# Testes baseados em grafos: fluxo de controle

Criado: abr/2005

Última atualização: mar/2013

# Tópicos

- Terminologia
- Exemplos de grafos usados em testes
- Relações
- Uso de grafos em testes

# Referências

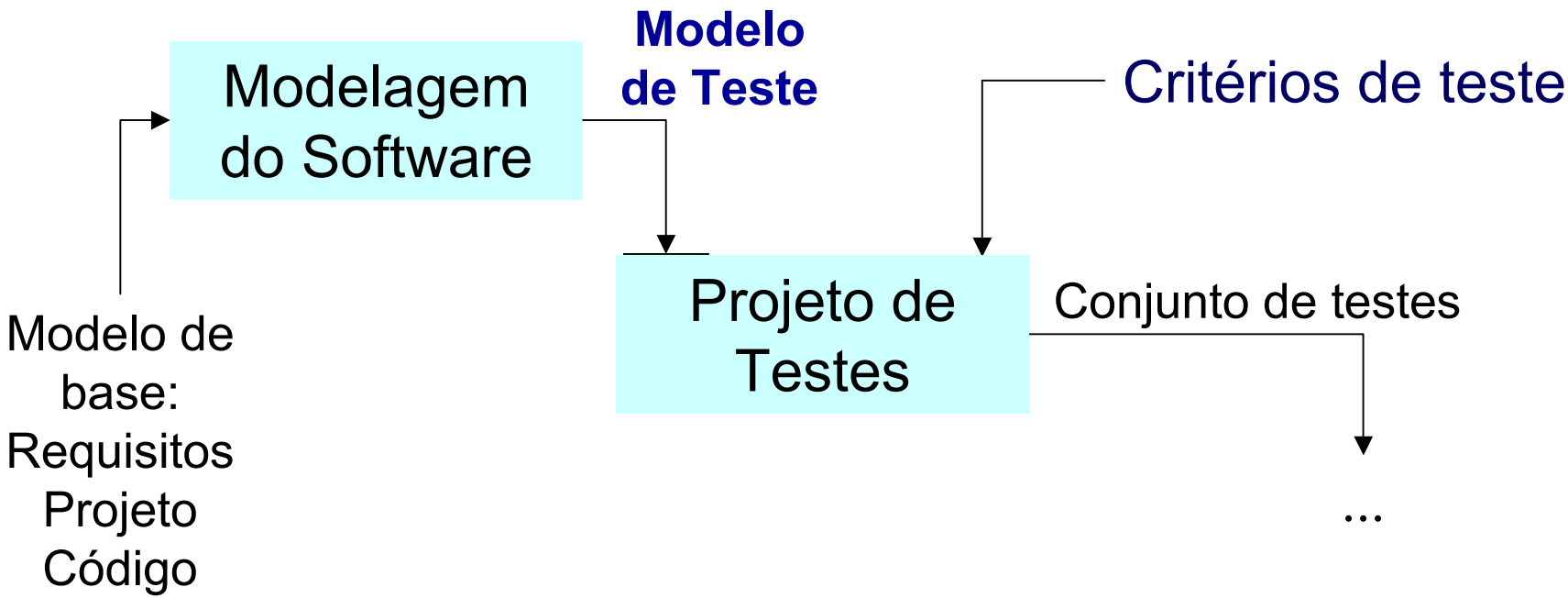
P. Amman, J. Offutt. *Introduction to Software Testing*, 2008, cap.2.

P.B.Menezes. Linguagens Formais e Autômatos, Editora Sagra Luzzato, 3a. Edição, 2000, c.1.

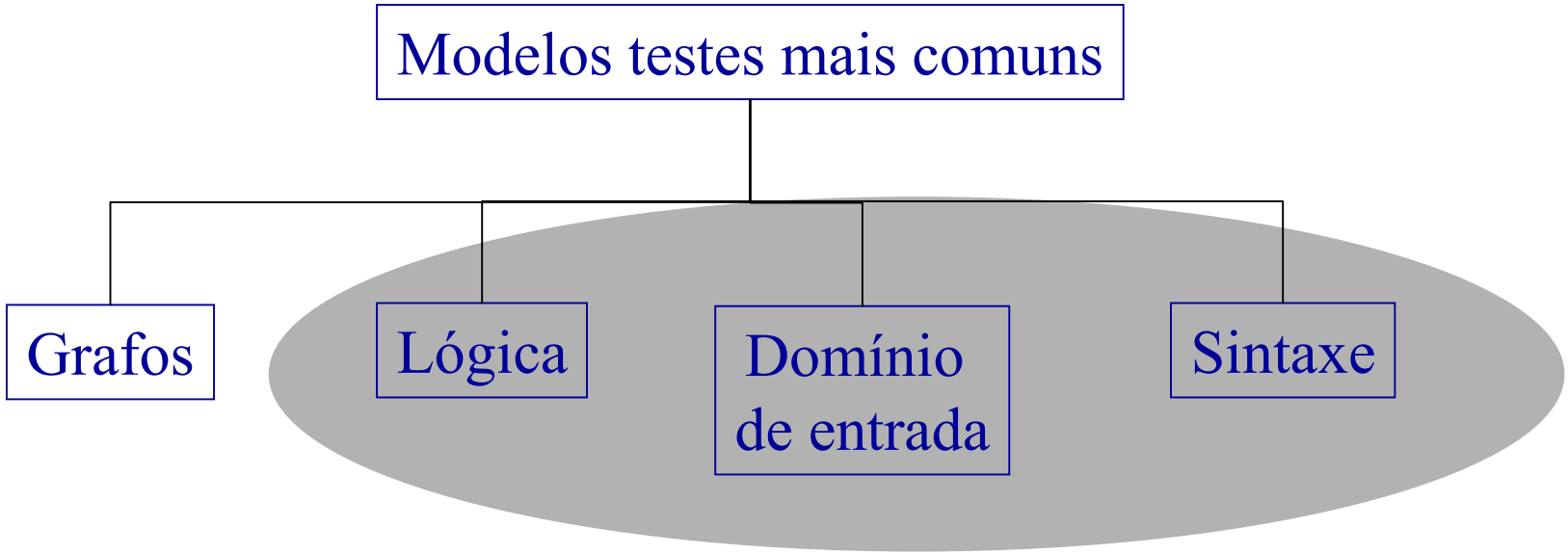
B.Beizer. *Black-Box Testing*. John Wiley, 1995. cap. 2

R.Binder. *Testing Object-Oriented Systems. Models, Patterns and Tools.* Addison Wesley, 2000. c. 8.11

# Projeto de testes



# Modelos de Testes

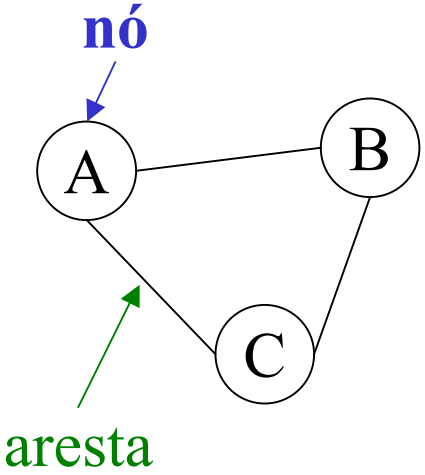


# Grafos

- Estrutura muito comum em testes
- Pode ser obtido a partir de diferentes modelos de base:
  - Casos de uso
  - Diagramas da UML: estados, de atividades, de colaboração, de classes, de componentes
  - Máquinas finitas de estados (clássicas e variações)
  - Código
  - ...
- Geração de testes ➲ percorrer o grafo
  - Existem inúmeros algoritmos na literatura

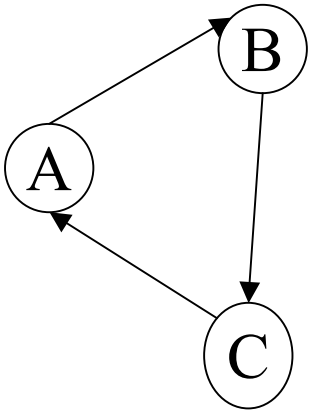
# Terminologia - 1

- Grafo
  - Representa relações entre entidades
  - Relação: associação de interesse entre entidades
    - **a R b** → a está relacioando com a
    - ex.:
      - **a** chama **b**
      - **a** calcula valor que é usado em **b**
      - a ação **a** é seguida da ação **b**
  - Elementos:
    - **Nós** (ou vértices): representam as entidades
    - **Arestas**: representam as relações entre as entidades



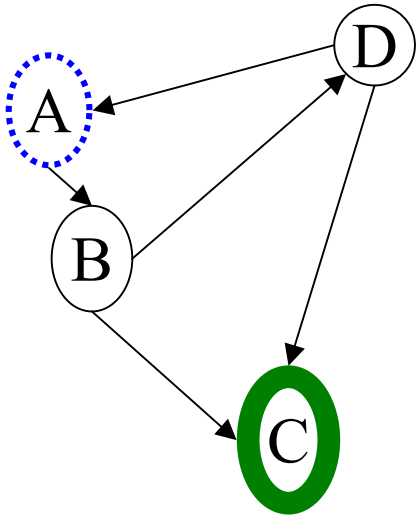
# Terminologia -2

- As arestas de um grafo podem ser direcionadas, indicando que elas só podem ser atravessadas em uma única direção.
- Um grafo que só contém arestas direcionadas (também chamadas de **arcos**) é chamado de **grafo direcionado** ou **digrafo**.
- A maioria dos grafos usados em testes são digrafos.



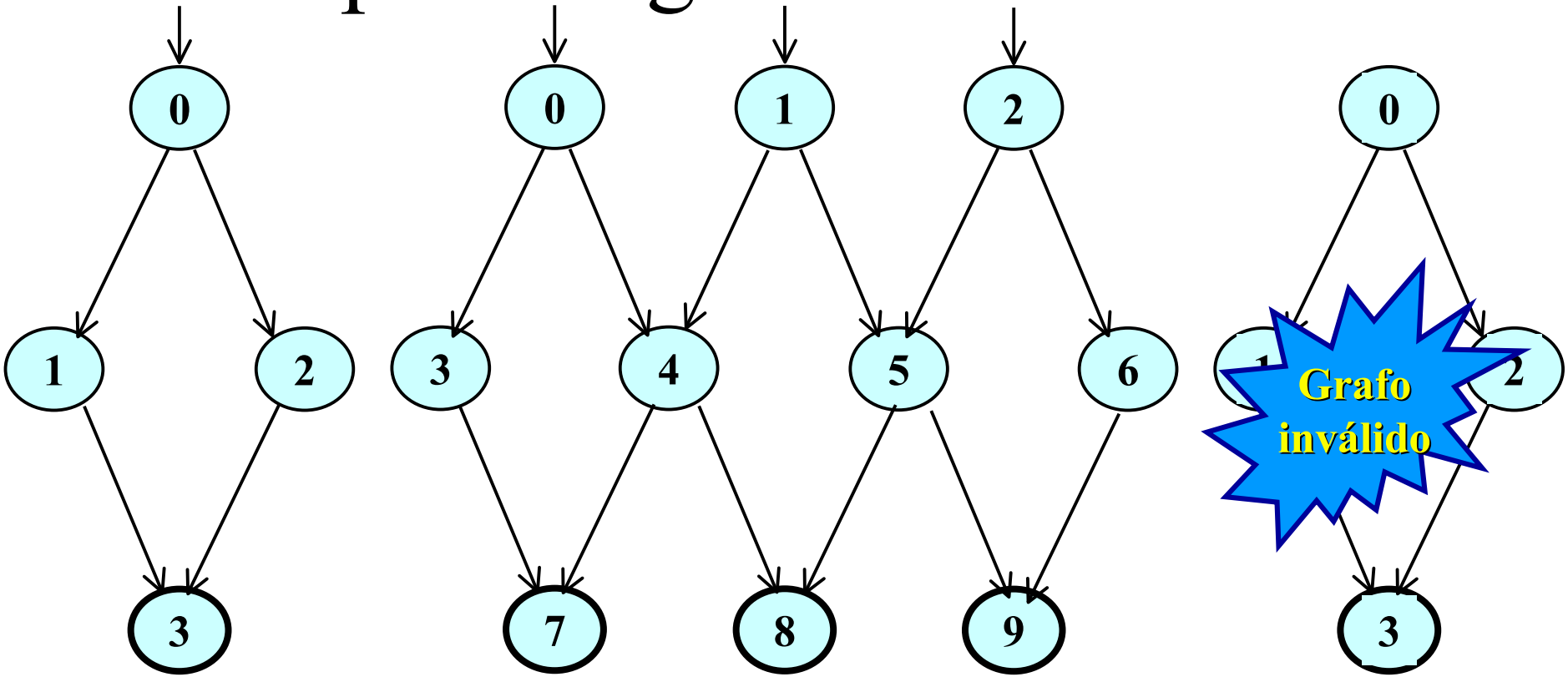
# Terminologia - 3

- Em testes, é interessante definir dois tipos especiais de nós:
  - **Nó de entrada** (**inicial**): representa um ponto de entrada do programa ou sistema. Pode ter 1 ou mais nós de entrada.
  - **Nó de saída** (**terminal**): um nó que não tem arcos saindo dele. Representa um ponto de saída de um programa ou sistema.
    Um grafo pode conter zero ou mais nós de saída.



