

ACH2028 – Qualidade de Software

Aula 09 – Geração Automática de Testes Execução Simbólica

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Teste de Software

- Consiste em executar um programa com o objetivo de revelar uma falha (Myers, 1979)
- Uma falha é qualquer evento do sistema que viola um objetivo de qualidade estabelecido
- Exemplos típicos de falhas (failures)
 - Crash
 - Resultado errado
 - Tempo de resposta excedido
 - Formato de saída fora do padrão

Teste de Software

- Envolve:
 - Determinar as partes do sistema e as propriedades que serão testadas
 - Definir valores de entrada apropriados
 - Determinar resultados esperados (oráculos)
 - Executar o sistema (ou parte dele) com os valores entrada definidos e selecionados
 - Comparar os resultados de execução com os oráculos
 - Avaliar os resultados

Teste de Software

- Um **caso de teste** especifica:
 - A configuração/estado do sistema antes da execução dos testes
 - Os dados de teste
 - Sequências de invocação
 - Os resultados esperados (oráculos)

Teste Estrutural

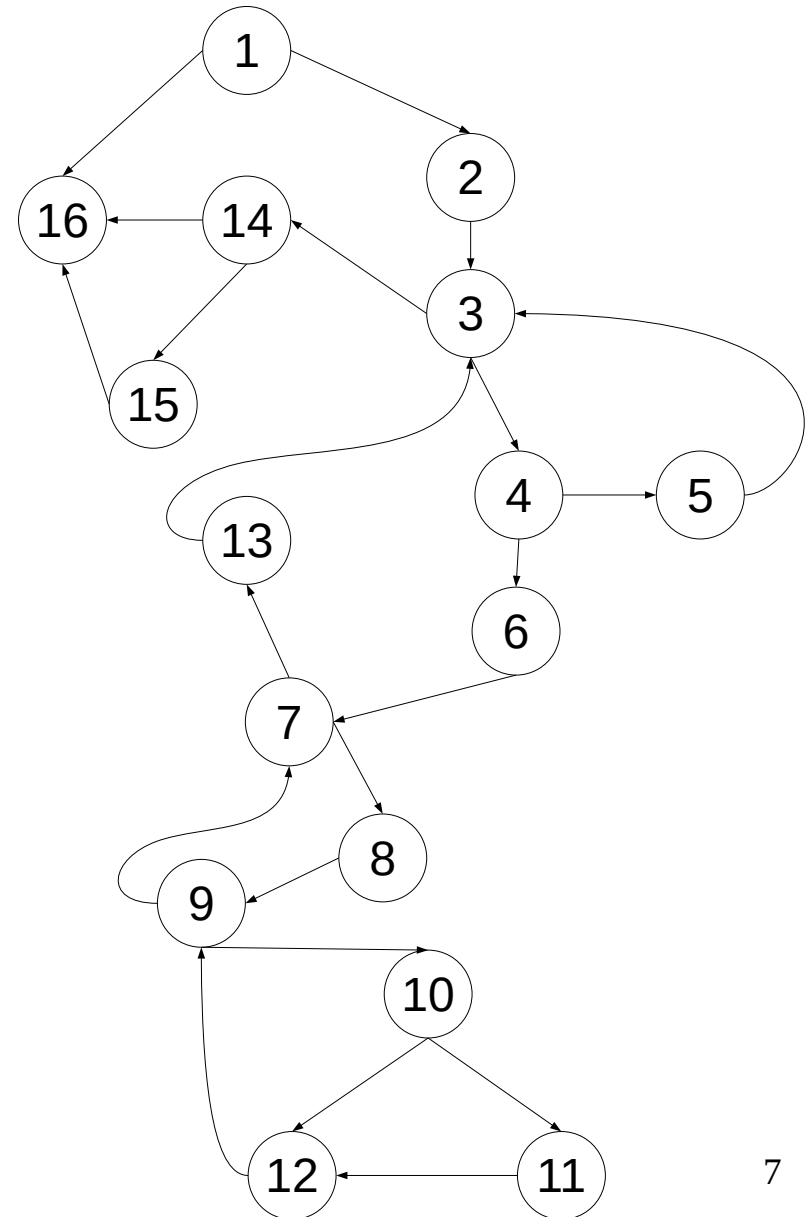
- Tem o objetivo de exercitar todas as estruturas internas (instruções e dados) de um programa
- Define os valores de entrada com base na implementação (código-fonte, por exemplo)
- Exemplos de critérios:
 - Todas as instruções
 - Todos os caminhos de execução
 - Todos os blocos de instruções (todos-nós)
 - Todos os desvios de fluxos (todas-arestas)
 - Todos os usos de variáveis (todos-usos)

