

CAJA BLANCA

CHRISTIAN MILLÁN SORIA

1º DAW TARDE

1. Si tenemos el siguiente algoritmo, que averigua si un número es primo o no:

```
algoritmo primos
  escribir "Numero: "
  leer num

  cont<-2;
  swPrimo<-'V'

  mientras((cont<=(num/2))y(swPrimo=='V'))hacer
    si((num mod cont)==0)entonces
      swPrimo<-F
    fin si

    cont<-cont+1
  fin mientras

  si(swPrimo=='V')entonces
    escribir num, " SI es primo"
  sino
    escribir num, " NO es primo"
  fin si
fin algoritmo
```

Indica varios casos de prueba posibles, así como algún procedimiento de prueba.

Algunos casos de prueba posibles podrían ser, por ejemplo:

- Caso de prueba válido: Ingresar el número 2, que es el número primo más pequeño.
- Caso de prueba válido: Ingresar el número 7, que es un número primo.
- Caso de prueba inválido: Ingresar el número 10, que no es un número primo.
- Caso de prueba válido: Ingresar el número 23, que es un número primo.
- Caso de prueba válido: Ingresar el número 29, que es un número primo.

Un posible procedimiento de prueba para verificar que el algoritmo funciona correctamente sería el siguiente (por pasos):

- Paso 1: Ingresar un número primo como entrada, por ejemplo, 7.
- Paso 2: Ejecutar el algoritmo.
- Paso 3: Verificar que el algoritmo indique que el número ingresado es un número primo.
- Paso 4: Repetir los pasos 1 a 3 con otros números primos para asegurarse de que el algoritmo los identifique correctamente.

- Paso 5: Ingresar un número que no sea primo, por ejemplo, 10.
- Paso 6: Ejecutar el algoritmo.
- Paso 7: Verificar que el algoritmo indique que el número ingresado no es un número primo.
- Paso 8: Repetir los pasos 5 a 7 con otros números que no sean primos para asegurarse de que el algoritmo los identifique correctamente.
- Paso 9: Asegurarse de que el algoritmo no tenga errores de lógica para cualquier número de entrada mediante el análisis del código fuente y la verificación de los cálculos realizados en el código.

2. ¿Qué diferencia existe entre las pruebas de caja blanca y las de caja negra? En el ejemplo anterior, ¿en qué consistirían?

Las pruebas de caja blanca y las de caja negra son dos enfoques diferentes para realizar pruebas de software.

Las pruebas de caja negra se basan en probar el software desde el punto de vista del usuario final, sin conocer los detalles internos de cómo se implementa el software. En este enfoque, el software se trata como una "caja negra" y se prueban sus entradas y salidas sin considerar su funcionamiento interno. Las pruebas de caja negra se centran en evaluar si el software cumple con los requisitos funcionales y no funcionales, y si el software es fácil de usar y comprender.

Por otro lado, las pruebas de caja blanca se enfocan en evaluar el software desde su funcionamiento interno y se basan en el conocimiento detallado de la implementación del software. En este enfoque, el tester conoce la estructura interna del código fuente y de la aplicación, y se asegura de que el software se ejecute según lo previsto. Las pruebas de caja blanca se centran en la cobertura de código, en la verificación de la lógica interna del software, y en la identificación de posibles problemas de rendimiento y seguridad.

En el ejemplo anterior, las pruebas de caja blanca implicarían examinar el código fuente del algoritmo para diseñar casos de prueba que cubran todas las posibles rutas de ejecución, lo que se puede lograr mediante pruebas de cobertura de código para asegurarse de que todas las declaraciones y ramas de código se hayan ejecutado al menos una vez. Por otro lado, las pruebas de caja negra implicarían proporcionar diferentes entradas al algoritmo y verificar si las salidas corresponden a las salidas esperadas.