# Exemplos de grafos

[Ammann, Offutt 2008]




$N_0 = \{ 0 \}$ $N_f = \{ 3 \}$

$N_0 = \{ 0, 1, 2 \}$ $N_f = \{ 7, 8, 9 \}$

$N_0 = \{ \}$ $N_f = \{ 3 \}$

# Uso de grafos em testes

- Para usar grafos para projetar casos de teste deve-se proceder de acordo com os seguintes passos:
  1. Defina o grafo
  2. Defina o tipo da relação
  3. Percorra o grafo para atender a critérios selecionados
     (de preferência, usando um algoritmo de percurso em grafos)

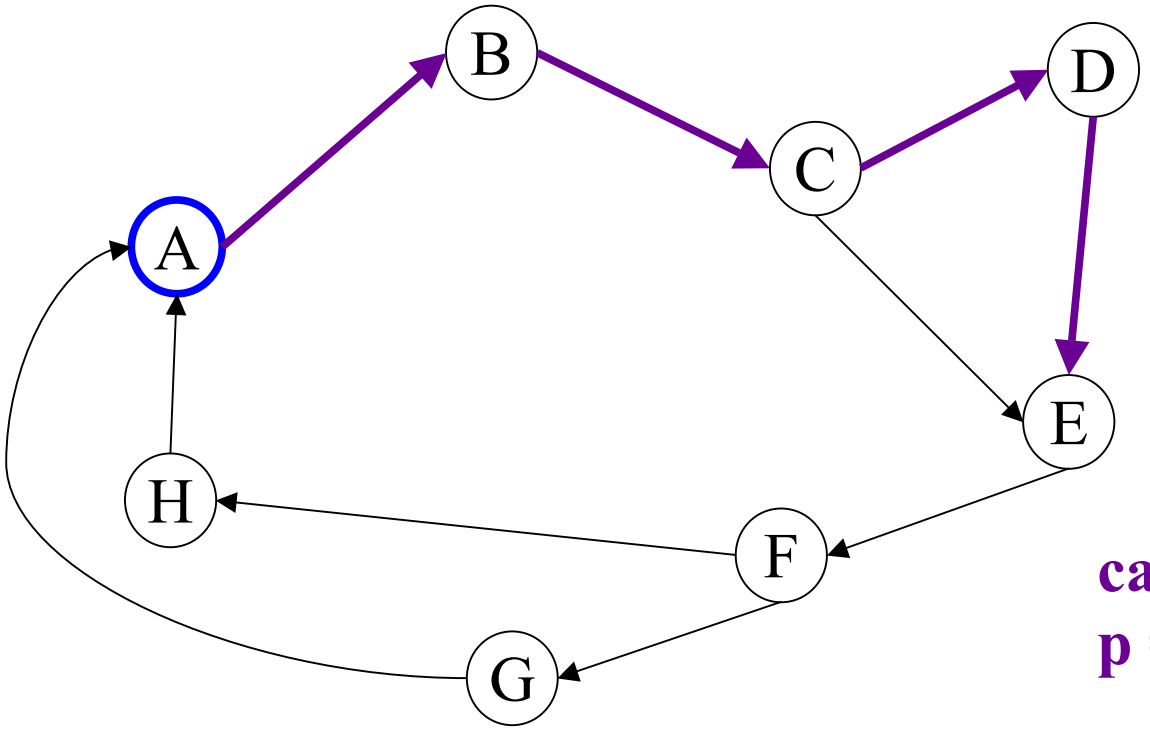
**ou seja, ao ver um grafo, o testador deve …**

**cobri-lo**

# Definições úteis para os testes

- Caminho: seqüência de arestas partindo de um nó origem a um nó destino
- Caminho completo: caminho de um nó de entrada a um nó de saída
- Comprimento de um caminho: número de nós ou número de arestas de um caminho
- Ciclo (ou laço): um caminho em que um nó (ou aresta) é visitado mais de uma vez
- Caminho livre de ciclos: um caminho em que nenhum nó (ou aresta) é visitado mais de uma vez

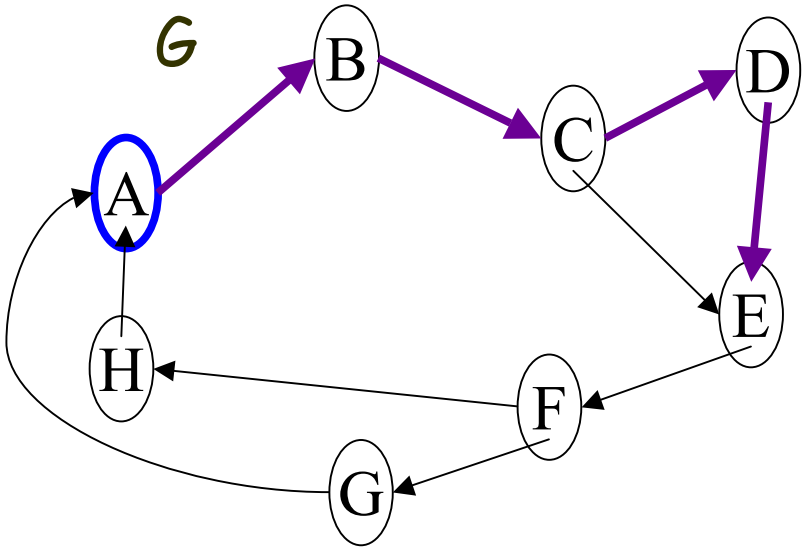
# Exemplo




**caminho de A a E:**
**p = [A, B, C, D, E]**

# Mais definições

- Subcaminho: subseqüência de nós em um caminho
- Alcance de um nó – *A*(n): subgrafo que pode ser alcançado a partir de n




p = [A, B, C, D, E]
q = [B, C, D] subcaminho de p
*A*({B, C}) = *G*

# Caminho de teste

- Caminho de teste:
  - caminho que começa em um nó inicial e termina em um nó final
  - Representam a execução de um caso de teste:
    - Alguns caminhos de teste podem ser executados por mais de um caso de teste
    - Alguns caminhos de teste não podem ser executados por nenhum caso de teste
- Grafos única entrada- única saída ou SESE (Single-Entry, Single-Exit):
  - Conjunto de nós iniciais e finais contém exatamente um nó cada.

# Visita e percurso

- Visita:
  - Um caminho de teste p **visita** um nó n se $n \in p$
  - Um caminho de teste p **visita** uma aresta e se $e \in p$

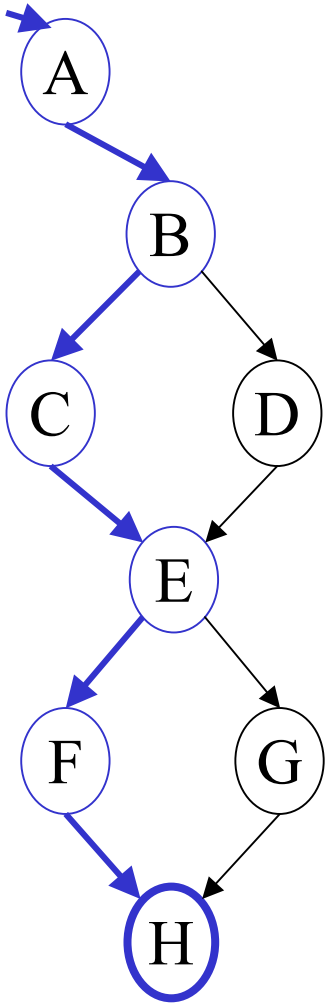
- Percurso (*tour*):
  - Um caminho de teste p **percorre** um subcaminho q se $q \in p$

# Visita e percurso

- Visita:
  - Um caminho de teste p **visita** um nó n se n ∈ p
  - Um caminho de teste p **visita** uma aresta e se e ∈ p

- Percurso (*tour*):
  - Um caminho de teste p **percorre** um subcaminho q se q ∈ p

P = [A, B, C, E, F, H]
Visita nós A, B, C, E, F, H
Visita arestas (A, B), (B, C), (C, E), (E, F), (F, H)
Percorre subcaminhos (A, B, C), (B, C, E), (C, E, F), (E, F, H), (B, C, E, F), …



# Testes e Caminhos de Testes

- p(t): caminho de teste executado por um caso de teste t
- p(T): conjunto de caminhos de testes executado pelo conjunto de testes T
- Cada teste t executa um e somente um caminho de teste
  - Software determinista: um caso de teste executa sempre o mesmo caminho de teste
  - Software indeterminista: um caso de teste pode executar diferentes caminhos a cada nova execução

# Adequabilidade

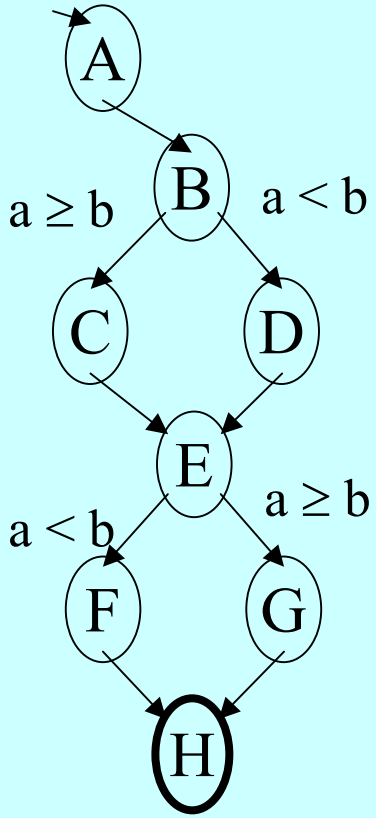
- Sejam:
  - C: critério de teste
  - $RT_C$: requisitos de teste para o critério C
  - T: conjunto de testes
  - G um grafo
  - p(T): conjunto de caminhos de teste em G executados por T

- Diz-se que T é **adequado** a C (ou **C-adequado**) sss $\forall rt \in RT_C \exists q \in p(T) \mid q$ exercita rt

# Alcançabilidade e caminho

- Um critério C pode requerer que um elemento do grafo (nó ou aresta) seja alcançado a partir de outro:
  - Um elemento `x` de G é sintaticamente alcançável a partir de um elemento `y` de G sss existe um subcaminho entre `x` e `y`
  - Um elemento `x` de G é semanticamente alcançável a partir de um elemento `y` de G sss existe um teste que pode executar o subcaminho de `x` a `y`.

# Exemplo




C: cobrir arestas (B, D) e (G, H)
$RT_C$: {(B, D), (G, H)}
(G,H) é sintaticamente alcançável a partir de (B, D):
Subcaminho: (B, D) → (D, E) → (E, G) → (G, H)

(G, H) é semanticamente alcançável a partir de (B, D)?

# Critérios de cobertura de grafos

- Critérios baseados no fluxo de controle:
  - Considera somente nós e arestas

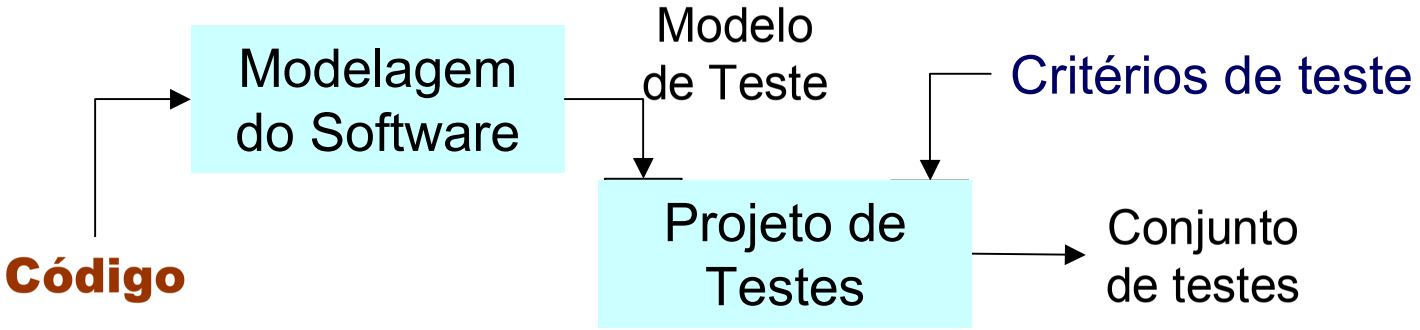
- Critérios baseados no fluxo de dados:
  - Requer que o grafo seja anotado com referências a variáveis

# Exercício

- Considere o seguinte grafo G(N, E):
  - N = {a, b, c}
  - $N_0$ = {a}
  - $N_f = \varnothing$
  - E = {(a, b), (a,c), (b,a), (b,c), (c,a)}
- Considere os seguintes caminhos candidatos:
  - $p_0$ = [a,b,c,a]
  - $p_1$ = [a,c, a,b,c]
  - $p_2$ = [a,b,c,a,b,a,c]
  - $p_3$ = [b,c, a,c]
  - $p_4$ = [a,b,c,b,c]
- Responda:
  - Quais dos caminhos dados são caminhos de teste?
  - Liste os oito requisitos de teste para o critério de cobertura de pares de arestas
  - Os caminhos de teste identificados anteriormente satistazem ao critério?

# Testes baseados na implementação (caixa branca)

# Testes caixa branca



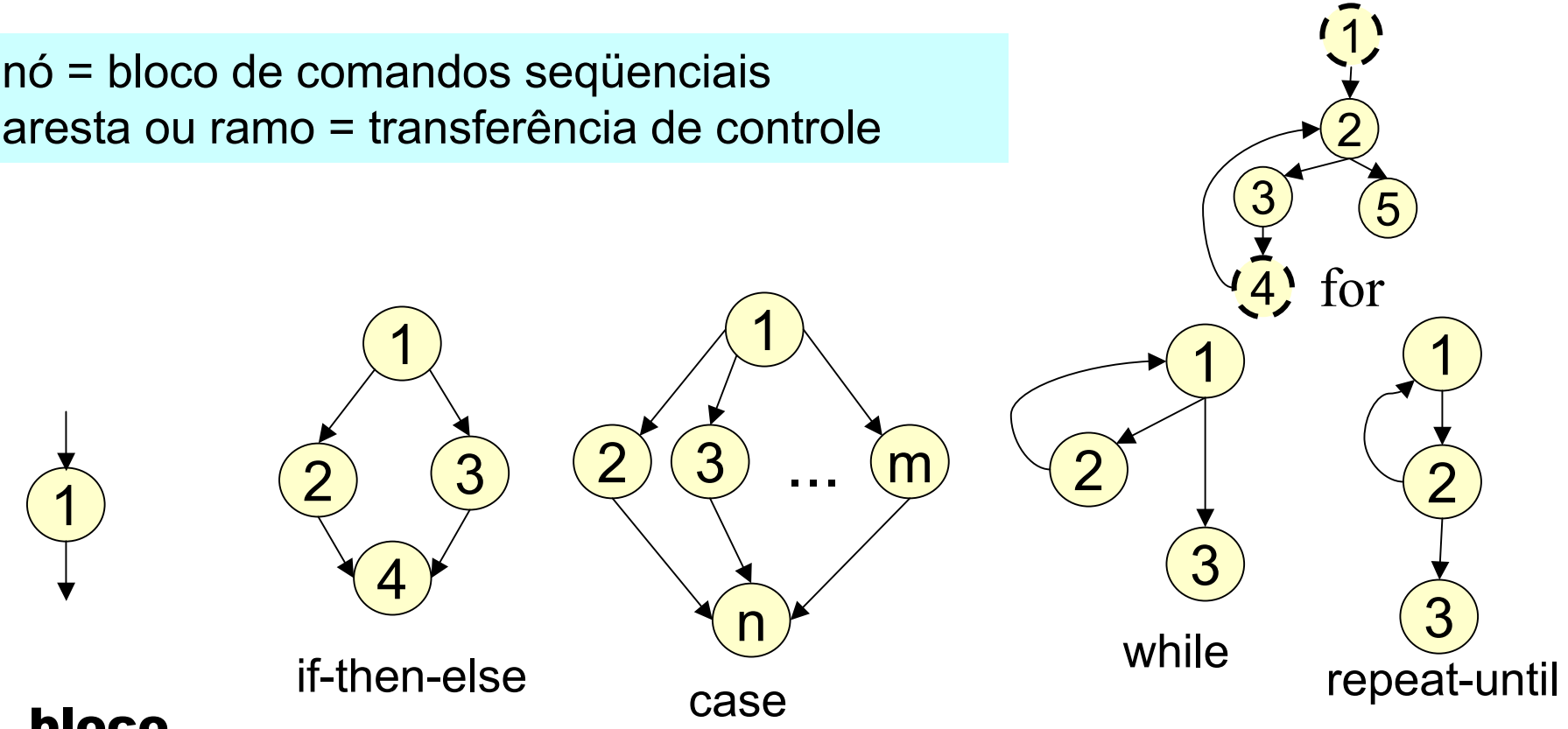

- Testes baseados na implementação ou testes caixa-branca:
  - visam exercitar estruturas de controle (instruções) e de dados de um programa

