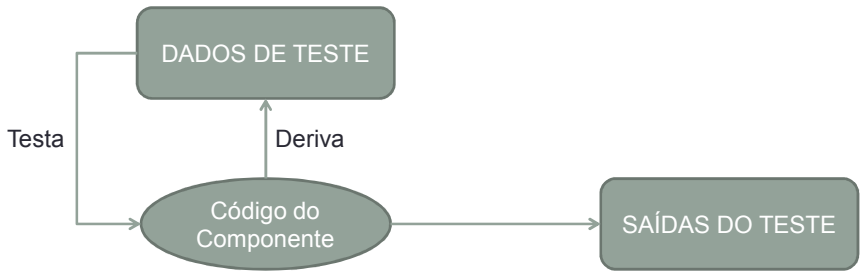
TESTE DE CAMINHO

Prof. Rafael Dias Ribeiro

Teste de Caminho

- Objetivo: Garantir que todos os possíveis caminhos sejam executados pelo menos uma vez, garantindo o conjunto de casos de teste.
- OBS: Em um processo de desenvolvimento Orientado à Objetos, este teste, pode ser utilizado quando são testados módulos associados a objetos.



Teste de Caminho

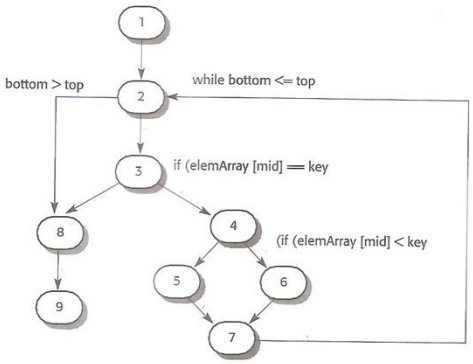
- Não testam todas as possíveis combinações de todos os caminhos no programa. Existem um número infinito de possíveis combinações de caminhos em programas com loops. Os defeitos se manifestam quando surgem combinações particulares de caminhos que podem estar presentes, embora todas as possíveis combinações tenham sido executadas pelo menos uma vez.

```
Class BinSeach {

// Esse é um encapsulamento de uma função de busca
// binária que considera um vetor de objetos ordenados e uma chave
// e retorna um objeto com 2 atributos, chamados
// index – o valor do índice de vetor
// found – um booleano que indica se uma chave está ou não no vetor
// Um objeto é retornado pois não é possível em Java passar
// tipos básicos por referência para uma função e
// retornar dois valores
// A chave será –1 se o elemento não for encontrado

    public static void search ( int key, int [ ] elemArray, Result r )
    {
        int bottom = 0 ;
        int top = elemArray.length – 1 ;
        int mid ;
        r.found = false ; r.index = –1 ;
        while ( bottom <= top )
        {
            mid = (top + bottom) / 2 ;
            if (elemArray [mid] = = key)
            {
                r.index = mid ;
                r.found = true ;
                return ;
            } // if part
            else
            {
                if (elemArray [mid] < key)
                    bottom = mid + 1 ;
                else
                    top = mid – 1 ;
            }
        } //while loop
    } // search
} //BinSearch
```

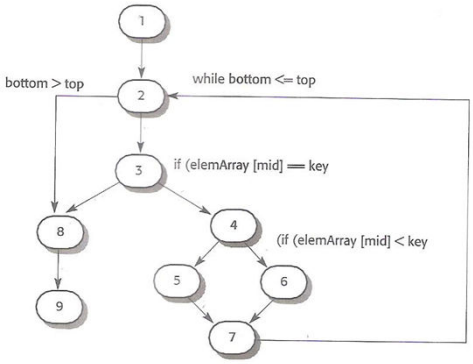
Exemplo: Função Busca Binária

```
Class BinSeach {
// Esse é um encapsulamento de uma função de busca
// binária que considera um vetor de objetos ordenados e uma chave
// e retorna um objeto com 2 atributos, chamados
// index – o valor do índice de vetor
// found – um booleano que indica se uma chave está ou não no vetor
// Um objeto é retornado pois não é possível em Java passar
// tipos básicos por referência para uma função e
// retornar dois valores
// A chave será –1 se o elemento não for encontrado
    public static void search ( int key, int [ ] elemArray, Result r )
    {
        int bottom = 0 ;
        int top = elemArray.length – 1 ;
        int mid ;
        r.found = false ; r.index = –1 ;
        while ( bottom <= top )
        {
            mid = (top + bottom) / 2 ;
            if (elemArray [mid] == key)
            {
                r.index = mid ;
                r.found = true ;
                return ;
            } // if part
            else
            {
                if (elemArray [mid] < key)
                    bottom = mid + 1 ;
                else
                    top = mid – 1 ;
            }
        } //while loop
    } // search
} //BinSearch
```

Exemplo: Função Busca Binária



Caminhos Independentes:
1,2,3,8,9
1,2,3,4,6,7,2
1,2,3,4,5,7,2
1,2,3,4,6,7,2,8,9



Caminhos Independentes:
1,2,3,8,9
1,2,3,4,6,7,2
1,2,3,4,5,7,2
1,2,3,4,6,7,2,8,9

O número de caminhos independentes pode ser descoberto calculando a **Complexidade Ciclomática**, de qualquer grafo G.

CC(G) = Número de Ramos – Número de Nós + 2

Ex:
CC = 11-9+2
CC = 4

Teste de Caminho

- Para programas sem declaração "*goto*" o valor da CC é 1 a mais que o número de condições no programa.
- Em condições de mais de um teste, deve-se contar cada teste, assim se houver 6 declarações *IF* e 1 loop *while*, com todas as condicionais simples, a CC será 8.
- Se uma expressão condicional for composta de 2 operadores lógicos (*and* ou *or),* a CC será 10

Teste de Caminho

- Após a descoberta do número de possíveis caminhos, deve-se projetar os casos de teste para executar cada um dos caminhos pelo menos 1 vez.
- O número mínimo de casos de teste requerido para testar todos os caminhos é equivalente a complexidade ciclomática.

Notas de Aula retiradas do Livro

Livro:
ENGENHARIA DE SOFTWARE
SOMMERVILLE, IAN
6 Ed.