Teste Estrutural

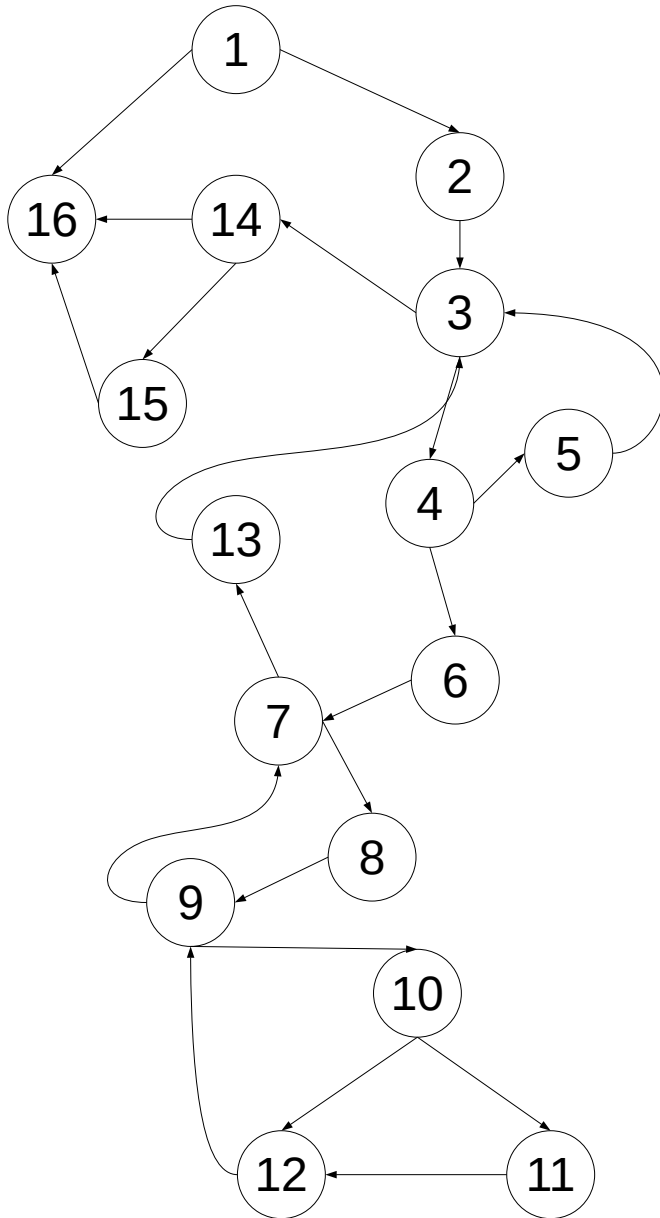
- Um modelo chamado de Grafo de Fluxo de Controle (CFG – Control Flow Graph) é geralmente adotado para representar a estrutura interna de um programa sob teste
- O GFC é usado para identificar os blocos de instruções, os desvios de fluxo e os caminhos de execução possíveis
- É possível definir os requisitos de teste para alguns critérios da técnica estrutural com o auxílio do GFC

Teste Estrutural

```
01 public void factorization(int N)
02     if (N>0){ //1
03         int prime= 2; //2
04         int number = N; //2
05         while (prime<=N/2){ //3
06             if (number % prime==0){ //4
07                 System.out.println(prime); //5
08                 number = number / prime; //5
09             }
10         } else
11         {
12             int nextPrime=prime; //6
13             int found=0; //6
14             while (found==0){ //7
15                 nextPrime = nextPrime+1; //8
16                 found=1; //8
17                 int d = 2; //8
18                 while (d<=nextPrime/2){ //9
19                     if (nextPrime % d == 0) //10
20                         found=0; //11
21                     d++; //12
22                 }
23             }
24             prime = nextPrime; //13
25         }
26     }
27     if (number>1) //14
28         System.out.println(number); //15
29 }
30 //16 - return
```



Teste Estrutural



Criterion	Requirements
all-nodes	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
all-edges	(1-2), (1-16), (2-3), (3-14), (3-4), (14-15) (14-16), (15-16), (4-5), (4-6), (5-3), (6-7) (7-8), (7-13), (13-3), (8-9), (9-10), (9-7) (10-11), (10-12), (11-12), (12-9)
all-paths (1-loop)	... P01: 1 2 3 4 6 7 8 9 10 11 12 9 7 13 3 14 15 16 P02: 1 2 3 4 6 7 8 9 10 11 12 9 7 13 3 14 16 P03: 1 2 3 4 6 7 8 9 10 12 9 7 13 3 14 15 16 P04: 1 2 3 4 6 7 8 9 10 12 9 7 13 3 14 16 P05: 1 2 3 4 6 7 8 9 7 13 3 14 15 16 P06: 1 2 3 4 6 7 8 9 7 13 3 14 16 P07: 1 2 3 4 6 7 13 3 14 15 16 P08: 1 2 3 4 6 7 13 3 14 16 P09: 1 2 3 4 5 3 14 15 16 P10: 1 2 3 4 5 3 14 16 P11: 1 2 3 14 15 16 P12: 1 2 3 14 16 P13: 1 16

Teste Estrutural

- Como escolher os valores de entrada para satisfazer todos os requisitos de teste gerados pelos critérios estruturais?
- Cenário comum:
 - 1 - Criar casos de teste para satisfazer os critérios da técnica de teste funcional
 - 2 – Executar os casos de teste e fazer uma análise da cobertura dos critérios da técnica estrutural
 - 3 – Analisar o código e criar casos de teste específicos para cobrir os requisitos de teste que ainda não foram satisfeitos

Teste Estrutural

- Limitações deste cenário:
 - Complexidade
 - Descobrir a configuração necessária para se satisfazer um determinado critério, principalmente quando a lógica é complexa
 - Complexidade de alguns critérios, como o todos os usos, por exemplo
 - Testadores devem conhecer bem a linguagem de desenvolvimento
 - Estruturas implícitas
 - Custo

Teste estrutural

- Essas limitações são determinadas pelas restrições da capacidade humana
- Uma solução viável:
 - automatizar o processo de geração dos dados de teste
- Uma abordagem em particular:
 - Combinação das técnicas de **execução simbólica** e de **solucionadores de restrições** para gerar dados de teste que satisfazem os requisitos de teste definidos pelos critérios estruturais

Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

x: x, y: y, N: 0

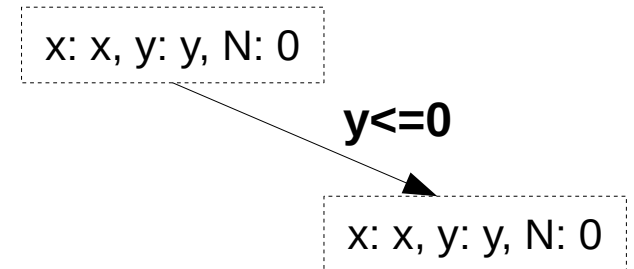
```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

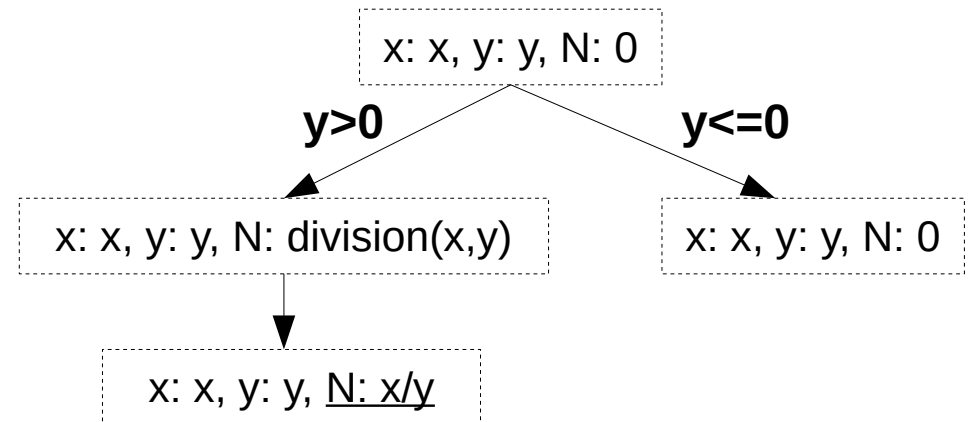


Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

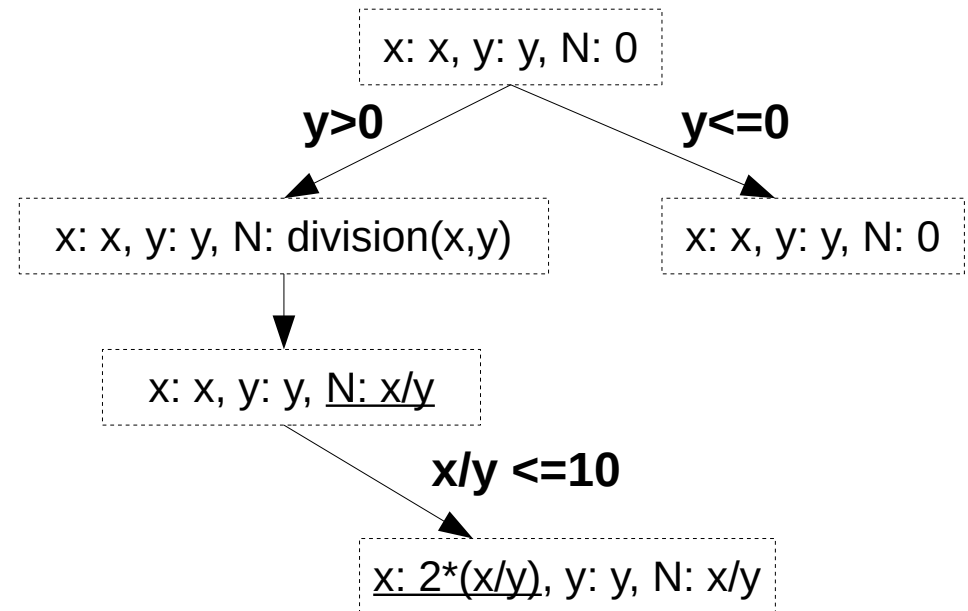


Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

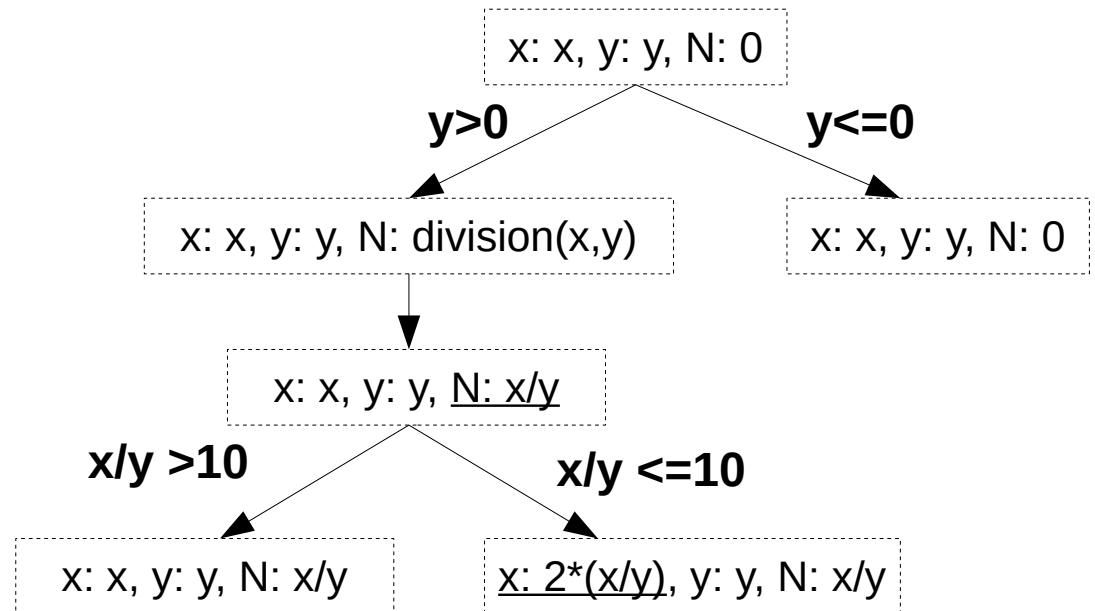


Execução Simbólica

- A ideia geral é que os valores das variáveis locais sejam representados em função dos valores de entrada