# Modelos de Teste

- Grafo de fluxo de controle
  - Representa a ordem na qual as instruções são executadas
- Grafo de fluxo de dados
  - Representa também as definições de variáveis e seus usos no programa

# Grafo de fluxo de controle

nó = bloco de comandos seqüenciais
aresta ou ramo = transferência de controle




**bloco seqüencial**

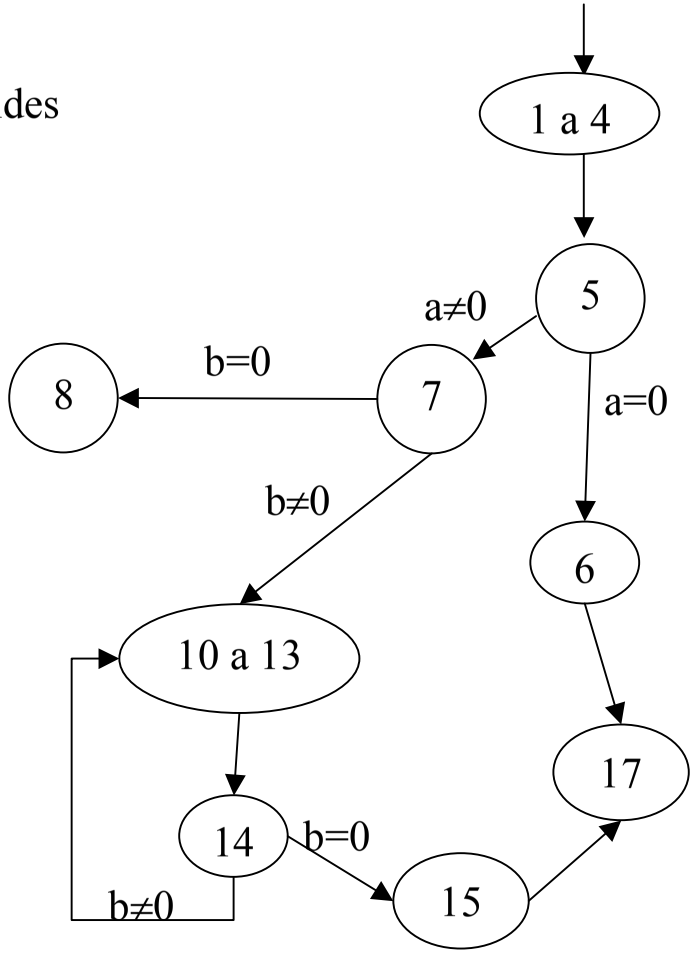
**seleção**

**repetição**

# Exemplo

Cálculo do mdc (a, b) baseado no algoritmo de Euclides

```
1.      function mdc (int a, int b) {
2.          int temp, value;
3.          a := abs(a);
4.          b := abs(b);
5.          if (a = 0) then
6.                value := b;    // b é o MDC
7.          else if (b = 0) then
8.               exceção;
9.          else
10.               repeat
11.                    temp := b;
12.                    b := a mod b;
13.                    a := temp;
14.               until (b  =  0)
15.               value := a;
16.         end if;
17.         return value;
18.     end mdc
```




Modelo de base → Modelo de teste

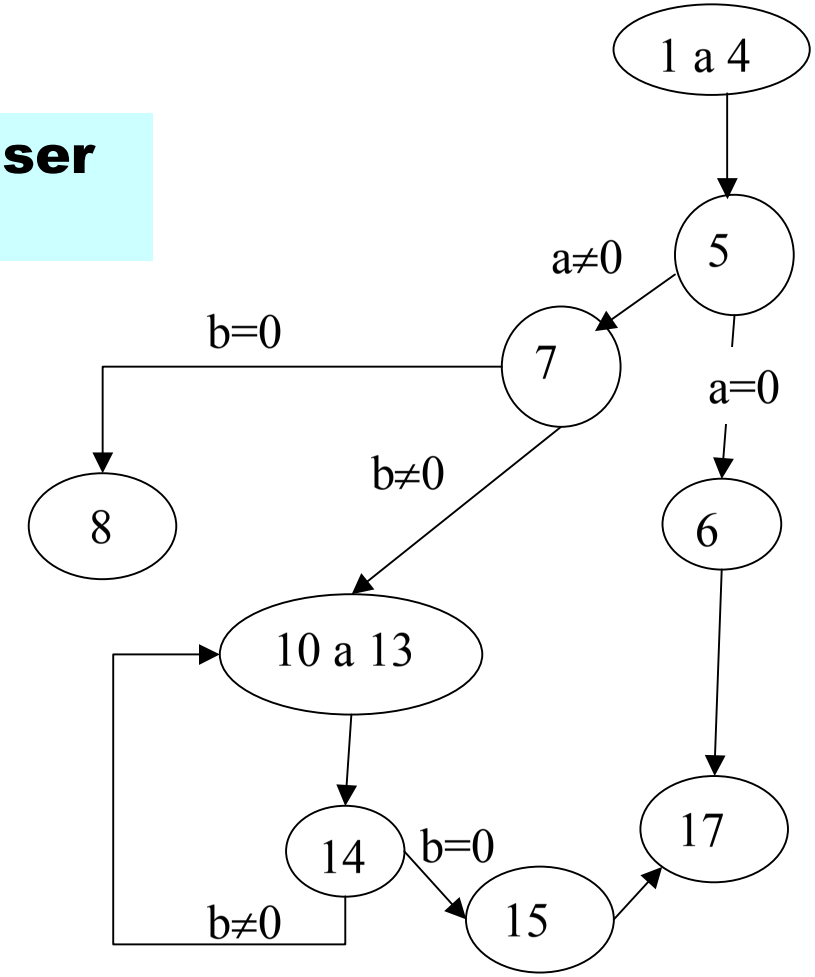
# Critérios de cobertura

- Cobertura de nós
- Cobertura de ramos
  - Cobertura de pares de ramos
  - Cobertura de condições
- Cobertura de caminhos
  - Cobertura de caminhos relevantes
  - Cobertura de caminhos básicos
- Cobertura de laços

# Teste de instruções

**Critério: cada instrução deve ser executada pelo menos 1 vez**

| nós | predicados | dados |
|---|---|---|

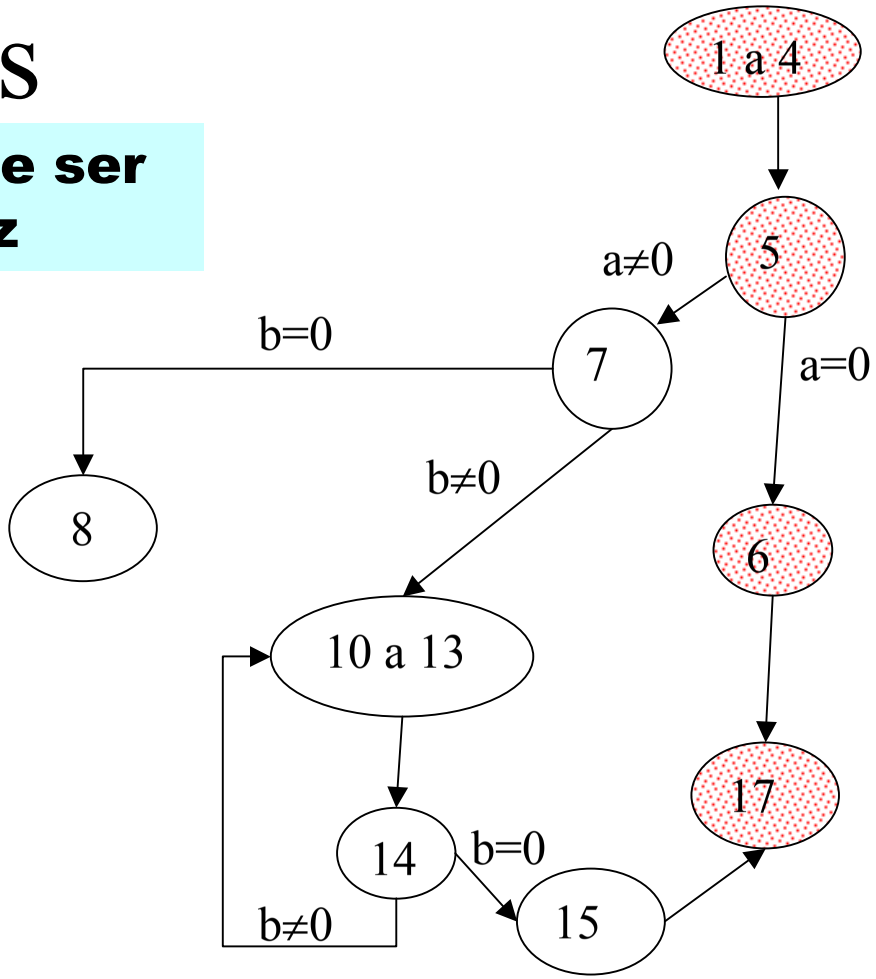



[Bruno Abreu 2006]

# Teste de instruções

**Critério: cada instrução deve ser executada pelo menos 1 vez**

| nós | predicados | dados |
|---|---|---|
| {1,5, 6, 16, 17} | a = 0, ∀b | (0, 3) |

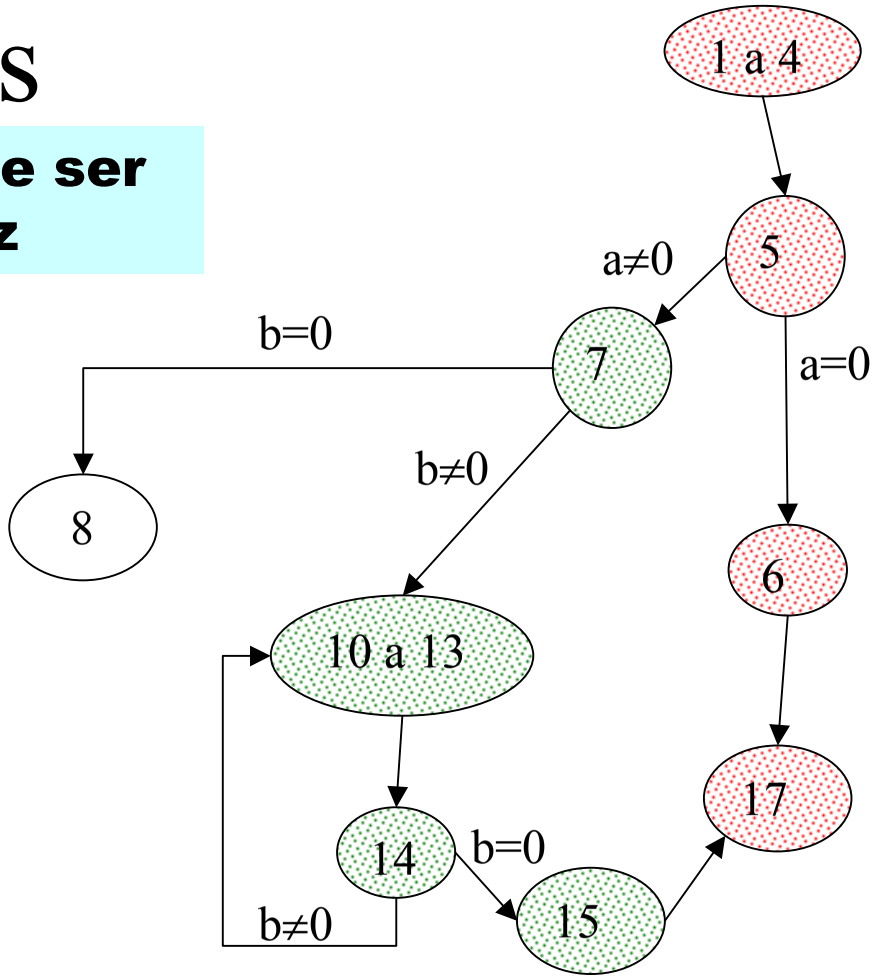



[Bruno Abreu 2006]

# Teste de instruções

**Critério: cada instrução deve ser executada pelo menos 1 vez**

| nós | predicados | dados |
|---|---|---|
| {1,5, 6, 16, 17} | a = 0, ∀b | (0, 3) |
| {1,5, 7, 9, 10, 14 15, 16, 17} | a ≠0, b ≠ 0 | (4, -2) |