3. ¿En qué consiste la cobertura o cubrimiento? Busca en el ejemplo 1 los diferentes tipos de coberturas que existen, si no encuentras ninguno, plantea un ejemplo adicional.

La cobertura o cubrimiento es una medida que se utiliza para evaluar la calidad de las pruebas realizadas en un programa. Consiste en medir qué porcentaje del código ha sido ejecutado durante la realización de las pruebas.

En el ejemplo 1, se podrían aplicar diferentes tipos de coberturas, por ejemplo:

- Cobertura de sentencias: en este caso, se evalúa si se han ejecutado todas las sentencias del código. En el ejemplo, se debería verificar que se han ejecutado todas las sentencias dentro del bucle mientras, así como las sentencias condicionales dentro del mismo.
- Cobertura de ramas: este tipo de cobertura evalúa si se han evaluado todas las condiciones posibles de las sentencias condicionales. En el ejemplo, se debería verificar que se han evaluado tanto la rama verdadera como la falsa del condicional dentro del bucle mientras.
- Cobertura de condiciones: este tipo de cobertura evalúa si se han evaluado todas las combinaciones posibles de las condiciones en las sentencias condicionales. En el ejemplo, se debería verificar que se han evaluado tanto la condición `num mod cont == 0` como su negación.

La elección de qué tipo de cobertura aplicar dependerá del objetivo de las pruebas y del código que se esté evaluando. Por ejemplo, si se quiere evaluar la robustez del código, es posible que sea más relevante aplicar una cobertura de condiciones.

5. Utilizando la técnica de caja blanca del "Camino básico", obtén el grafo, el número d complejidad ciclomática y los casos de prueba del siguiente algoritmo:

```
public static int contarLetras(char cadena[], char caracter){
    int contador=0, n=0, lon=cadena.length;

    if(lon>0){
        do{
            if(cadena[contador]==caracter){
                n++;
            }

            contador++;
            lon--;
        }while(lon>0);
    }

    return n;
}
```

Grafo:

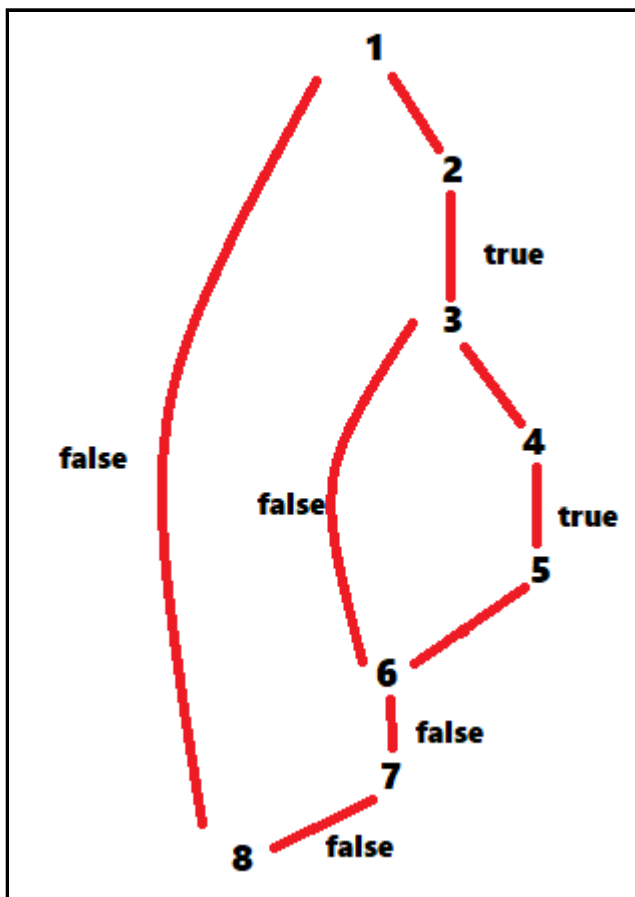
```

public static int contarLetras(char cadena[], char caracter){
    int contador=0, n=0, lon=cadena.length;

    if(lon>0){
        do{
            if(cadena[contador]==caracter){
                n++;
            }
            contador++;
            lon--;
        }while(lon>0);
    }

    return n;
}

```



Supongamos que tenemos una cadena de caracteres "hola mundo" y queremos contar la cantidad de letras "o" que hay en ella.

