Teste de Software

# Estratégias de Teste Estrutural de Software

PUC Minas

Instituto de Ciências Exatas
e Informática

Prof. Lesandro Ponciano

Departamento de Engenharia de Software
e Sistemas de Informação (DES)

# Objetivos da Aula

- Contextualizar a técnica de Teste Estrutural de software
- Introduzir o conceito de “grafo de fluxo de controle” de um programa
- Introduzir a análise de critério de adequação e de cobertura

# A Técnica de Teste Estrutural

- Método para projeto de casos de teste utilizando a estrutura de controle do módulo
  - Casos de teste são gerados a partir da implementação

- Objetivos
  - Garantir que todos os caminhos independentes dentro de um módulo sejam executados ao menos uma vez
  - Executar todas as decisões lógicas do programa, tanto a parte verdadeira quanto a falsa
  - Executar todas as estruturas de repetição nos seus limites definidos ou operacionais
  - Garantir a validade das estruturas de dados internas

# Grafo de Fluxo de Controle (GFC)

- Um programa *P* é representado como um grafo direcionado *G=(N, E, s)*, onde
  - *N* é o conjunto de vértices
  - *E* é o conjunto de arestas
  - *s* é o vértice de entrada

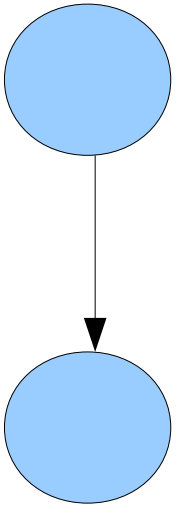
- No Grafo G
  - Um vértice indica um bloco indivisível de comandos
  - Uma aresta indica um possível desvio de um bloco para outro

# Grafo de Fluxo de Controle (GFC)

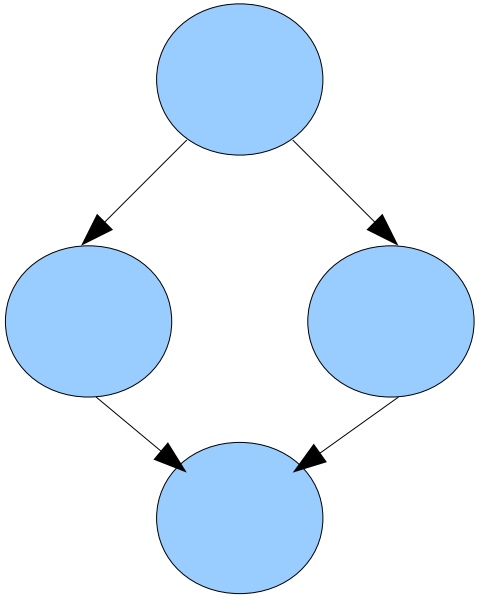
- Cada bloco de comandos tem as seguintes características
  - Não existe desvio de execução dentro do bloco
  - Uma vez que o primeiro comando do bloco é executado, todos os comandos são

- No grafo G
  - há um único vértice de entrada, $s \in N$
  - há um único vértice de saída, $o \in N$
  - um "caminho" é uma sequência finita de vértices $(n_1, n_2, \ldots, n_k)$, $k \geq 2$, tal que existe um vértice de $n_1$ para $n_{i+1}$, para $i=1,2,\ldots,k-1$