```
public int division(int x, int y)
    return x/y;
}
```

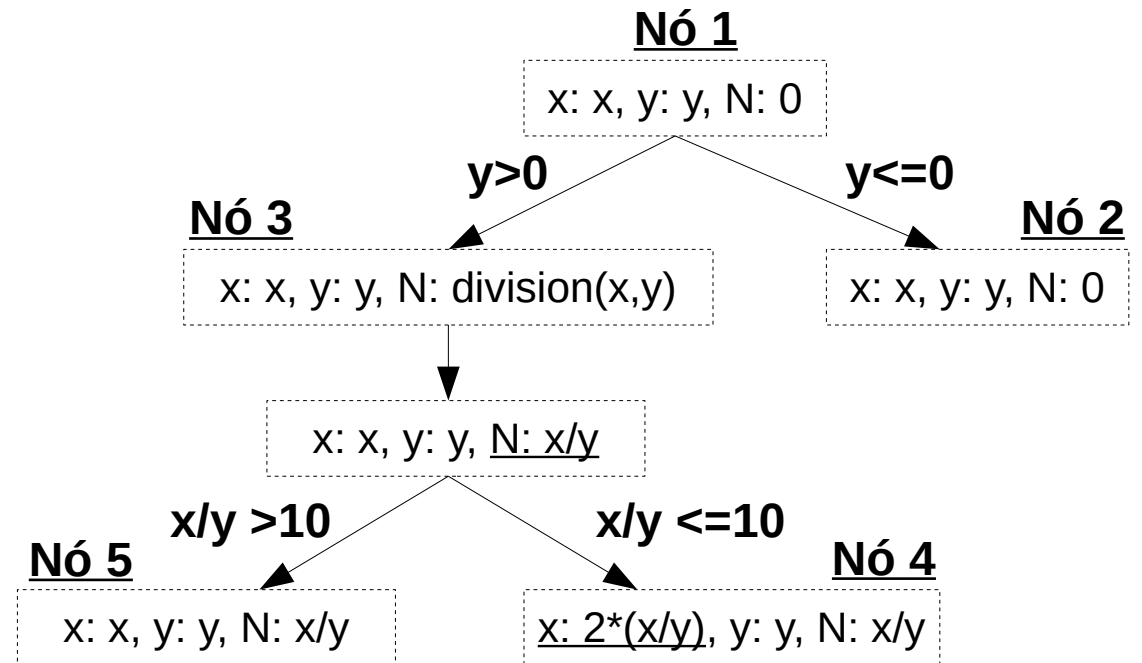
```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```