- Paso 1: `int contador=0, n=0, lon=cadena.length;`

Este paso simplemente inicializa las variables contador, n y lon. No hay posibles casos de prueba para este paso.

- Paso 2: `if(lon>0){`

Este paso comprueba si la cadena tiene al menos un caracter. Si la longitud de la cadena es mayor que cero, se ejecuta el bloque de código siguiente.

Posibles casos de prueba:

- Si la cadena está vacía, entonces lon será igual a cero y no se ejecutará el bloque de código siguiente.

cadena = "", caracter = "o"

Resultado esperado: 0

- Si la cadena tiene al menos un caracter, entonces lon será mayor que cero y se ejecutará el bloque de código siguiente.

cadena = "hola mundo", caracter = "o"

Resultado esperado: Continuar con el siguiente paso.

-
- Paso 3: do{

Este paso inicia un ciclo "do-while" que se ejecutará al menos una vez.

No hay posibles casos de prueba para este paso.

-
- Paso 4: if(cadena[contador]==caracter){

Este paso comprueba si el caracter en la posición "contador" de la cadena es igual al caracter que se desea contar. Si es así, se incrementa la variable "n".

Posibles casos de prueba:

- Si el caracter en la posición "contador" de la cadena no es igual al caracter que se desea contar, entonces no se incrementará la variable "n".

cadena = "hola mundo", caracter = "z"

Resultado esperado: 0

- Si el caracter en la posición "contador" de la cadena es igual al caracter que se desea contar, entonces se incrementará la variable "n".

cadena = "hola mundo", caracter = "o"

Resultado esperado: n = 2

-
- Paso 5: n++;

Este paso incrementa la variable "n" si el caracter en la posición "contador" de la cadena es igual al caracter que se desea contar.

No hay posibles casos de prueba para este paso.

-
- Paso 6: contador++; lon--;

Este paso incrementa el contador y decrementa lon en cada iteración del ciclo.

No hay posibles casos de prueba para este paso.

- Paso 7: }while(lon>0);

Este paso termina el ciclo "do-while" cuando lon es igual a cero.

No hay posibles casos de prueba para este paso.

- Paso 8: return n;

Este paso devuelve la cantidad de veces que se encontró el caracter en la cadena.

No hay posibles casos de prueba para este paso.

En cuanto a los posibles caminos que puede tomar el algoritmo, hay dos posibles situaciones:

- En cuanto a los posibles caminos que puede tomar el algoritmo, hay dos posibles situaciones:
- Si la longitud de la cadena es igual a cero, el algoritmo no ejecutará el bloque de código siguiente y devolverá cero como resultado.