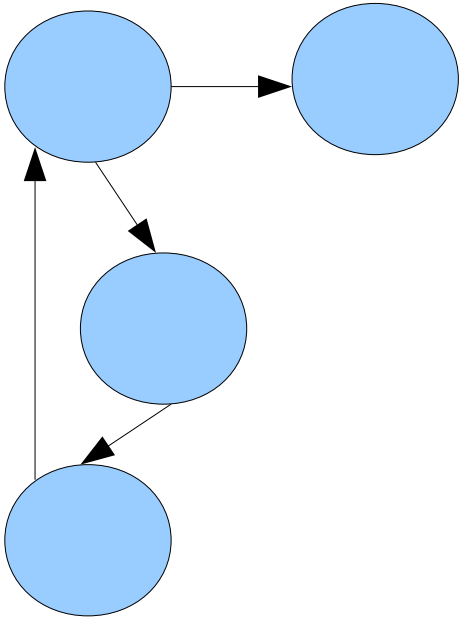
# Fluxo de Controle



Sequência

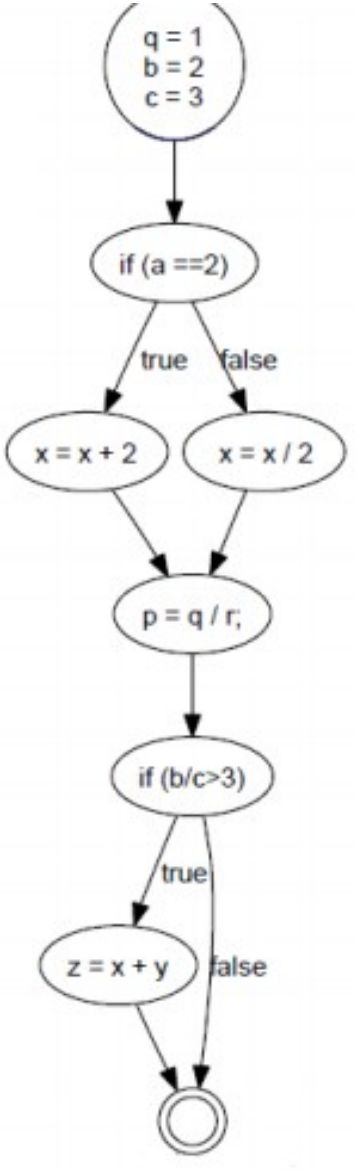


Condição/Decisão

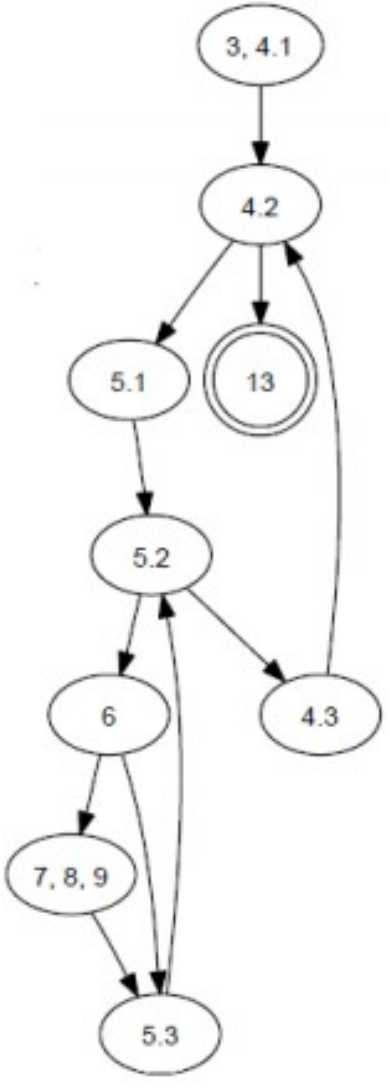


Repetição

```
1   q = 1;
2   b = 2;
3   c = 3;
4   if (a ==2) {
5       x = x + 2;
6   }   else {
7       x = x / 2;
8   }
9   p = q / r;
10  if (b/c>3) {
11      z = x + y;
12  }
```



```
2   public void bolha(int[] a, int size) {
3       int i, j, aux;
4       for (i = 0; i < size; i++) {
5           for (j = size - 1; j > i; j--) {
6               if (a[j - 1] > a[j]) {
7                   aux = a[j - 1];
8                   a[j - 1] = a[j];
9                   a[j] = aux;
10              }
11          }
12      }
13  }
```



# Casos de Teste para Cobertura

- Conjunto de casos de teste é dito adequado de acordo com um critério, se
  - Exercita todos os elementos definidos no critério
  - Ou seja, causa todos os elemento definidos no critério
- Conjunto de testes deve:
  - maximizar a cobertura
  - minimizar o tempo
- Grau de cobertura pode ser menor do que 100% devido a elementos inatingíveis

# Aplicação da Cobertura

- Tipos de Cobertura
  - Cobertura de código: Cobrir possíveis maneiras em que o código é executado
  - Cobertura de dados: Cobrir as possíveis combinações de dados

- Critérios de cobertura
  - Cobertura de comandos (*statements*)
  - Cobertura de decisões (*branches)*
  - Cobertura de condições
  - Cobertura de caminhos

