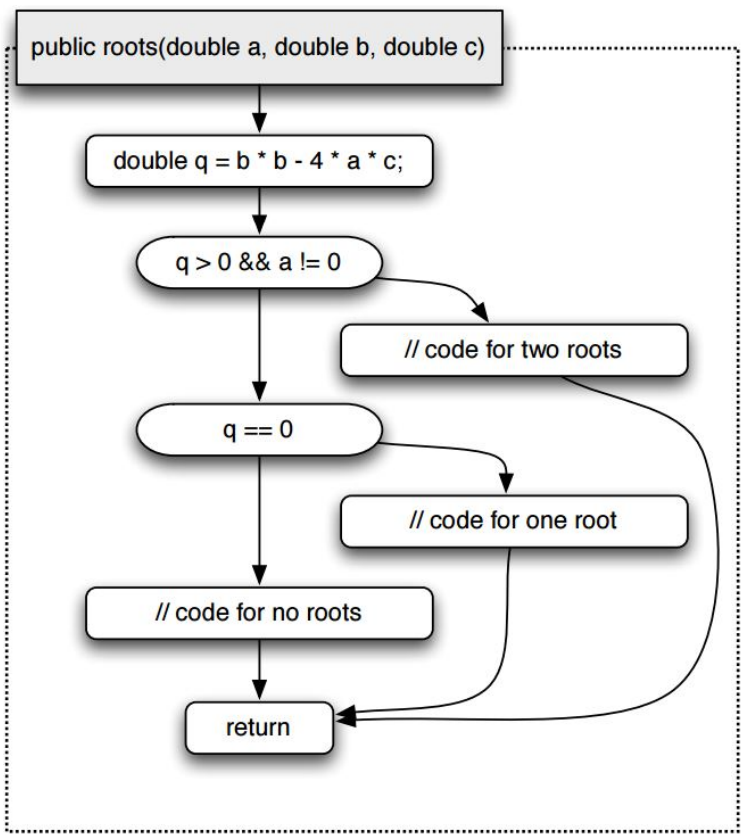
Testing

Algoritmos y Estructuras de Datos I

Departamento de Computación

16 de Mayo del 2018

Diagrama de Control de Flujo (Repaso de la teórica)



- El Diagrama de Control de Flujo (abreviado a veces CFG por Control Flow Graph, su nombre en inglés) es la representación gráfica de un programa (que instrucciones vamos a ejecutar, en que orden y bajo que condiciones).
- La definición del CFG de un programa es estática, porque la hacemos en base al código (no a ejecuciones particulares).
- A nosotros nos va a servir para definir criterios de adecuación de test suites[1].

[1] test suite: el conjunto de casos de test que vamos a usar para mostrar que nuestro programa tiene un comportamiento determinado (correcto).

Diagrama de Control de Flujo

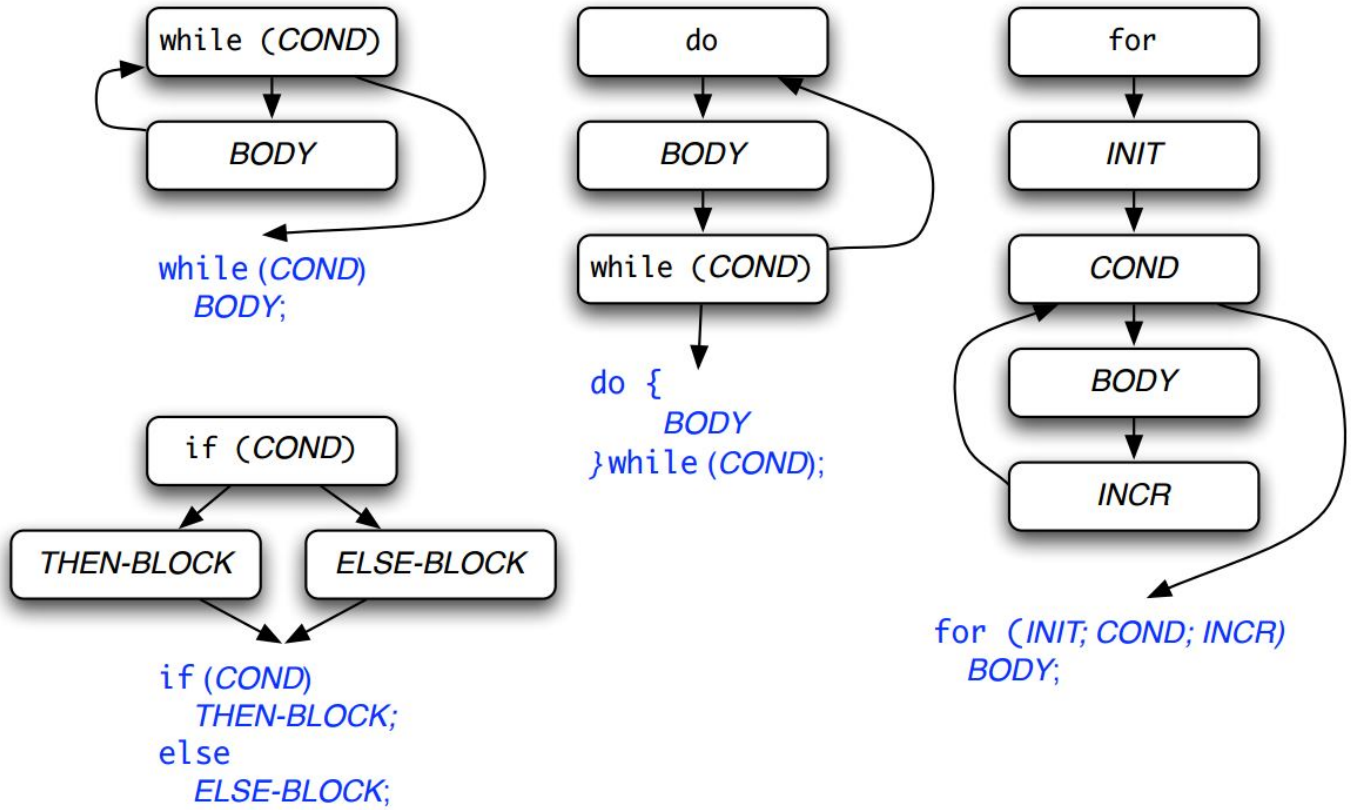
¿Cómo contruimos el CFG para un programa?