Execução Simbólica

```
public int division(int x, int y)
    return x/y;
}
```

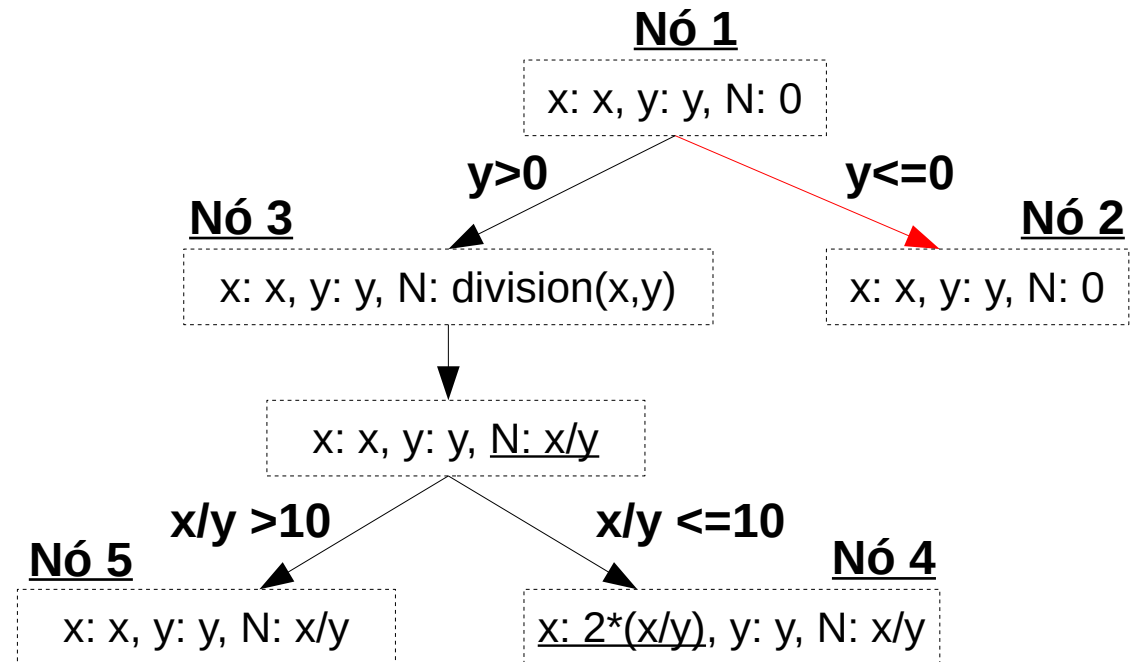
```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```



Execução Simbólica

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```

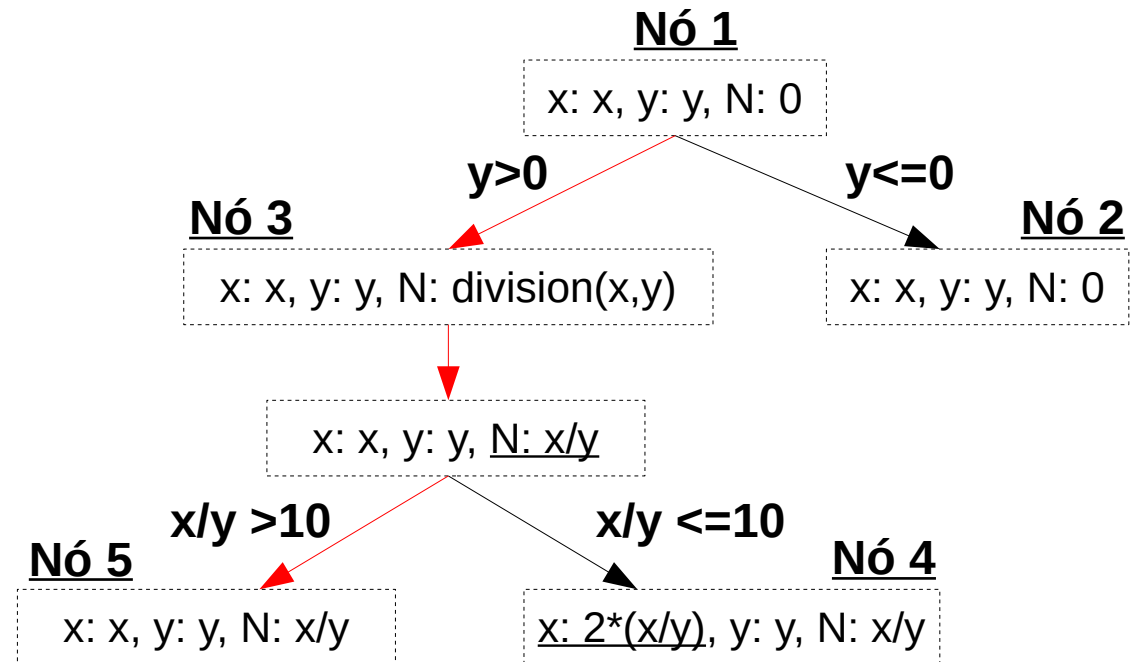


Caminho a: 1, 2.
Restrições: $y \leq 0$.
Possível solução $y=0$

Execução Simbólica

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```



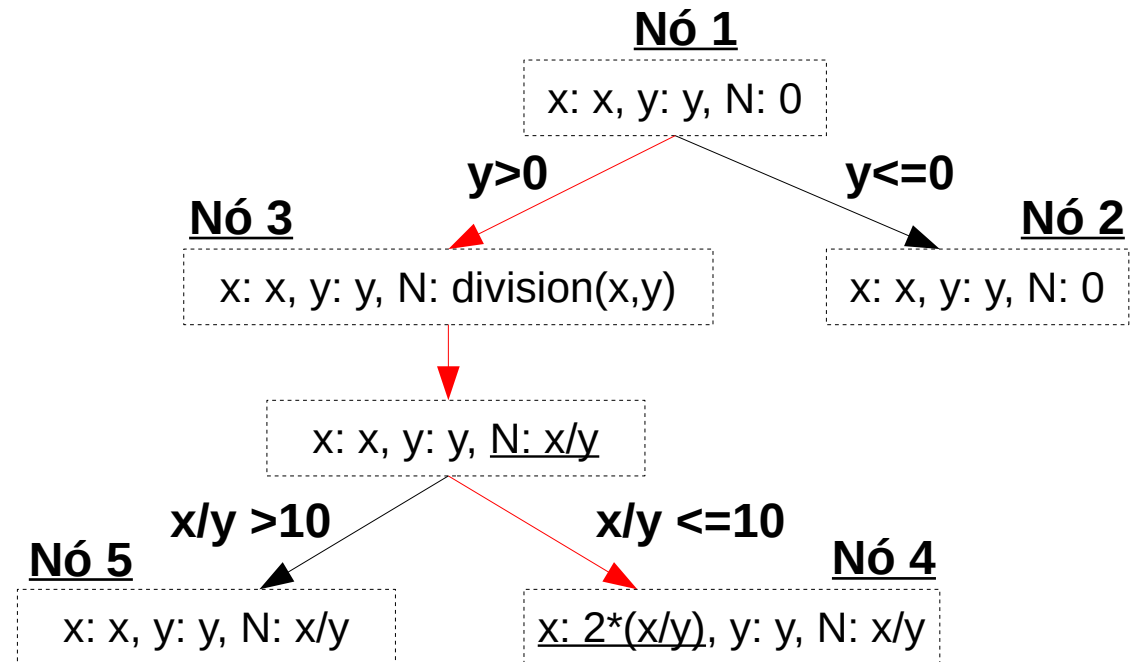
Caminho a: 1, 2.
Restrições: $y \leq 0$.
Possível solução $y=0$

Caminho b: 1, 3, 5.
Restrições: $y > 0 \wedge x/y > 10$
Possível solução: $x=11, y=1$

Execução Simbólica

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```



Caminho a: 1, 2.
Restrições: $y \leq 0$.
Possível solução $y=0$

Caminho b: 1, 3, 5.
Restrições: $y > 0 \wedge x/y > 10$
Possível solução: $x=11, y=1$

Caminho c: 1, 3, 4.
Restrições: $y > 0 \wedge x/y \leq 10$
Possível solução: $x=10, y=1$

Context


- Symbolic execution (a simple example)

