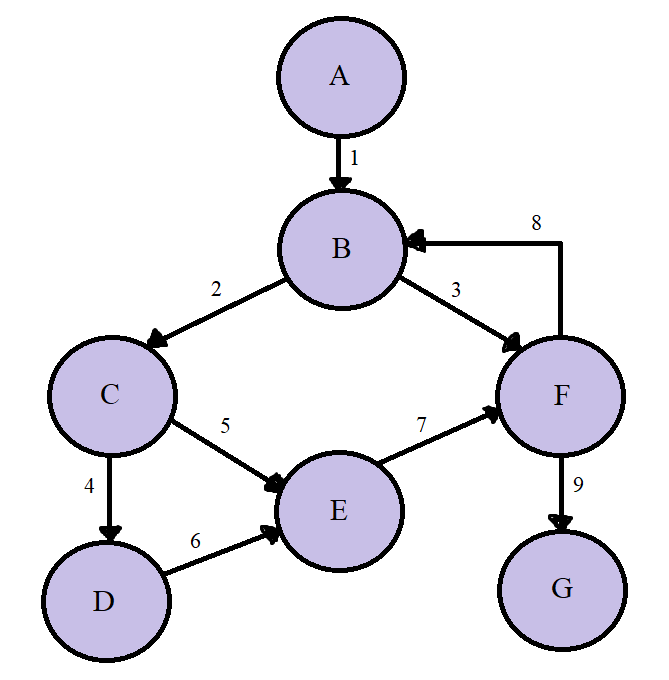
El número de casos calculado en el apartado anterior es correcto, ya que las decisiones son similares a las condiciones que han de cumplirse dentro de dichas condiciones. Es decir, las decisiones son condicionales de las condiciones de la primera decisión. Por lo tanto el número de casos de prueba calculado es correcto.

**Ejercicio 3:**

a) Representa el grafo de flujo para realizar pruebas de caja blanca y calcula la complejidad ciclomática.

****

El grafo de flujo resultante sería el siguiente:



Calculamos la complejidad ciclomática del grafo anterior:

* Número de regiones cerradas:

V(G) = 1 (3, 8) + 1 (2, 3, 5, 7) + 1 (4 ,5, 6) + 1 = 4

* Número de arcos - número de nodos + 2:

V(G) = 9 flechas -7 nodos + 2 = 4

* Número de nodos de condición:

V(G) = 3 nodos (B, C, F) + 1 = 4

Por lo tanto, la complejidad ciclomática es 4.

a) Supongamos que la interfaz del módulo es la que se especifica a continuación. Con el objetivo de realizar pruebas de caja negra, ¿cuáles serían las clases de equivalencia? Represéntalas en una tabla. Diseña, además, 5 casos de prueba de caja negra.

Los parámetros de entrada y de salida son los siguientes:

| Atributo | Clases validas | Clases no validas |
| --- | --- | --- |
| Usuario | 1.Cadena de caracteres y espacios  2.Longitud = 10  3.Primer caracter ∈ [a-zA-Z]  4.Ultimo caracter = # | 1.Caracteres numericos  2.Logitud != 10  3.Ultimo caracter != #  4.Primer caracter ∉ [a-zA-Z]  5.Signos de puntuación |
| Retorno | 1.Valores en rango de [-3, 0] | 1.Valor > 0  2.Valor < -3  3.Valor no numerico |

Con las condiciones previamente establecidas, podemos diseñar los siguientes 5 casos de prueba:

| Nº prueba | Usuario | Retorno |
| --- | --- | --- |
| 1 | Guille MC# | 0 |
| 2 | Chabokunn# | 0 |
| 3 | danivarelas# | -1 |
| 4 | Santi\_Ago# | -2 |
| 5 | Dream Team | -3 |

**Ejercicio 4:**

**Se pide realizar una tabla completa con las clases de equivalencia.**

| Atributo | Clases validas | Clases no validas |
| --- | --- | --- |
| Fecha ingreso | 1.Formato = DD-MM-AAA  2.DD en rango de [1, 31]  3.MM en rango de [1, 12]  4.AAAA en rango de [0000, 9999] | 1.Formato != DD-MM-AAAA  2.DD no en rango de [1, 31]  3.MM no en rango de [1, 12]  4.AAAA no en rango de [0000, 9999] |
| Fecha alta | 1.Formato = DD-MM-AAA  2.DD en rango de [1, 31]  3.MM en rango de [1, 12]  4.AAAA en rango de [0000, 9999] | 1.Formato != DD-MM-AAAA  2.DD no en rango de [1, 31]  3.MM no en rango de [1, 12]  4.AAAA no en rango de [0000, 9999] |
| Fallecido | 1.No fallecido = 0  1.Fallecido = 1 | 1.Valor < 0  2.Valor >1  3.Caracteres no numéricos |
| Ciudad residencia | 1.Cadena de caracteres  2.Longitud3= 3  3.Caracteres ∈ [A-Z] | 1.Caracteres numericos  2.Longitud != 3  4.Caracteres ∉ [A-Z]  5.Signos de puntuación |
| Edad | 1.Valor numerico  2.Longitud = 2  2.Valor en rango de [00, 99] | 1.Caracteres no numéricos  2.Longitud != 2  3.Valor no en rango[00, 99] |
| Genero | 1.Longitud = 1  2.Caracter = M  3.Caracter = F | 1.Longitud != 1  2.Caracter != M  3.Caracter !=F |
| Codigo UCI | 1.Formato = NXX  2.N caracter ∈ [A-D]  3.XX valor numérico contenido en [00,99] | 2.Formato != NXX  2.N caracter ∉ [A-D]  3.XX valor numérico fuera de [00,99] |
| Retorno (contagio) | 1.Formato = ContagioTipoNN  2.NN valor numérico contenido en [00,99] | 1.Formato != ContagioTipoNN  2.NN valor numérico fuera de [00,99]  3.NN caracter no numerico |
| Retorno (dias UCI) | 1.Valor numerico  2.Longitud = 3  3.Valor numérico contenido en [000, 999] | 1.Valor no numerico  2.Longitud != 3  3.Valor numérico fuera de [000, 999] |