Diagrama de Control de Flujo

¿Cómo contruimos el CFG para un programa?
Lo vamos a contruir recursivamente en base a las distintas estructuras de control de flujo que haya (if, while, etc. . .)

Diagrama de Control de Flujo

¿Cómo contruimos el CFG para un programa?
Lo vamos a contruir recursivamente en base a las distintas estructuras de control de flujo que haya (if, while, etc...)



Ejercicio

Dado el siguiente programa, construir su Diagrama de Control de Flujo:

```
int factorial(int n){
    int result;
    if(n==1){
        result = 1;
    }
    result = n;
    n--;
    while(n > 1){
        result = result * n;
        n--;
    }
    return result;
}
```

Criterios de Adecuación

- Como ya sabemos, no existe ninguna técnica practicable que garantice la ausencia de errores, el testing exhaustivo (probar que para cada valor de entrada el valor de salida es el que corresponde) es demasiado caro en general y si agarramos un subconjunto de casos de test, no podemos garantizar que cubran todos los posibles bugs[1].

[1] a veces incluso hay programas para los cuales un caso de test puede pasar algunas veces y fallar otras (programas aleatorios o paralelos, por ejemplo).

Criterios de Adecuación

- Como ya sabemos, no existe ninguna técnica practicable que garantice la ausencia de errores, el testing exhaustivo (probar que para cada valor de entrada el valor de salida es el que corresponde) es demasiado caro en general y si agarramos un subconjunto de casos de test, no podemos garantizar que cubran todos los posibles bugs[1].
- Sin embargo, podemos definir criterios basados en el análisis de programas ya existentes y sus errores o en **heurísticas**, basadas en la estructura del programa.

[1] a veces incluso hay programas para los cuales un caso de test puede pasar algunas veces y fallar otras (programas aleatorios o paralelos, por ejemplo).

Cubrimiento

- Un parametro que vamos a usar para saber si una test suite es adecuada, es ver si usa todo el código a testear, ya que si hay una falla en el código que no estamos usando, no vamos a poder detectarla, pero... ¿Qué quiere decir usar todo el códgio?

Cubrimiento

- Un parametro que vamos a usar para saber si una test suite es adecuada, es ver si usa todo el código a testear, ya que si hay una falla en el código que no estamos usando, no vamos a poder detectarla, pero... ¿Qué quiere decir usar todo el códgio?
- Los diferentes criterios de cubrimiento de código nos van a especificar en cada caso, que quiere decir esto.

Cubrimiento de sentencias

- El cubrimiento de sentencias o líneas (statement coverage o line coverage) nos dice que nuestro test suite debería cubrir la mayor cantidad de líneas de código posibles. Esto es, óptimamente, ejecutar al menos 1 vez **entre todos los test del test suite** cada línea de código.

Cubrimiento de sentencias

- El cubrimiento de sentencias o líneas (statement coverage o line coverage) nos dice que nuestro test suite debería cubrir la mayor cantidad de líneas de código posibles. Esto es, óptimamente, ejecutar al menos 1 vez **entre todos los test del test suite** cada línea de código.
- Esto equivale a ver si pasamos una vez por cada nodo del CFG del programa.