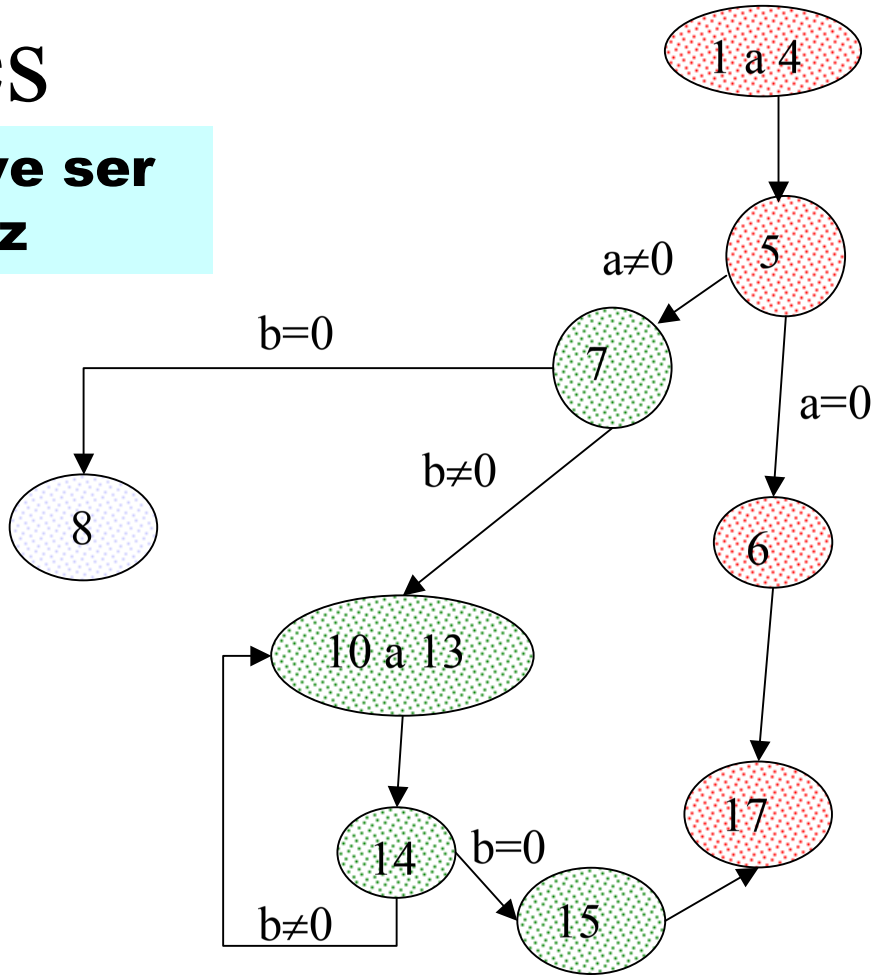



[Bruno Abreu 2006]

# Teste de instruções

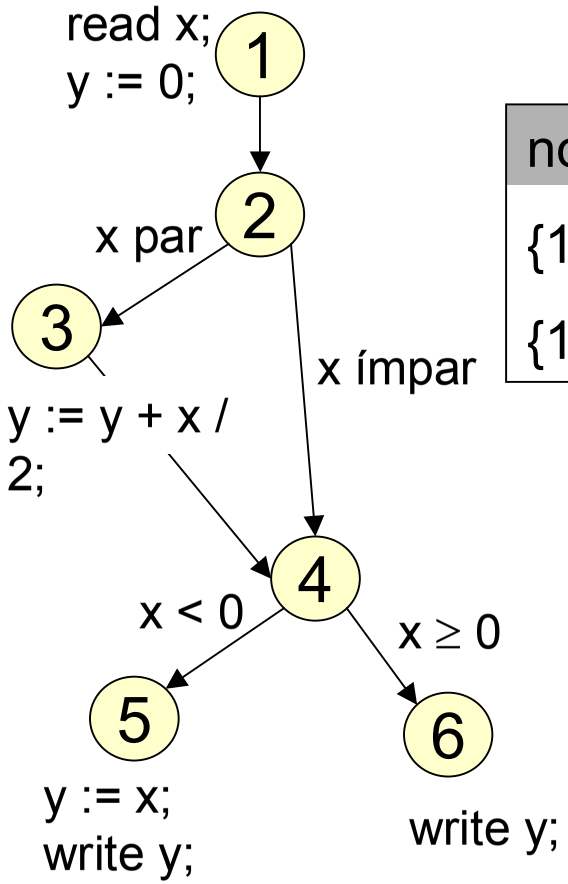
**Critério: cada instrução deve ser executada pelo menos 1 vez**

| nós | predicados | dados |
|---|---|---|
| {1,5, 6, 16, 17} | a = 0, ∀b | (0, 3) |
| {1,5, 7, 9, 10, 14 15, 16, 17} | a ≠0, b ≠ 0 | (4, -2) |
| {1, 5, 7, 8} | a ≠0, b = 0 | (4, 0) |



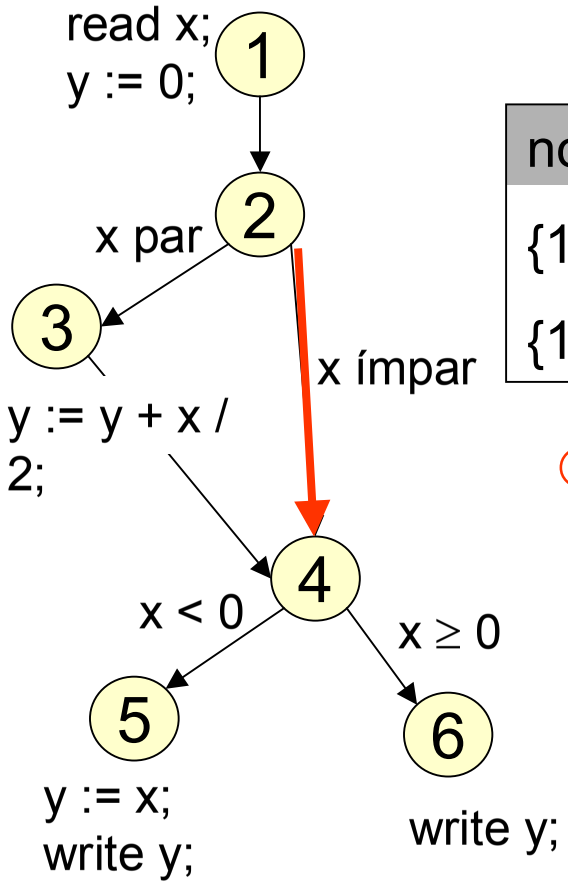

[Bruno Abreu 2006]

# Cobertura de nós




| nós | predicados | dados |
|---|---|---|
| {1,2,3,4,5} | x par negativo | -2 |
| {1,2,3,4,6} | x par positivo | 2 |

# Cobertura de nós



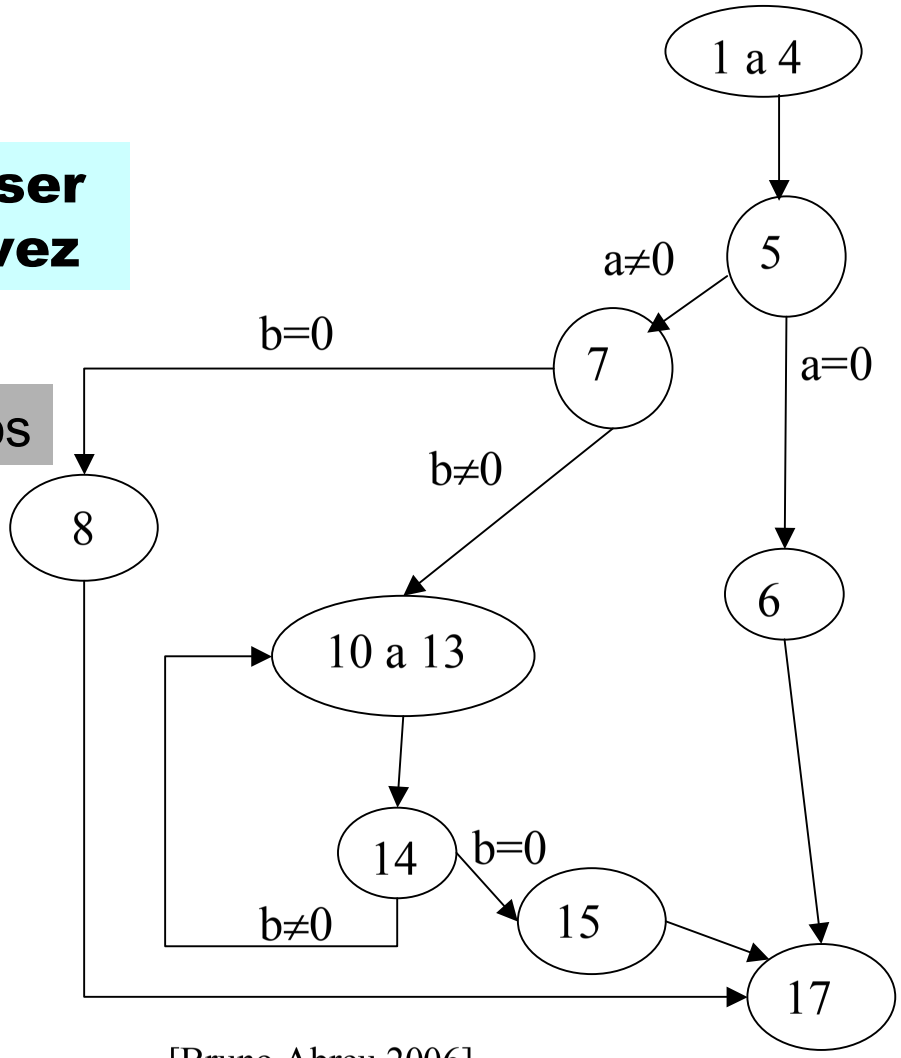

| nós | predicados | dados |
|---|---|---|
| {1,2,3,4,5} | x par negativo | -2 |
| {1,2,3,4,6} | x par positivo | 2 |

☹ note que o ramo 2 → 4 não é exercitado

# Teste de decisões

**Critério: cada ramo deve ser percorrido pelo menos 1 vez**

| ramos | predicados | dados |
|---|---|---|

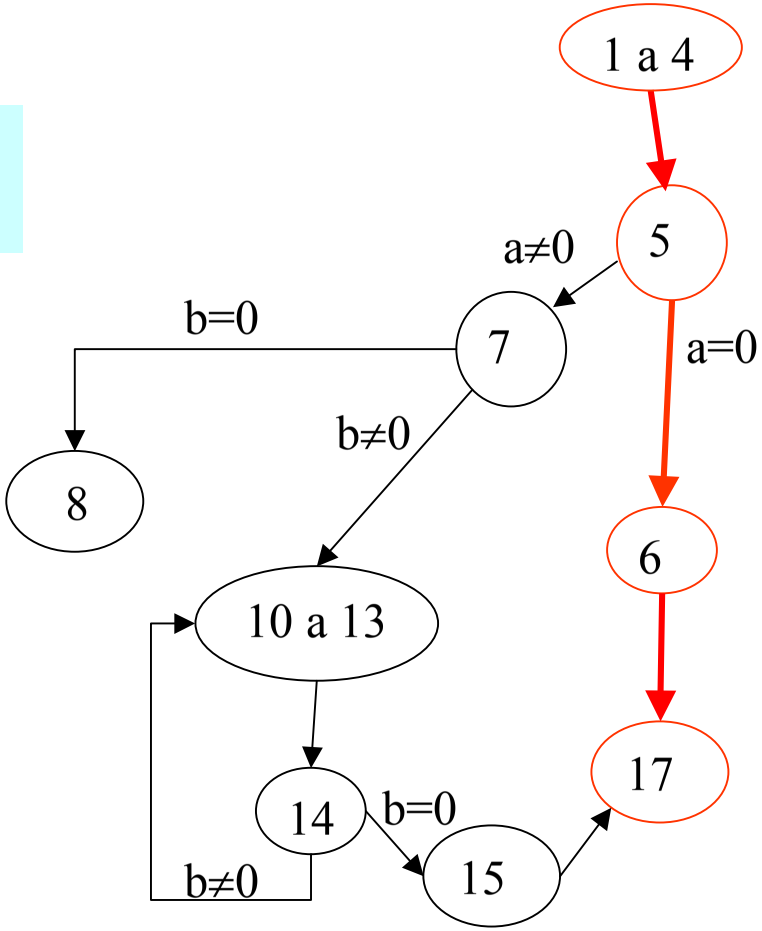



[Bruno Abreu 2006]

# Teste de decisões

**Critério: cada ramo deve ser percorrido pelo menos 1 vez**

| ramos | predicados | dados |
|---|---|---|
| {(1,5), (5,6), (6,17)} | a = 0, ∀b | (0, -1) |

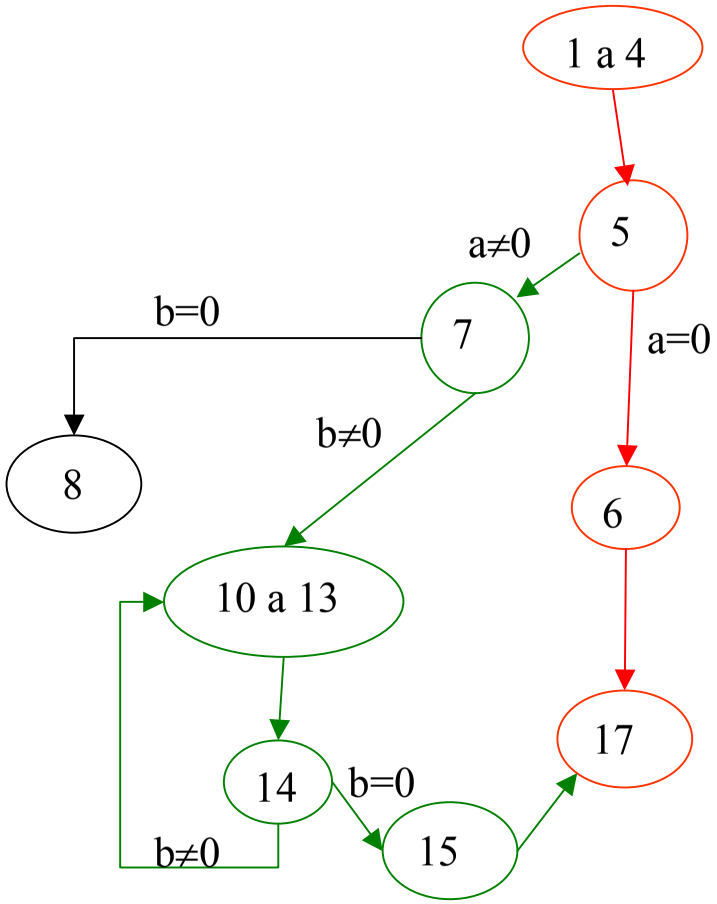



[Bruno Abreu 2006]

# Teste de decisões

**Critério: cada ramo deve ser percorrido pelo menos 1 vez**

| ramos | predicados | dados |
|---|---|---|
| {(1,5), (5,6), (6,17)} | a = 0, ∀b | (0, -1) |
| {(1,5), (5,7), (7,10), (10,14), (14,10), (14,15), (15,17)} | a ≠0, b≠0 e b>a | (4, 6) |