```
public int testMe(int x, int y)
{
    if (y!=0){
        int N = x/y;
        if (N > 10)
            return N;
        else {
            N = 2*x*y;
            return N;
        }
    }
    else
        return x;
}
```

Context

- Symbolic execution (a simple example)

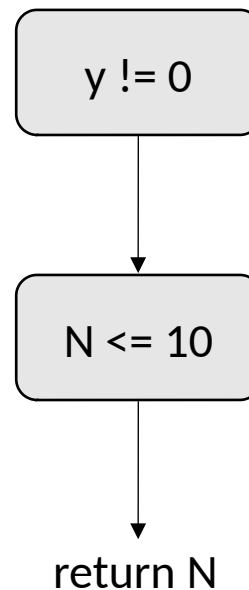
```
public int testMe(int x, int y)
{
    if (y!=0){
        int N = x/y;
        if (N > 10)
            return N;
        else {
            N = 2*x*y;
            return N;
        }
    }
    else
        return x;
}
```



Context

- Symbolic execution (a simple example)

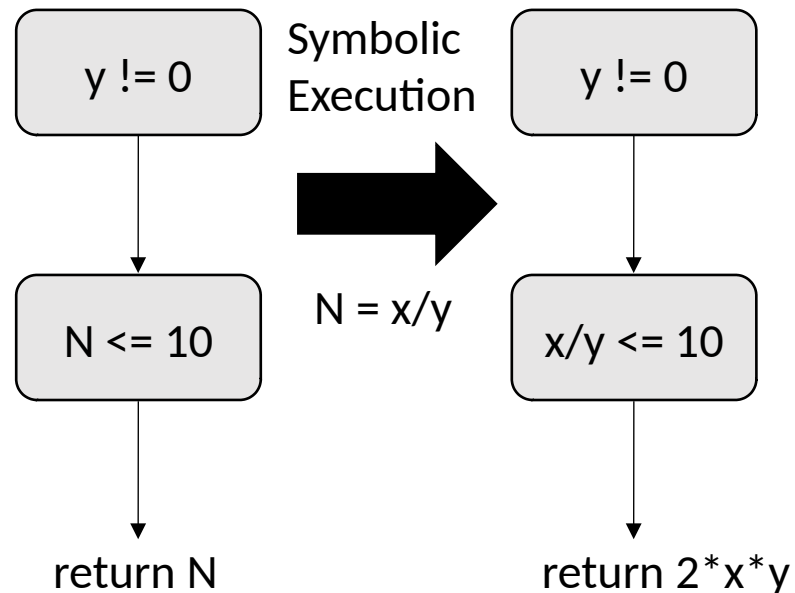

```
public int testMe(int x, int y)
  if (y!=0){
    int N = x/y;
    if (N > 10)
      return N;
    else {
      N = 2*x*y;
      return N;
    }
  }
  else
    return x;
}
```



Context

- Symbolic execution (a simple example)

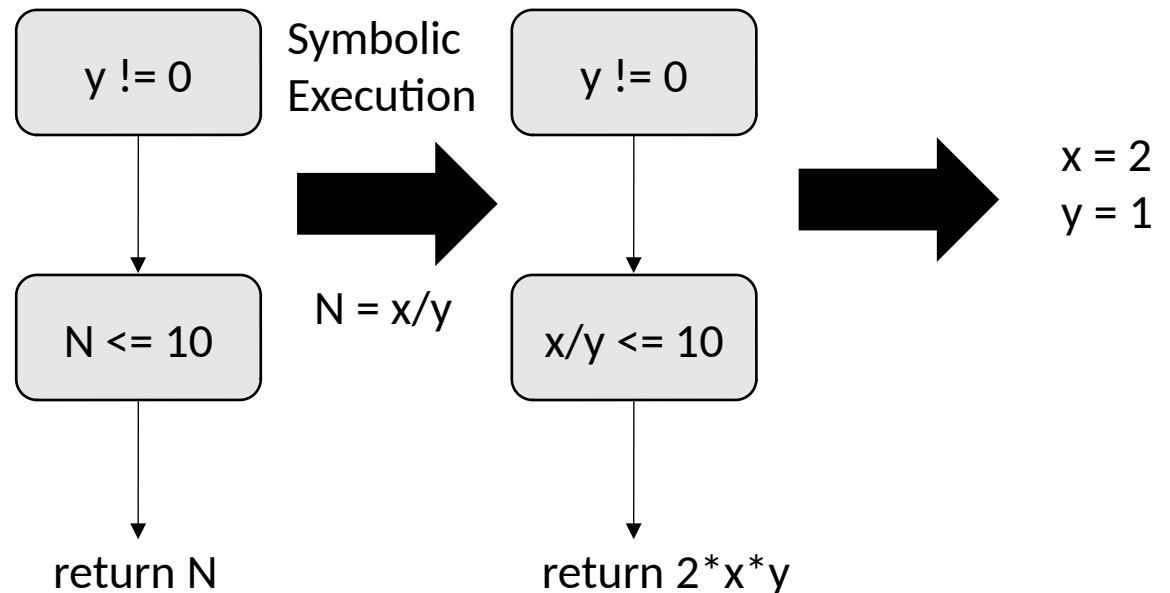

```
public int testMe(int x, int y)
{
    if (y!=0){
        int N = x/y;
        if (N > 10)
            return N;
        else {
            N = 2*x*y;
            return N;
        }
    }
    else
        return x;
}
```



Context

- Symbolic execution (a simple example)