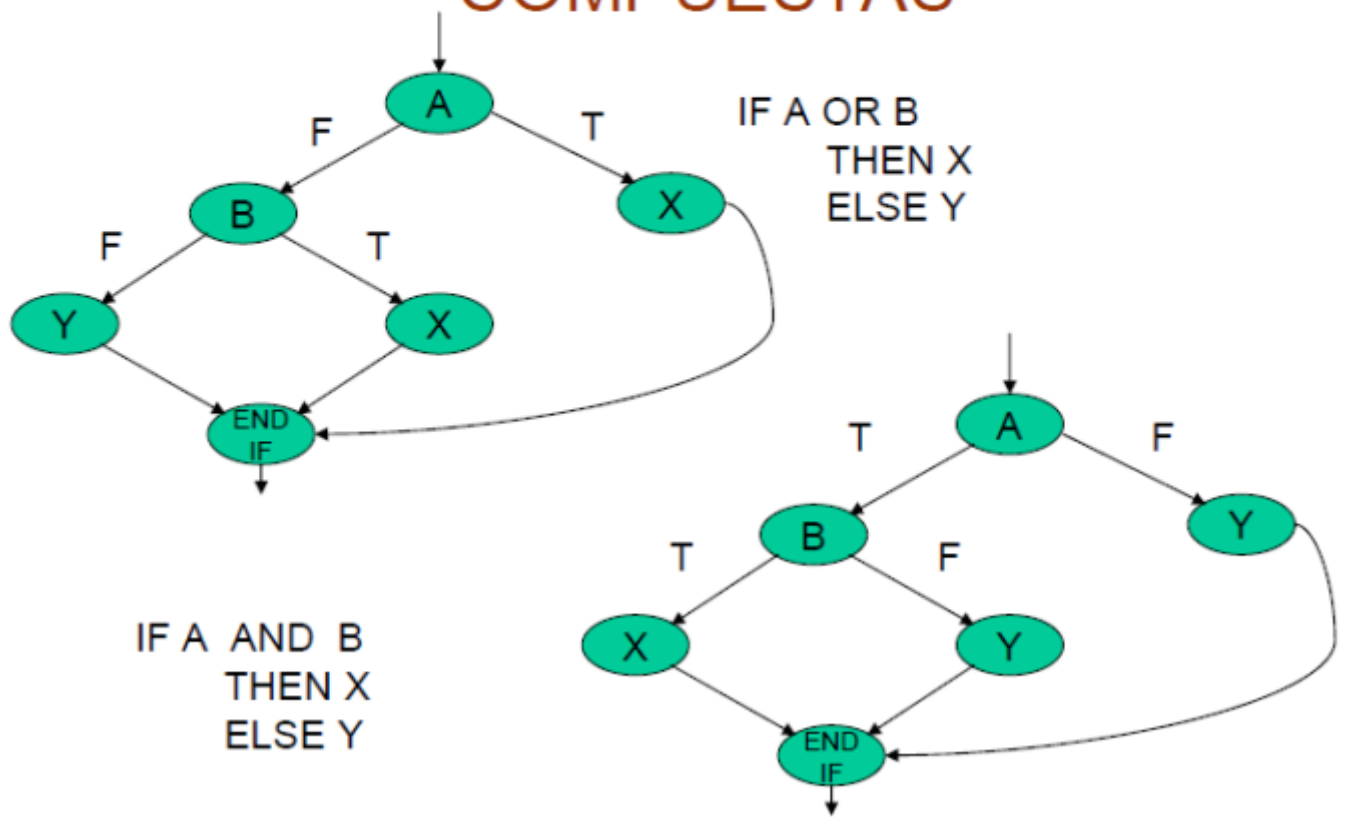
Por lo tanto, el número de complejidad de este algoritmo es $O(n)$. El algoritmo recorre la cadena una vez, y por cada iteración, realiza una operación constante (comparar el caracter actual con el caracter que se desea contar, incrementar el contador y decrementar la longitud de la cadena). Por lo tanto, el tiempo de ejecución del algoritmo aumenta linealmente con el tamaño de la cadena de entrada.

Casos de prueba posibles:

- Caso de prueba 1: cadena vacía, caracter no importa. En este caso, se espera que la función devuelva 0.
- Caso de prueba 2: cadena con un solo carácter que no coincide con el caracter buscado. En este caso, se espera que la función devuelva 0.
- Caso de prueba 3: cadena con un solo carácter que coincide con el caracter buscado. En este caso, se espera que la función devuelva 1.
- Caso de prueba 4: cadena con varios caracteres, pero ninguno coincide con el caracter buscado. En este caso, se espera que la función devuelva 0.
- Caso de prueba 5: cadena con varios caracteres, algunos de los cuales coinciden con el caracter buscado. En este caso, se espera que la función devuelva el número de caracteres que coinciden con el caracter buscado.