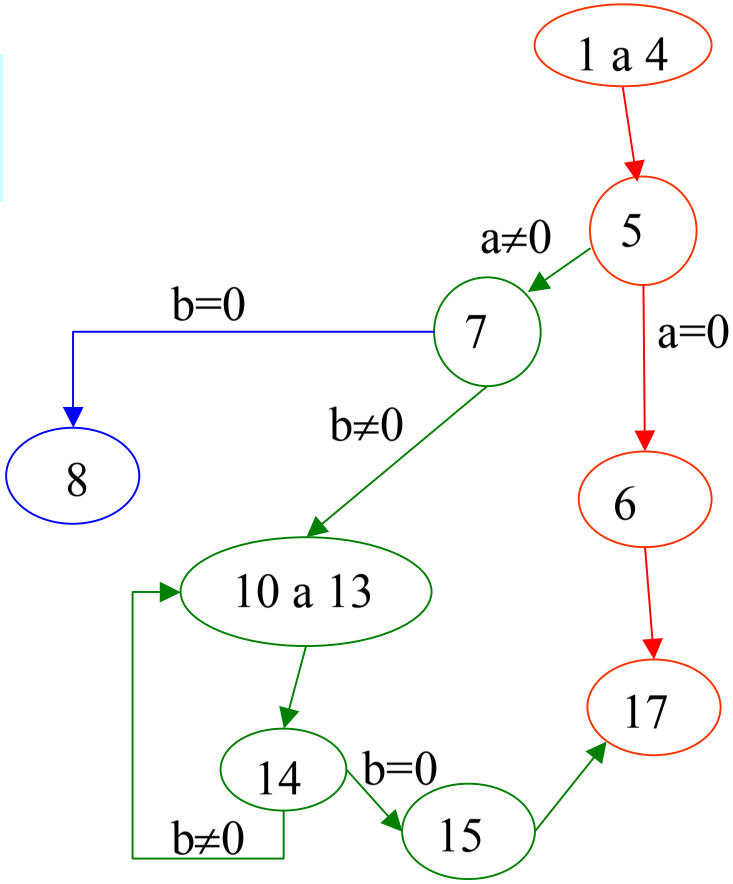



[Bruno Abreu 2006]

# Teste de decisões

**Critério: cada ramo deve ser percorrido pelo menos 1 vez**

| ramos | predicados | dados |
|---|---|---|
| {(1,5), (5,6), (6,17), | a = 0, ∀b | (0, -1) |
| {(1,5), (5,7), (7,10), (10,14), (14,10), (14,15), (16,17)} | a ≠0, b≠0 e b>a | (4, 6) |
| {(1,5), (5,7), (7,8)} | a ≠0, b=0 | (4, 0) |




[Bruno Abreu 2006]

# O que fazer com decisões compostas?

```
if (a > 0 && c < 3) {x = a + 1;}
if (b ==2 || d ≥ 4) {y = d - 2;}
```

- Cobertura de condições:
  - Critério: **cada condição deve ser avaliada uma vez como V e F**
    - a>0, c<3, b=2, d≥4 ⇦ todas as condições = V
    - a≤0, c≥3, b≠2, d<4 ⇦ todas as condições = F
  - Mas o que acontece se:
    - a≤0, c<3, b≠2, d ≥ 4 ou
    - a>0, c ≥ 3, b=2, d<4 ou
    - ...

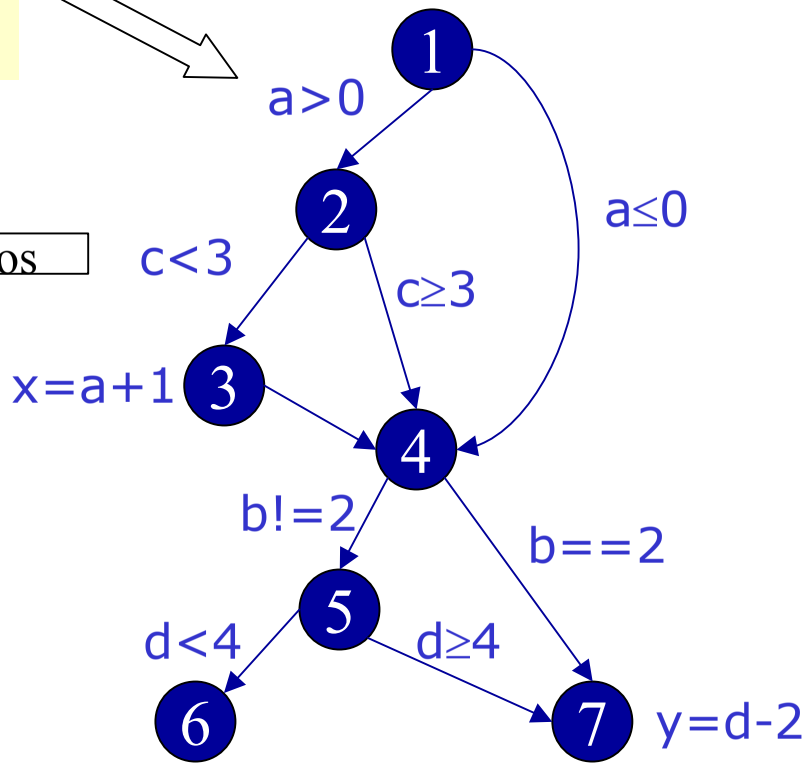
☞ Como cobrir todas as combinações interessantes?

# Testes de múltiplas condições

```
if (a > 0 && c < 3) {x = a+1;}
if (b ==2 || d ≥ 4) {y = d-2;}
```

Cobertura de ramos

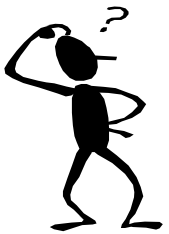
a>0, c<3, b=2, d≥4
a ≤ 0, c<3, b=2, d≥4
a>0, c≥3, b ≠ 2, d<4
a ≤ 0, c≥3, b ≠ 2, d≥4



# Cobertura de caminhos

- Caminho: seqüência de arestas partindo de um nó origem a um nó destino
- Caminho completo: caminho de um nó de entrada a um nó de saída
- Ciclo (ou laço): um caminho em que um nó (ou aresta) é visitado mais de uma vez
- Caminho livre de ciclos: um caminho em que nenhum nó (ou aresta) é visitado mais de uma vez

**Critério: todos os caminhos possíveis do grafo devem ser percorridos pelo menos 1 vez**




Esse critério é sempre factível?

# Cobertura de caminhos

**Critério: todos os caminhos possíveis do grafo devem ser percorridos pelo menos 1 vez**

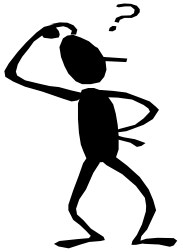
- Dificuldade 1: Grafos com ciclos
  - nº de caminhos pode ser indeterminado ou infinito

    ⇩
  - necessidade de critérios que permitam limitar o nº de caminhos:
    - Cobertura de caminhos relevantes
    - Cobertura de caminhos básicos
    - Cobertura de laços

# Cobertura de caminhos básicos

- Critério proposto McCabe
- Em vez de todos os caminhos, busca os caminhos linearmente independentes:
  - **Caminho linearmente independente (L.I.)**: dois caminhos são independentes quando nenhum contém nós internos do outro

    ⇓
  - Um caminho L.I. contém pelo menos 1 nova aresta não coberta pelos demais



- É possível calcular o nº de caminhos L.I?

# Número de caminhos L.I.

- O nº de caminhos independentes é dado pela complexidade ciclomática (V(G)) :

V(G) = nº de nós predicados + 1

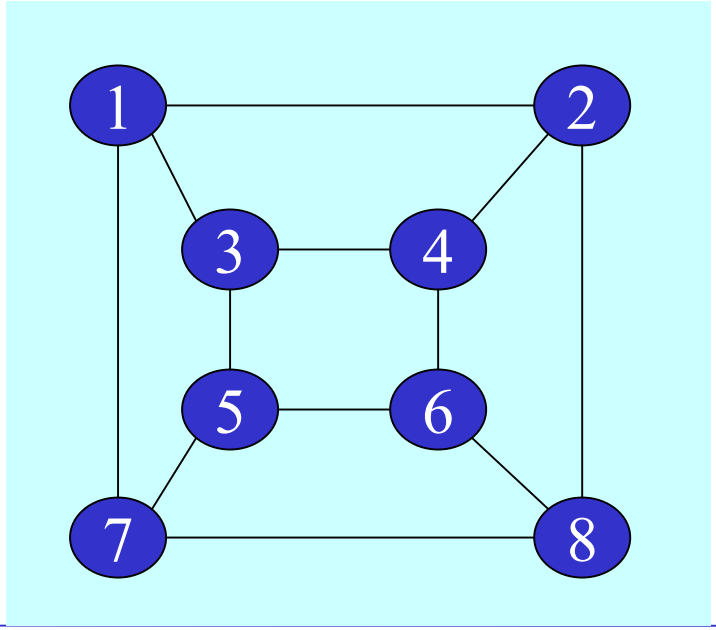
V(G) = nº de arestas – nº de nós + 2

V(G) = nº de regiões do grafo

- É um **limite máximo** do nº de testes necessários para cobrir todas as instruções de um programa
- Uma vez que todos os outros caminhos do grafo são combinações dos caminhos independentes ⇒ V(G) é o **limite inferior** do nº de testes de caminhos

# De onde vem V(G) ? (1)

- Euler (séc. XVIII) percebeu que um grafo simples, planar (desenhado em um plano) e conexo divide o plano em um certo nº de regiões totalmente fechadas + a região exterior, que é infinita

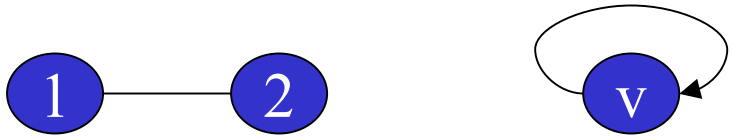



Só tem intersecção entre 2 arestas nos vértices inicial e final

Qtas regiões tem esse grafo?

# De onde vem V(G)? (2)

- Foi estabelecida então uma relação entre o número de arestas (a), o número de vértices (v) e o número de regiões (r) para tais grafos que é dada pela fórmula:
  - **r + n = a + 2** → r = a – n + 2 ← conhecida como fórmula de Euler para grafos planares
  - Se n ≥ 3 então a ≤ 3n – 6




☞ r é o limite máximo de caminhos L.I.

# Combinação linear de vetores

Fonte: http://www.fem.unicamp.br/~em421/semII-1999/textos/vetor.pdf

- Um conjunto de vetores $\{v_i\} = \{v_1, v_2, ..., v_n\}$ é linearmente independente se a combinação linear:

$$\sum_{i=1}^{n} \alpha_i v_i = 0$$

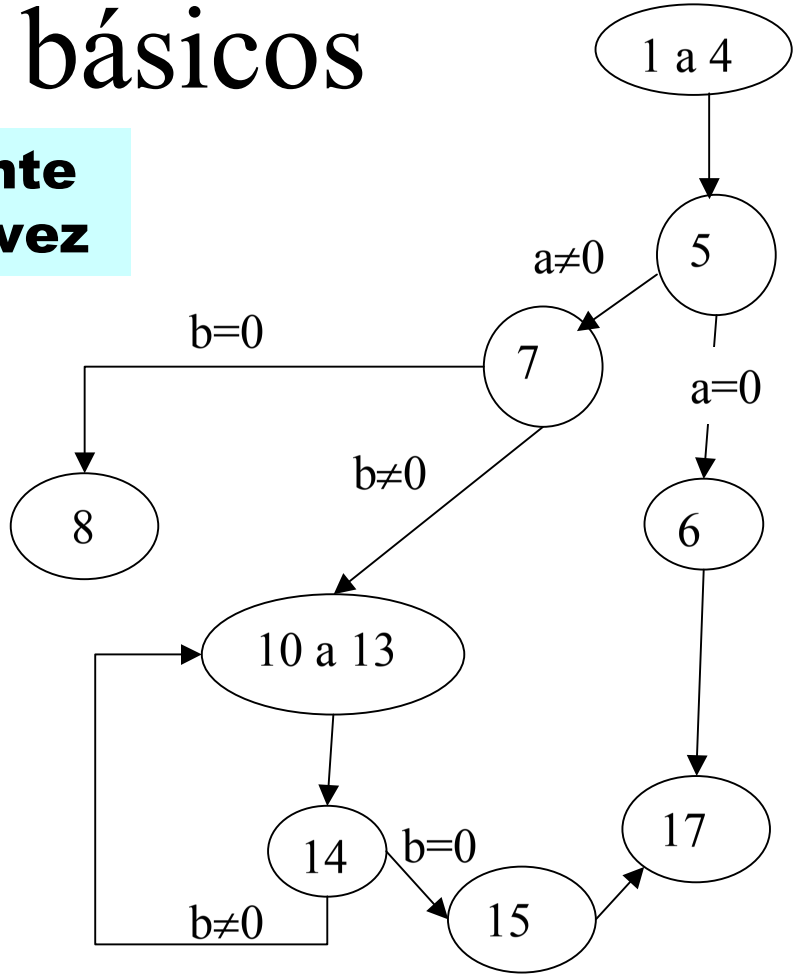
é válida somente se $\alpha_1 = \alpha_2 = ... = \alpha_n = 0$

Senão, se existir algum $\alpha_i \neq 0$ então os vetores são linearmente dependentes

# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

caminhos predicados dados

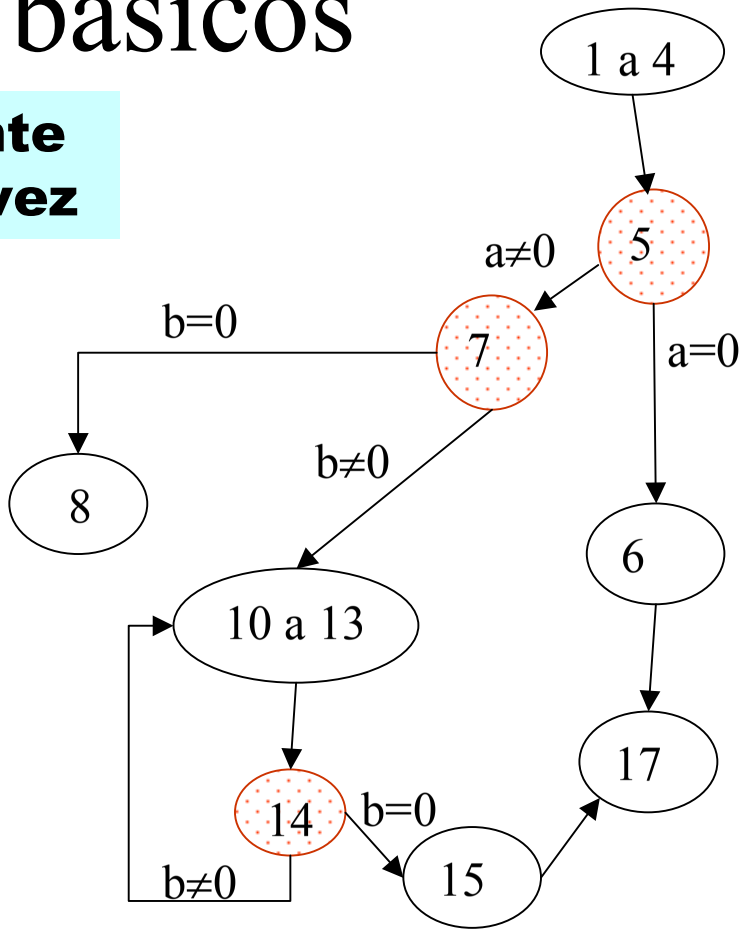


# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

**V(G) = nº condições + 1 = 4**

caminhos predicados dados

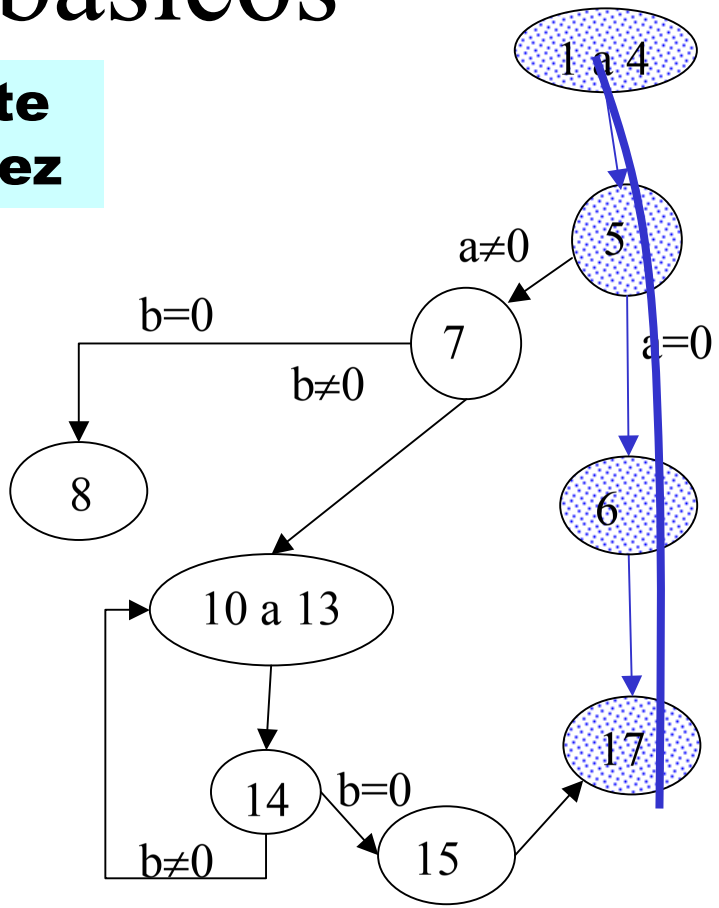


# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

**V(G) = nº condições + 1 = 4**

| caminhos | predicados | dados |
|---|---|---|
| {1-5-6-17} | a = 0, ∀b | (0, -2) |

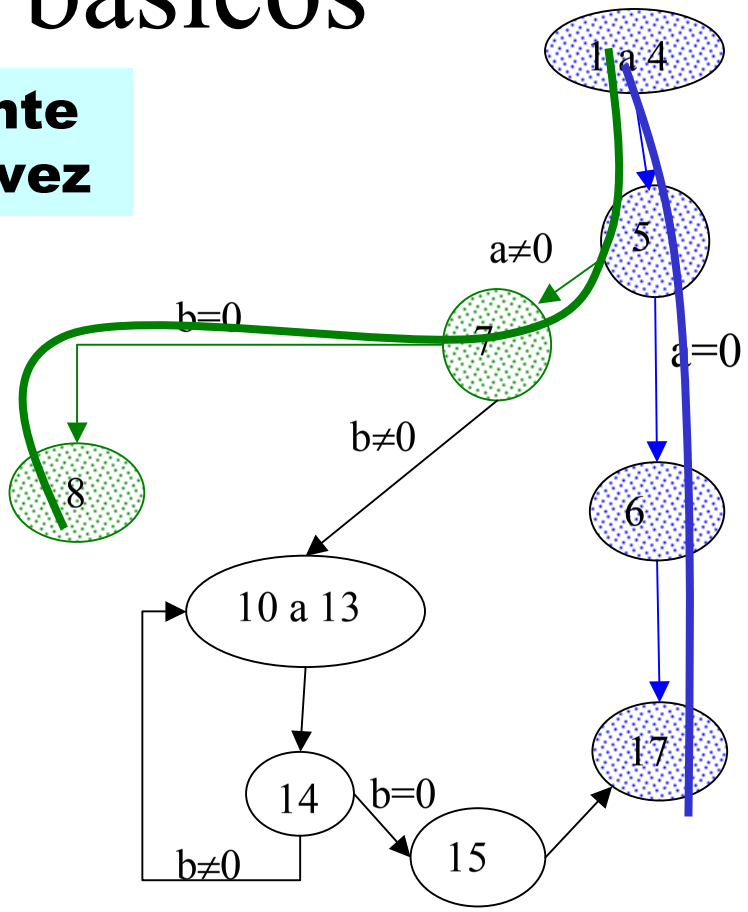


# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

**V(G) = nº condições + 1 = 4**

| caminhos | predicados | dados |
|---|---|---|
| {1-5-6-17} | a = 0, ∀b | (0, -2) |
| {1-5-7-8} | a ≠0, b=0 | (4, 0) |

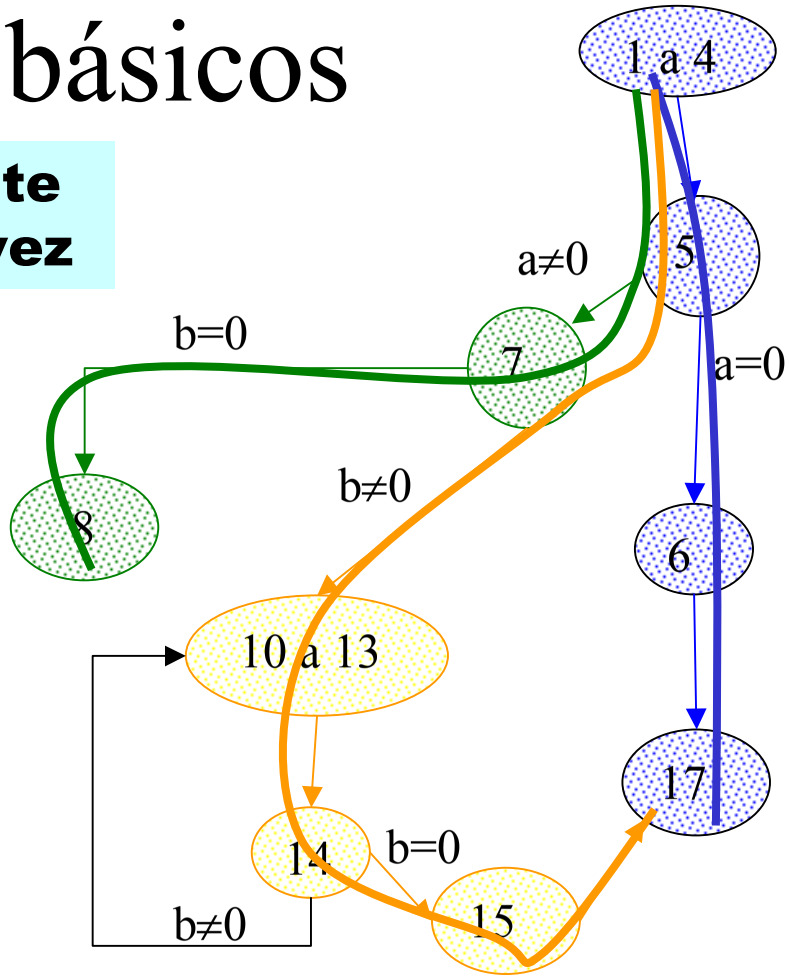


# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

**V(G) = nº condições + 1 = 4**

| caminhos | predicados | dados |
|---|---|---|
| {1-5-6-17} | a = 0, ∀b | (0, -2) |
| {1-5-7-8} | a ≠0, b=0 | (4, 0) |
| {1-5-7-10-14 | a ≠0, b≠0, | |
| 15-17} | a mod b = 0 | (8, 4) |

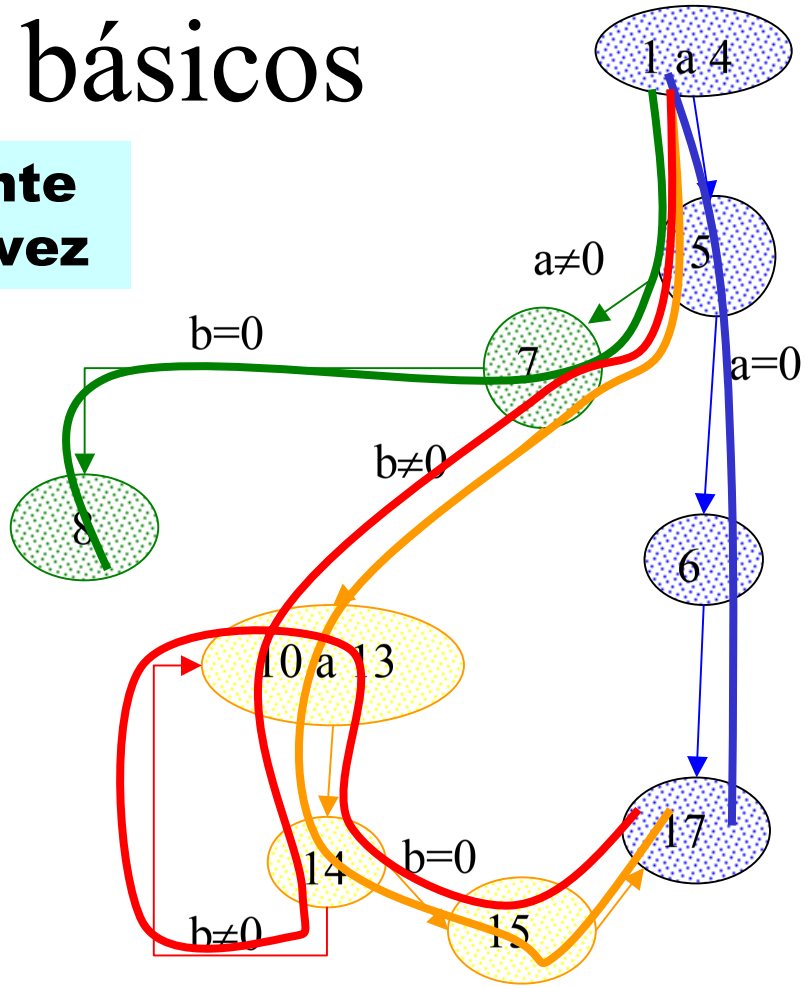


# Testes de caminhos básicos

**Critério: cada caminho independente deve ser percorrido pelo menos 1 vez**

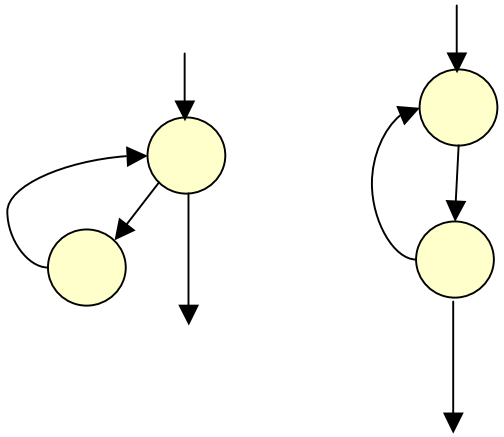
**V(G) = nº condições + 1 = 4**

| caminhos | predicados | dados |
|---|---|---|
| {1-5-6-17} | a = 0, ∀b | (0, -2) |
| {1-5-7-8} | a ≠0, b=0 | (4, 0) |
| {1-5-7-10-14-15-17} | a ≠0, b≠0<br>a mod b = 0 | (8, 4) |
| {1-5-7-10-14-10-14-15-17} | a ≠0, b≠0<br>a mod b ≠0 | (6, 4) |



# Cobertura de laços

laços simples



- Procedimento:
  - ① pule o laço
  - ② passe pelo laço 1 vez
  - ③ passe pelo laço m vezes, onde m < max
  - ④ passe pelo laço max-1 vezes
  - ⑤ passe pelo laço max vezes
  - ⑥ tente passar pelo laço max+1 vezes

# Cobertura de laços - exemplo

① Pule o laço

Predicado: a=0 ou

$a \neq 0 \wedge b=0$

② Passe pelo laço 1 vez

Predicado: $a \neq 0 \wedge b \neq 0 \wedge a \bmod b = 0$

③ Passe pelo laço m vezes, onde m < max

Predicado: $a \neq 0 \wedge b \neq 0 \wedge ?$

④ Passe pelo laço max-1 vezes

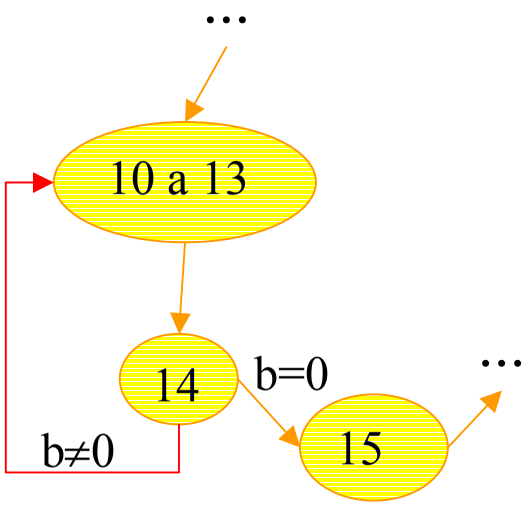
Predicado: ?