Ejercicio

Dado el siguiente programa, dar un test suite que cubra todas sus líneas:

```
    int factorial(int n){
L1:     int result;
L2:     if(n==1){
L3:         result = 1;
        }
L4:     result = n;
L5:     n--;
L6:     while(n > 1){
L7:         result = result * n;
L8:         n--;
        }
L9:     return result;
    }
```

Cubrimiento de sentencias

- Normalmente, nuestro objetivo es pensar una test suite que pueda ejecutar todas las líneas de código.

Cubrimiento de sentencias

- Normalmente, nuestro objetivo es pensar una test suite que pueda ejecutar todas las líneas de código.
- ¿Siempre es posible conseguir esto?¿Qué quiere decir que no lo sea?¿Qué deberíamos hacer en ese caso?

Cubrimiento de decisiones

- Otro criterio que podemos utilizar es el de cubrimiento de decisiones (branch coverage). En este caso, lo que vamos a querer es que se tomen todas las decisiones posibles entre todos los tests (esto es, que cada **condición** evaluada se tome ambos valores posibles, `true` y `false`).

Cubrimiento de decisiones

- Otro criterio que podemos utilizar es el de cubrimiento de decisiones (branch coverage). En este caso, lo que vamos a querer es que se tomen todas las decisiones posibles entre todos los tests (esto es, que cada **condición** evaluada se tome ambos valores posibles, `true` y `false`).
- Pensar: Si tenemos cubrimiento de sentencias,¿tenemos cubrimiento de decisiones?¿Y al revés?

Ejercicio

Dado el siguiente programa, dar una test suite que de mayor cubrimiento de decisiones:

```
    int esTriangulo(int lado1, int lado2, int lado3){
L1:     bool res = true;
L2:     if(lado1 + lado2 <= lado3){
L3:     res = false;
        }
L4:     if(lado2 + lado3 <= lado1){
L5:         res = false;
        }
L6:     if(lado1 + lado3 <= lado2){
L7:      res = false;
        }
L8:     return res;
    }
```

Ejercicio

Dado el siguiente programa, dar una test suite que de mayor cubrimiento de decisiones:

```
    int esTriangulo(int lado1, int lado2, int lado3){
L1:     bool res = true;
L2:     if(lado1 + lado2 <= lado3){
L3:     res = false;
        }
L4:     if(lado2 + lado3 <= lado1){
L5:         res = false;
        }
L6:     if(lado1 + lado3 <= lado2){
L7:      res = false;
        }
L8:     return res;
    }
```

Si ejecuto esTriangulo(0,0,0),¿cubro sentencias?¿Cubro decisiones?

Ejercicio

- Contruir el CFG del programa.
- Escribir un test suite que cubra todas las líneas del programa.

Ejercicio

- Contruir el CFG del programa.
- Escribir un test suite que cubra todas las líneas del programa.
- Escribir un test suite que cubra todas las decisiones (branches) del programa.

Ejercicio

- Contruir el CFG del programa.
- Escribir un test suite que cubra todas las líneas del programa.
- Escribir un test suite que cubra todas las decisiones (branches) del programa.
- ¿Es posible escribir para este programa un test suite que cubra todas las líneas pero no cubra todas las decisiones? En caso afirmativo, describirlo. En caso negativo, justificarlo.

```
   int diferenciaMasGrande(vector<int> a){
L1:    int longitudVector = a.size();
L2:    int res = 0;
L3:    int i = 0;
L4:    while (i < longitudVector){
L5:        int j = i;
L6:        while(j<longitudVector){
L7:            if(a[i]-a[j]>res){
L8:                res = a[i] - a[j];
               }
L9:            j++;
           }
L10        i++;
       }
L11    return res;
   }
```

```
   int diferenciaMasGrande(vector<int> a){
L1:    int longitudVector = a.size();
L2:    int res = 0;
L3:    int i = 0;
L4:    while (i < longitudVector){
L5:        int j = i;
L6:        while(j<longitudVector){
L7:            if(a[i]-a[j]>res){
L8:                res = a[i] - a[j];
               }
L9:            j++;
           }
L10        i++;
       }
L11    return res;
   }
```

¿Hay algún defecto en el programa?

```
   int diferenciaMasGrande(vector<int> a){
L1:    int longitudVector = a.size();
L2:    int res = 0;
L3:    int i = 0;
L4:    while (i < longitudVector){
L5:        int j = i;
L6:        while(j<longitudVector){
L7:            if(a[i]-a[j]>res){
L8:                res = a[i] - a[j];
               }
L9:            j++;
           }
L10        i++;
       }
L11    return res;
   }
```

¿Hay algún defecto en el programa? ¿Nuestro test suite lo detecta?