```
public int testMe(int x, int y)
{
    if (y!=0){
        int N = x/y;
        if (N > 10)
            return N;
        else {
            N = 2*x*y;
            return N;
        }
    }
    else
        return x;
}
```



Exemplo - Triângulo

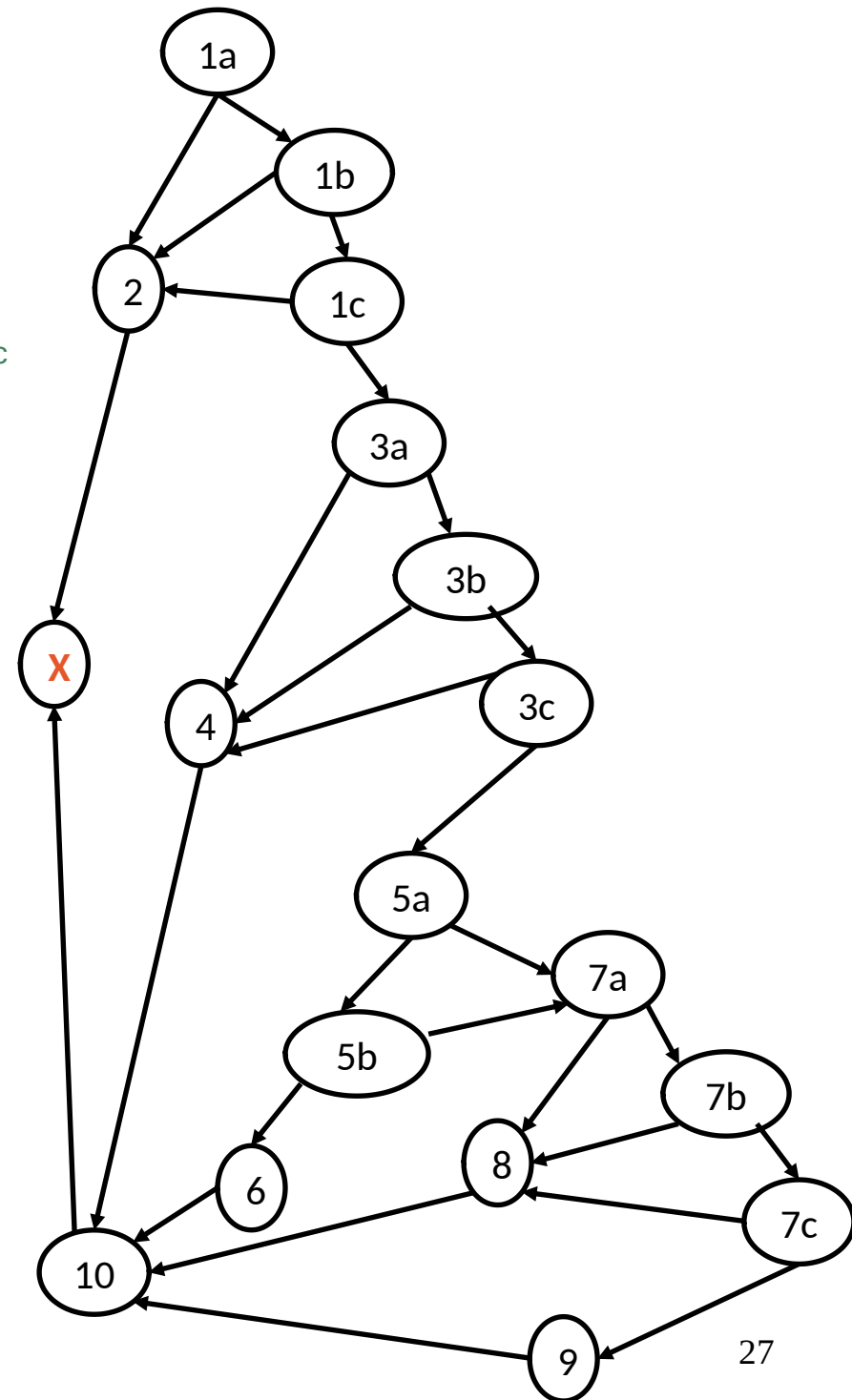
```

classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a, 3b, 3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a, 5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a, 7b, 7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}