6. Cuando tenemos condiciones lógicas compuestas se nos complica un poco más la forma de obtener el grafo. El siguiente esquema lo resume de forma bastante clara:

GRAFOS ASOCIADOS A CONDICIONES LÓGICAS COMPUESTAS



Utilizando la técnica del "Camino básico", intenta obtener el grafo, el número de complejidad ciclomática y los casos de prueba del siguiente algoritmo que calcula la media aritmética de 2 números:

```

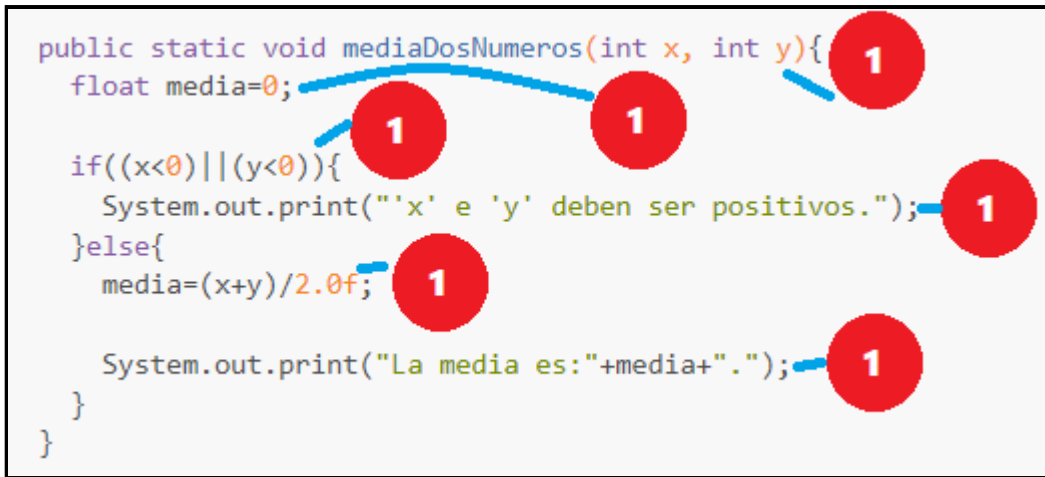
public static void mediaDosNumeros(int x, int y){
    float media=0;

    if((x<0)|| (y<0)){
        System.out.print("'x' e 'y' deben ser positivos.");
    }else{
        media=(x+y)/2.0f;

        System.out.print("La media es:"+media+".");
    }
}
  
```

Grafo:

```
public static void mediaDosNumeros(int x, int y){  
    float media=0;  
    if((x<0)||(y<0)){  
        System.out.print("'x' e 'y' deben ser positivos.");  
    }else{  
        media=(x+y)/2.0f;  
        System.out.print("La media es:"+media+".");  
    }  
}
```



Posibles casos de prueba:

- Caso de prueba válido: Ingresar los números 2 y 4.
- Caso de prueba válido: Ingresar los números 10 y 20.
- Caso de prueba inválido: Ingresar los números -5 y 10.

En cuanto al procedimiento de prueba, podríamos seguir los siguientes pasos:

- Ingresar dos números positivos como entrada, por ejemplo, 4 y 6.
- Ejecutar el algoritmo.
- Verificar que el resultado de la media sea 5.
- Repetir los pasos 1 a 3 con otros pares de números positivos para asegurarse de que el algoritmo calcule la media correctamente.
- Ingresar un número negativo y un número positivo como entrada, por ejemplo, -5 y 10.
- Ejecutar el algoritmo.
- Verificar que el algoritmo indique que los números deben ser positivos.
- Repetir los pasos 5 a 7 con otros pares de números que incluyan al menos un número negativo para asegurarse de que el algoritmo valide correctamente la entrada.