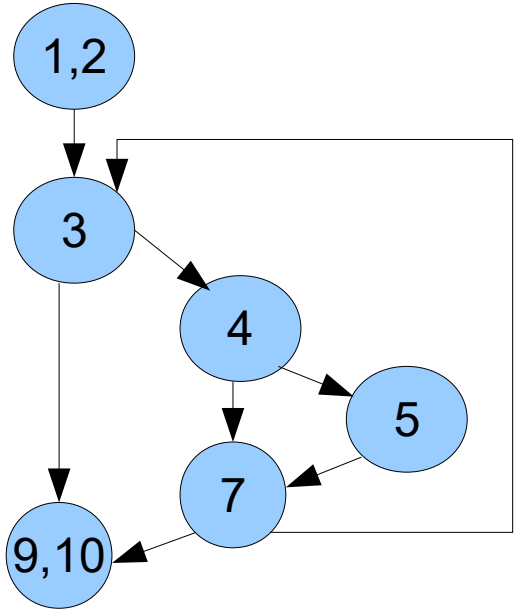
# Cobertura de Comandos

**Cada comando deve ser executado pelo menos uma vez**

- No grafo, garantir que cada vértice será exercitado pelo menos uma vez pelo conjunto de testes

```
1. int foo(int a, int b){
2.       int c=0;
3.       while(a<0){
4.            if (b<0){
5.                 b=b+2;
6.            }
7.            a=a+1;
8.       }
9.       c=a+b;
10.      return c;
11. }
```

| a=-1<br>b=-1 |
|---|
| c=1 |



# Limitações

- Não é capaz de detectar diversos tipos de defeito. Ex.:
  - if (a > 0) onde deveria ser if (a >= 0)
  - Executar um comando de repetição uma única vez
  - *"ifs"* sem "*elses*" são particularmente vulneráveis

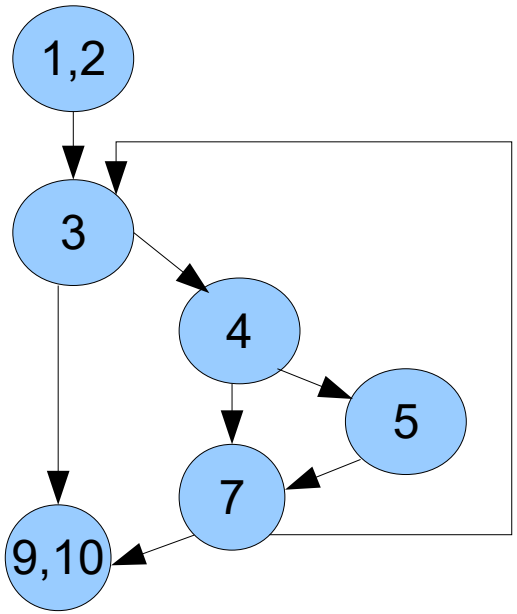
# Cobertura de Decisões/Desvios

**Cada ramo deve ser percorrido pelo menos uma vez**

- No grafo, garantir que todas as arestas serão executadas pelo menos uma vez pelo conjunto de testes

- Note que, um caso de teste pode ser suficiente para cobrir todos os comandos mas não todos os caminhos de diferente decisão

```
1. int foo(int a, int b){
2.       int c=0;
3.       while(a<0){
4.            if (b<0){
5.                 b=b+2;
6.            }
7.            a=a+1;
8.       }
9.       c=a+b;
10.      return c;
11. }
```



# Cobertura de Condições

**Todas as decisões devem ser avaliadas para valores verdadeiros e falsos em cada decisão**

- Não requer testar todas as possibilidades
- Desvios podem não estar totalmente cobertos
- Pode-se combiná-la com a cobertura de desvios

# Cobertura de Caminhos

**Deve-se seguir todos os caminhos pelo menos uma vez incluindo repetições**

- Deve-se cobrir todos os caminhos possíveis, incluindo repetições, da entrada até a saída
- Caminhos:
  - Podem chegar a $2^n$ para n condições
  - Comandos de repetição: o número de caminhos é muito grande e pode ser infinito
- O custo pode ser proibitivo

# Atividade de Fixação

- O que é o Teste Estrutural de software?
- O que é um grafo de fluxo de controle de um programa? Como esse grafo é construído?
- Explique cada um dos critério de cobertura
  - Cobertura de comandos (*statements*)
  - Cobertura de decisões (*branches)*
  - Cobertura de condições
  - Cobertura de caminhos

# Referências


- Delamaro, M. E., Maldonado, J. C., & Jino, M. (2016). Introdução ao teste de software. Rio de Janeiro: Elsevier. (Capítulo 4)

- Myers, Glenford J. et al (2004) "The Art of Software Testing." 2ed. New York, NY, USA: John Wiley & Sons.