⑤ Passe pelo laço max vezes

Predicado: ?

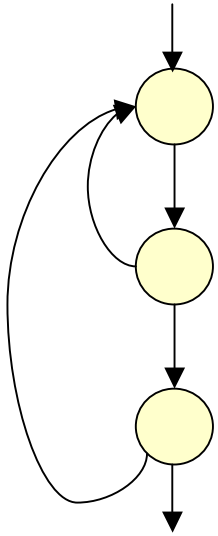
⑥ Tente passar pelo laço max+1 vezes

Predicado: ?



# Cobertura de laços

laços aninhados



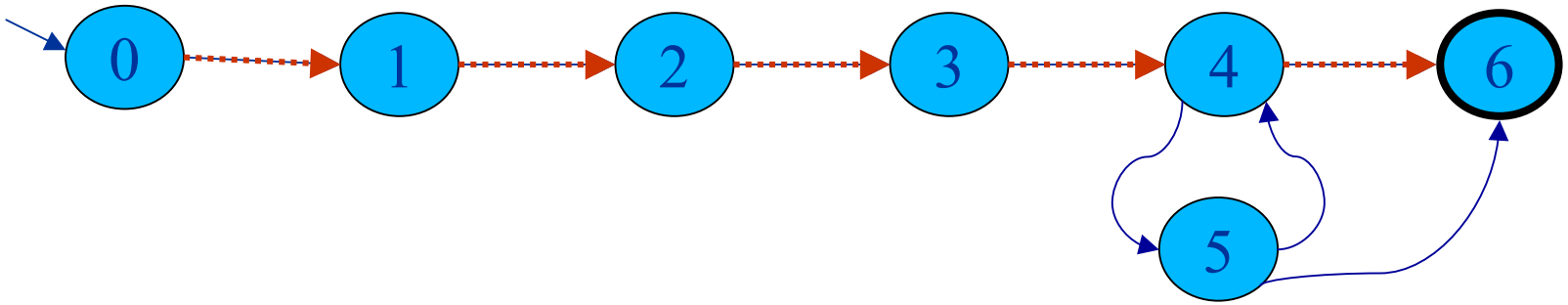
- Procedimento:
  ① comece pelo laço mais interno. Fixe os outros nos valores mínimos
  ② realize testes para laços simples
  ③ caminhe para fora, realizando testes no laço seguinte e mantendo os demais nos valores mínimos
  ④ continue até que todos tenham sido testados

# Testes de caminhos

- Dificuldade 2: alguns requisitos de teste não podem ser satisfeitos por nenhum caso de teste → são chamados **requisitos inviáveis**
  - Requisito de teste está em subgrafo que não é semanticamente alcançável
    - Trecho de código « morto »
    - Subcaminho com predicados contraditórios (ex.: `a>b e a≤b`)
- Determinar se requisitos de teste são inviáveis é um problema **indecidível**
- Ammann e Offutt propõem o uso de percursos com desvios e contornos como forma de satisfazer critérios com requisitos de teste inviáveis:
  - Ao invés de cobertura de caminhos básicos → cobertura de caminhos com desvios ou alternativas
  - ☹ Critério de teste fica mais fraco

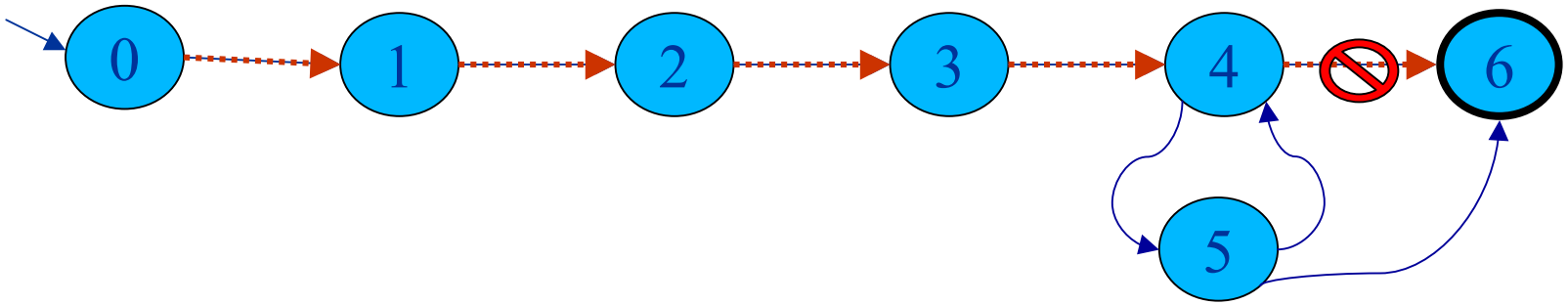
# Caminho requerido

## Percurso original:

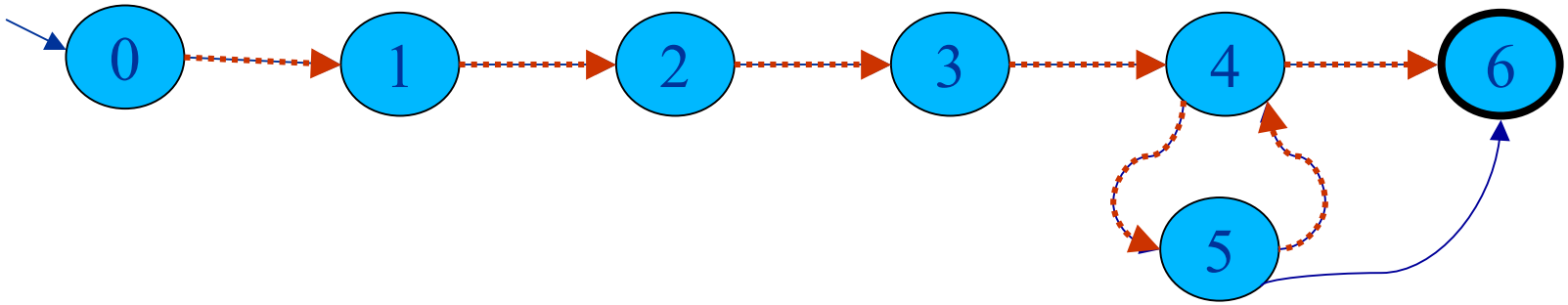


# Caminho com percurso alternativo

Percurso original:

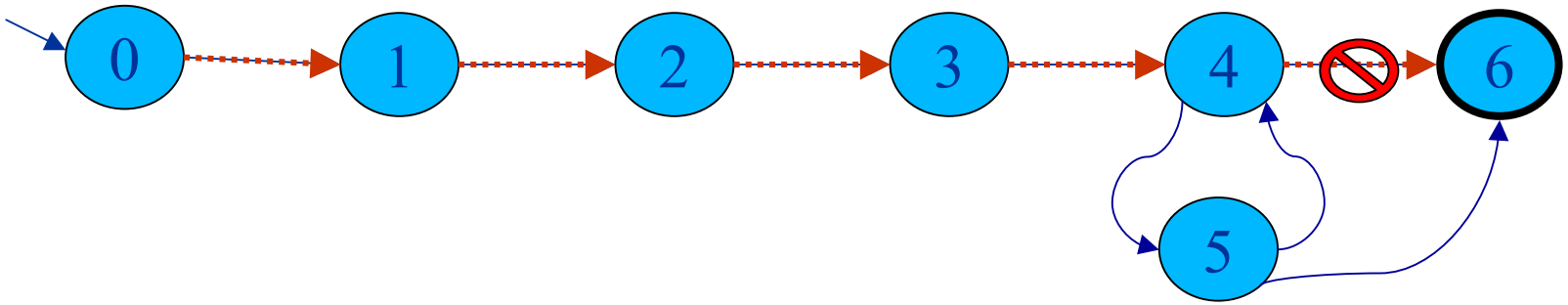



Percurso alternativo:

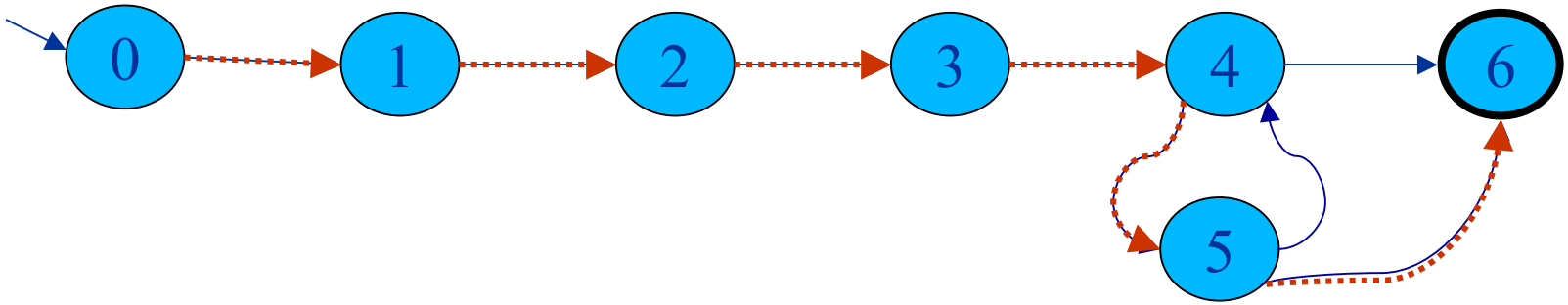


# Caminho com desvio

## Percurso original:




## Percurso com desvio:



# Em suma ...

- Grafos
  - Representam fluxo de controle (ou de dados):
    - Comportamental (testes de caixa preta)
    - Estrutural (testes de caixa branca)
- Critérios baseados em grafos
  - Os mesmos para testes caixa branca ou caixa preta
    - O que muda são os requisitos de teste: casos de uso, instruções do código, ...
  - Cobertura de elementos do grafo:
    - Nós
    - Arestas
    - ...

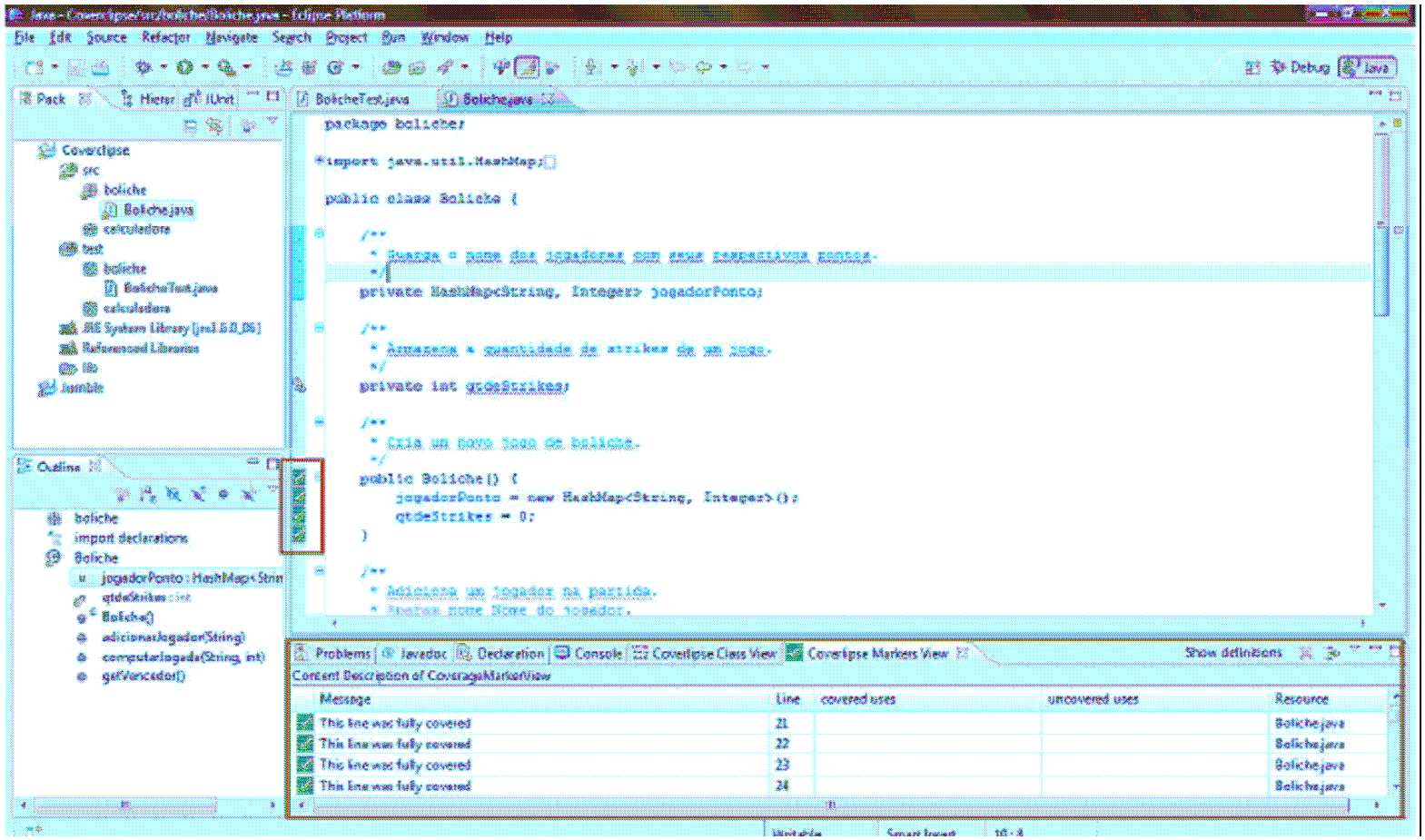
# Automação

- Para usar grafos para projetar casos de teste deve-se proceder de acordo com os seguintes passos:
  1. Defina o grafo
  2. Defina o tipo da relação
  3. Percorra o grafo para atender a critérios selecionados
  - Algoritmos:
    - Busca em profundidade
    - Busca em largura
  - Limitações:
    - Loops (ex.: busca em profundidade k-limitada)
    - Leva em conta aspectos estruturais, mas não semânticos → caminhos inviáveis
- Alternativas?

# Analisar a cobertura

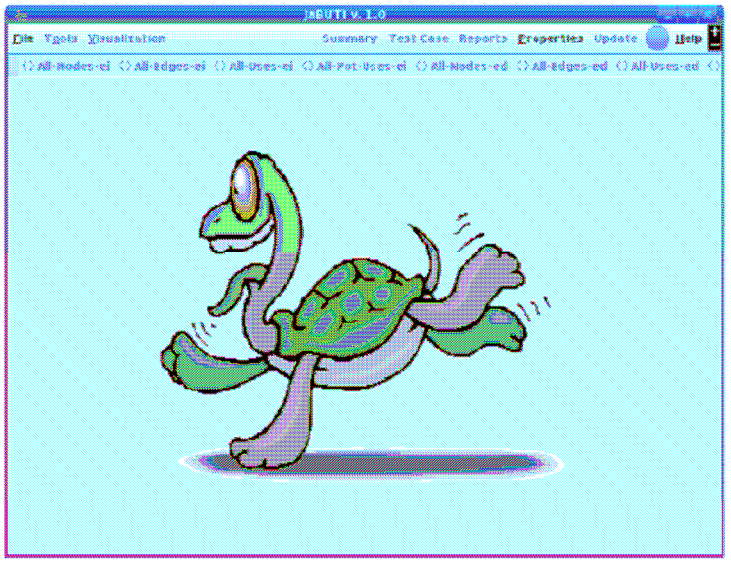
- Não gera testes automaticamente, mas analisa a cobertura dos casos de teste gerados
- Existem inúmeros analisadores de cobertura, tanto comerciais quanto livres:
  - Coverlipse
    - http://coverlipse.sourceforge.net/
  - PureCoverage (Rational/IBM)
  - EMMA/ECLEMMA
    - http://emma.sourceforge.net/
  - Outras ferramentas livres:
    - http://java-source.net/open-source/code-coverage

# Coverlipse


File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Java
Coverclipse
src
boliche
Bolichejava
calculadora
test
boliche
BolicheTest.java
calculadora
JRE System Library [jre1.6.0_06]
Referenced Libraries
lib
Jumble
BolicheTest.java
Bolichejava
package boliche;
import java.util.HashMap;
public class Boliche {
/**
* Guarda o nome dos jogadores com seus respectivos pontos.
*/
private HashMap<String, Integer> jogadorPonto;
/**
* Armazena a quantidade de strikes de um jogo.
*/
private int qtdeStrikes;
/**
* Cria um novo jogo de boliche.
*/
public Boliche() {
jogadorPonto = new HashMap<String, Integer>();
qtdeStrikes = 0;
}
/**
* Adiciona um jogador na partida.
* @param nome Nome do jogador.
Outline
boliche
import declarations
Boliche
jogadorPonto : HashMap<Strin
qtdeStrikes : int
Boliche()
adicionarJogador(String)
computarJogada(String, int)
getVencedor()
Problems
Javadoc
Declaration
Console
Coverlipse Class View
Coverlipse Markers View
Show definitions
Content Description of CoverageMarkerView
Message
Line
covered uses
uncovered uses
Resource
This line was fully covered
21
Bolichejava
This line was fully covered
22
Bolichejava
This line was fully covered
23
Bolichejava
This line was fully covered
24
Bolichejava

# Jabuti

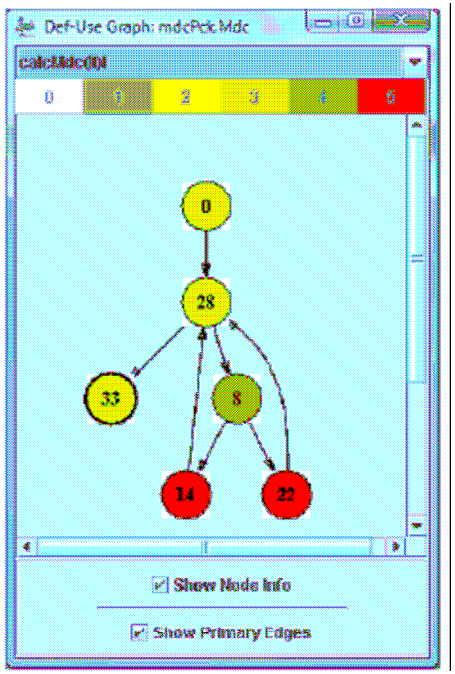
- **JaBUTi (Java Bytecode Understanding and Testing)**
  - **Ferramenta que automatiza critérios de teste caixa branca**
  - **Foi apresentada no XVII Simpósio Brasileiro de Engenharia de Software (2003)**
  - **Cobre tanto critérios de fluxo de controle quanto de dados**



# Preparação para os testes

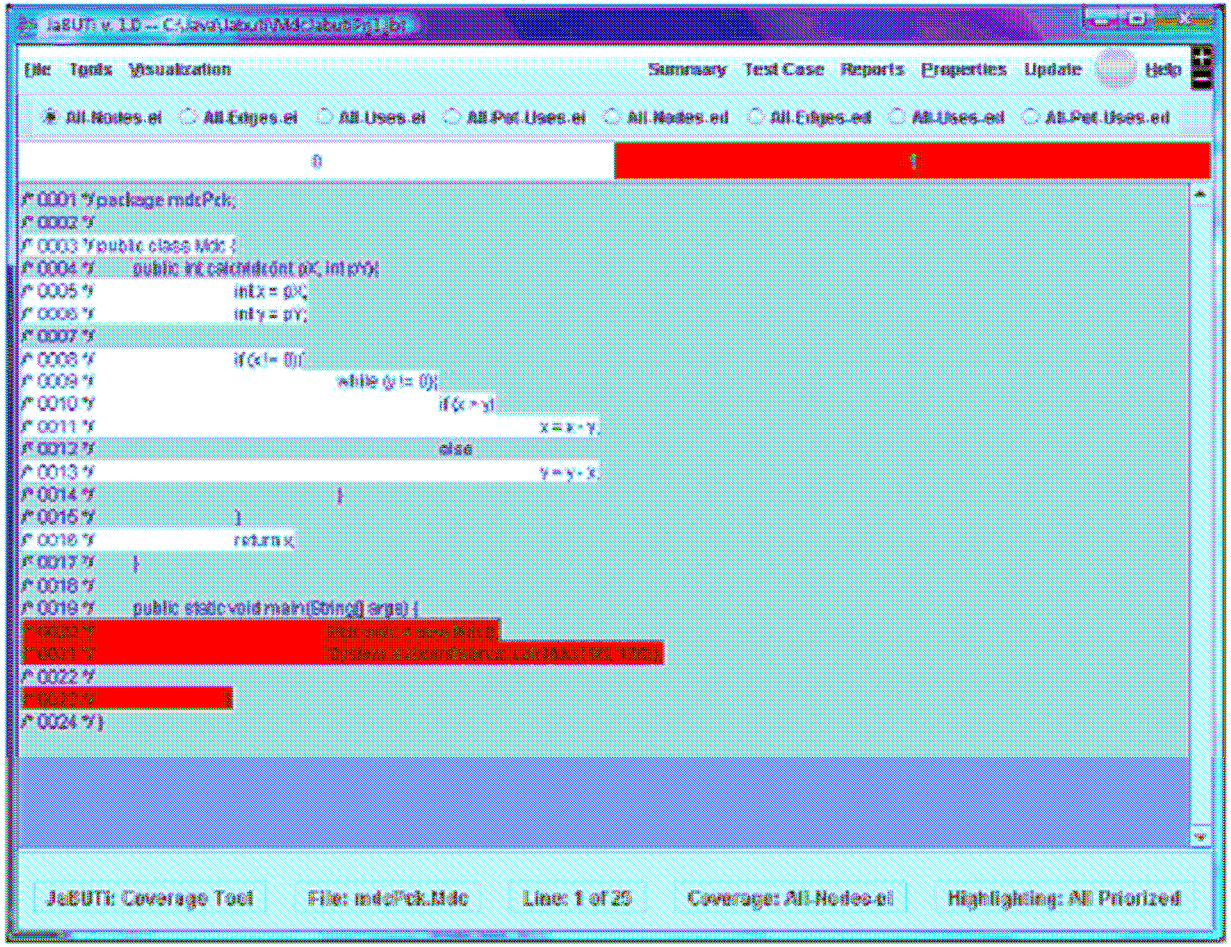

Seleção de critérios e indicação dos requisitos de teste

Grafo de fluxo de controle

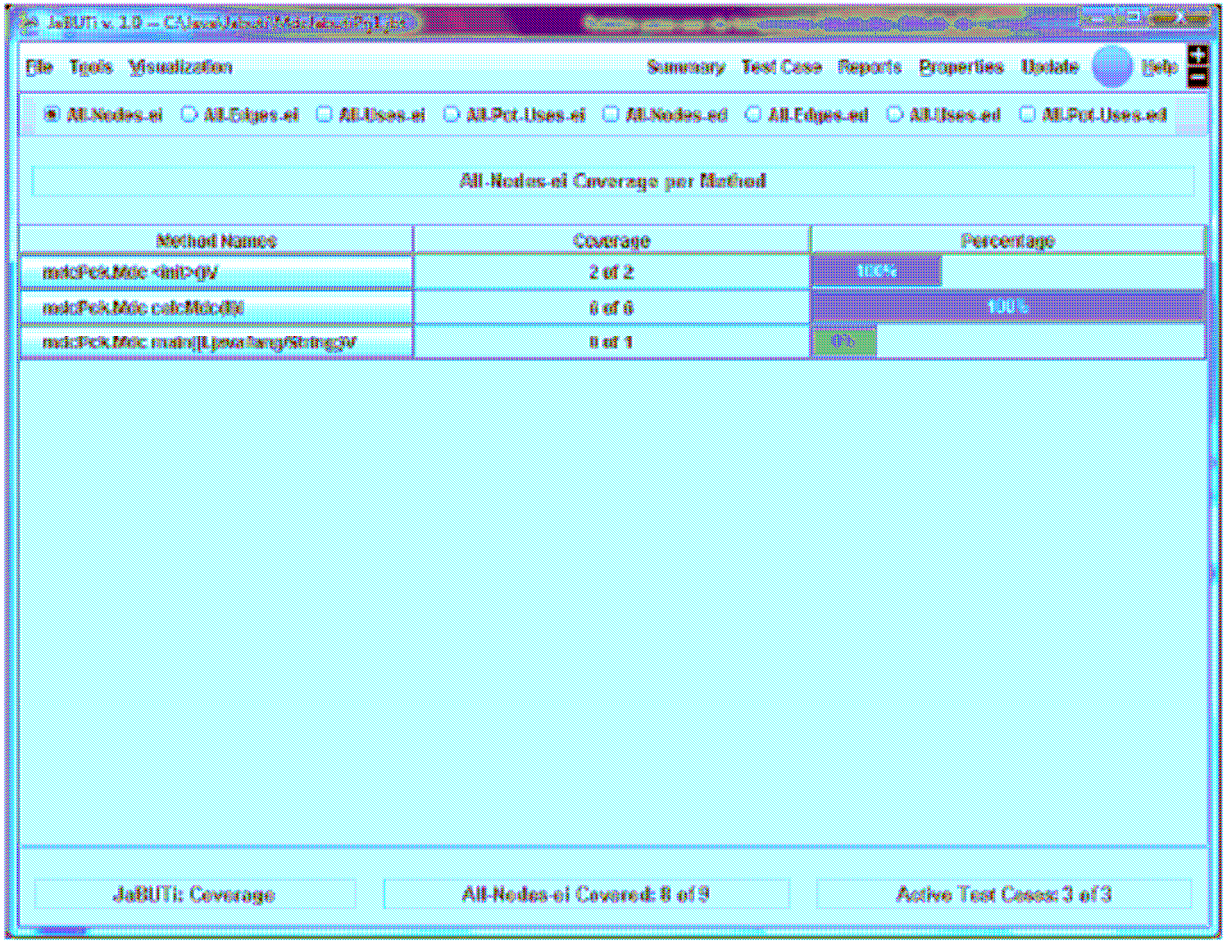


# Exemplo de resultados

Requisitos cobertos
e não cobertos



# Exemplo de resultados




Cobertura

# Análise de Cobertura - considerações

- Objetivos
  - Definir os objetivos dos testes
    - *Cobrir 100% dos requisitos de teste*
  - Avaliar os resultados obtidos
    - *Somente 70% dos requisitos de teste foram cobertos*

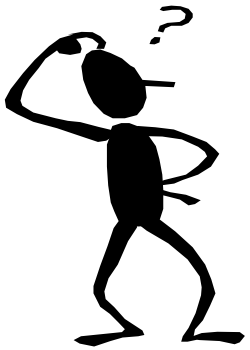
O que fazer?

# Análise de Cobertura - considerações

- Objetivos
  - Definir os objetivos dos testes
    - *Cobrir 100% dos requisitos de teste*
  - Avaliar os resultados obtidos
    - *Somente 70% dos requisitos de teste foram cobertos*

Projetar mais testes até que o objetivo seja atingido (grau de cobertura)

# Análise de Cobertura – considerações sobre completeza



É sempre possível ter
100% de cobertura?

# Análise de Cobertura – considerações sobre completeza


- Natureza da unidade
  - Simplicidade, baixa relevância → para que cobrir 100%?
  - Há caminhos que não são viáveis:
    - Ex.: instruções inatingíveis
  - Há partes difíceis de alcançar sem componentes auxiliares
    - Ex.: tratadores de exceção
- Falta de recursos
  - Pouco tempo disponível
  - Pouco pessoal treinado
  - Poucas ferramentas para auxiliar

```
/*
 * "Ler 3 números inteiros da entrada, imprimir o menor e o maior"
  * Autor: Jacques Sauvé
 */
public class MinMax4 {
    public static void main(String[] args) {
        final int NÚMEROS_A_LER = 3;
        Scanner sc = new Scanner(System.in);
        int mínimo = Integer.MAX_VALUE;
        int máximo = Integer.MIN_VALUE;
        for (int i = 0; i < NÚMEROS_A_LER; i++) {
            System.out.print("Entre com o proximo inteiro: ");
            int num = sc.nextInt();
            if (num < mínimo) {
                mínimo = num;
            }
            if (num > máximo) {
                máximo = num;
            }
        }
        System.out.println("O menor numero eh: " + mínimo);
        System.out.println("O maior numero eh: " + máximo);
    }
}
```

# Exercício

- Obtenha o grafo de fluxo de controle para o método main.
- Crie um conjunto de testes que satisfaça ao critério de todas as instruções.
- Crie um conjunto de testes que satisfaça ao critério de todos os ramos.
- O número de casos de teste nos dois casos é o mesmo? Porquê?