return resposta; //10
}

```



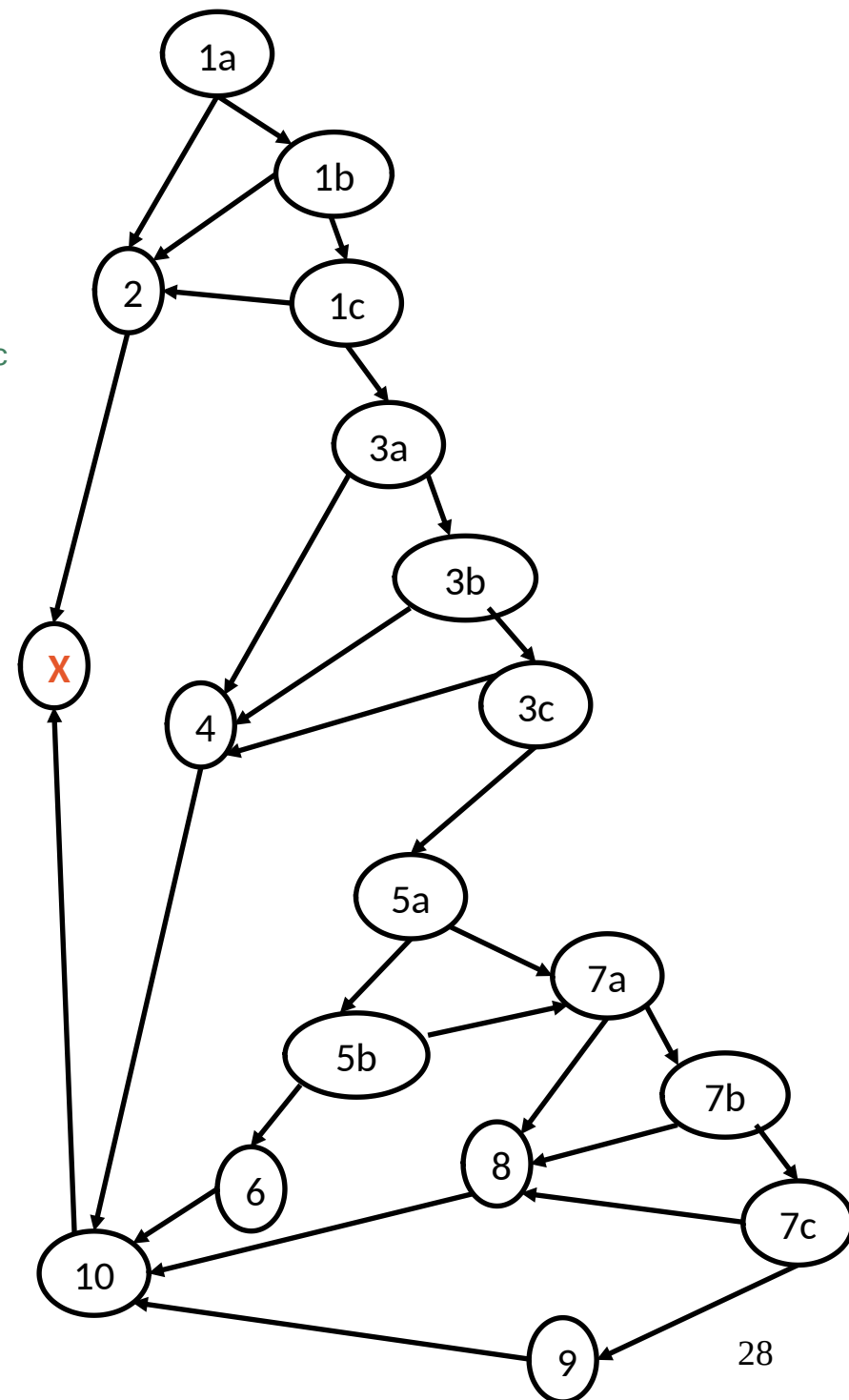
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminhos básicos:

1a - 2 - X
1a - 1b - 2 - X
1a - 1b - 1c - 2 - X
1a - 1b - 1c - 3a - 4 - 10 - X
1a - 1b - 1c - 3a - 3b - 4 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 4 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 5b - 6 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 8 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 5b - 7a - 8 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 7c - 8 - 10 - X
1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 7c - 9 - 10 - X

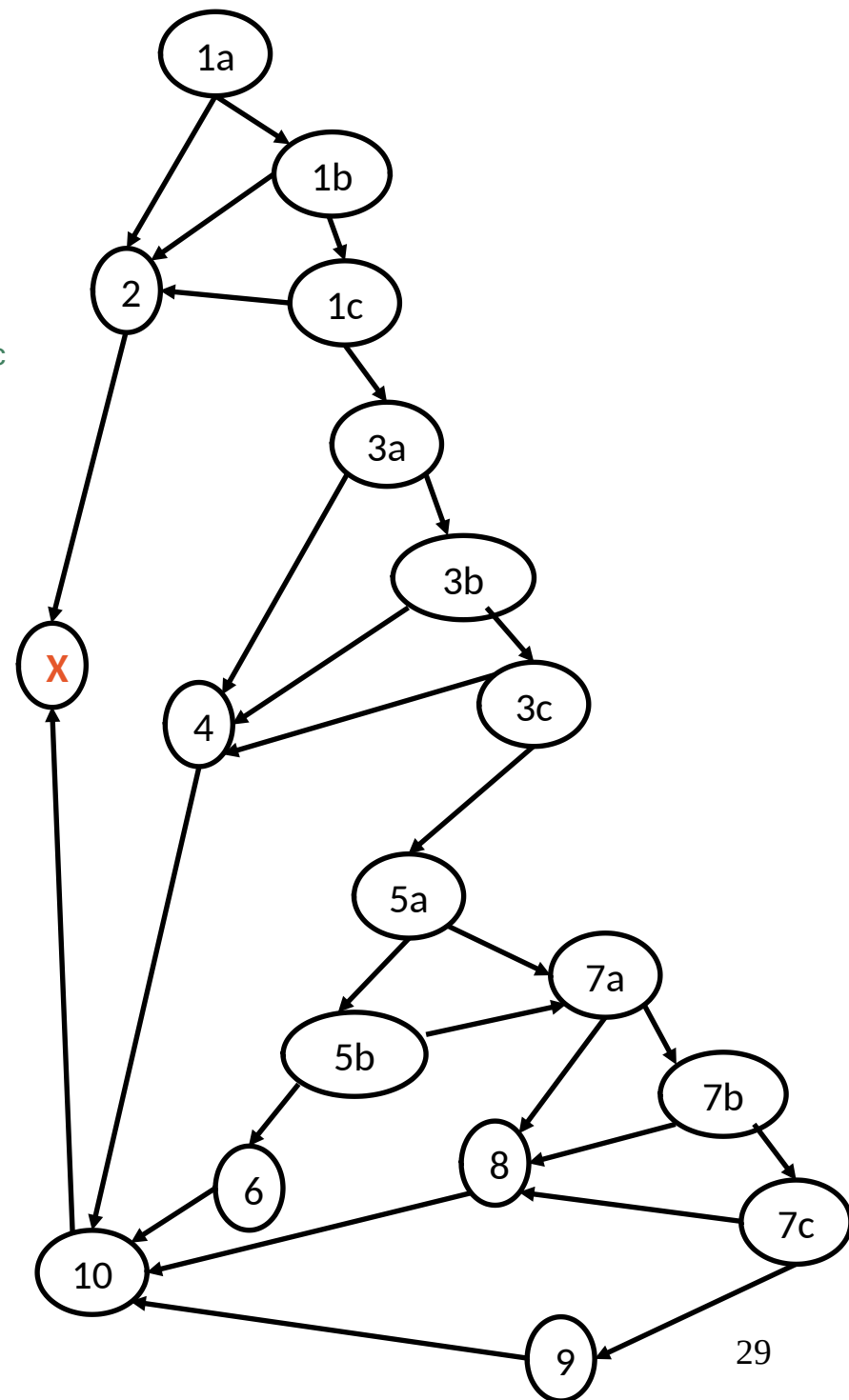


Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X



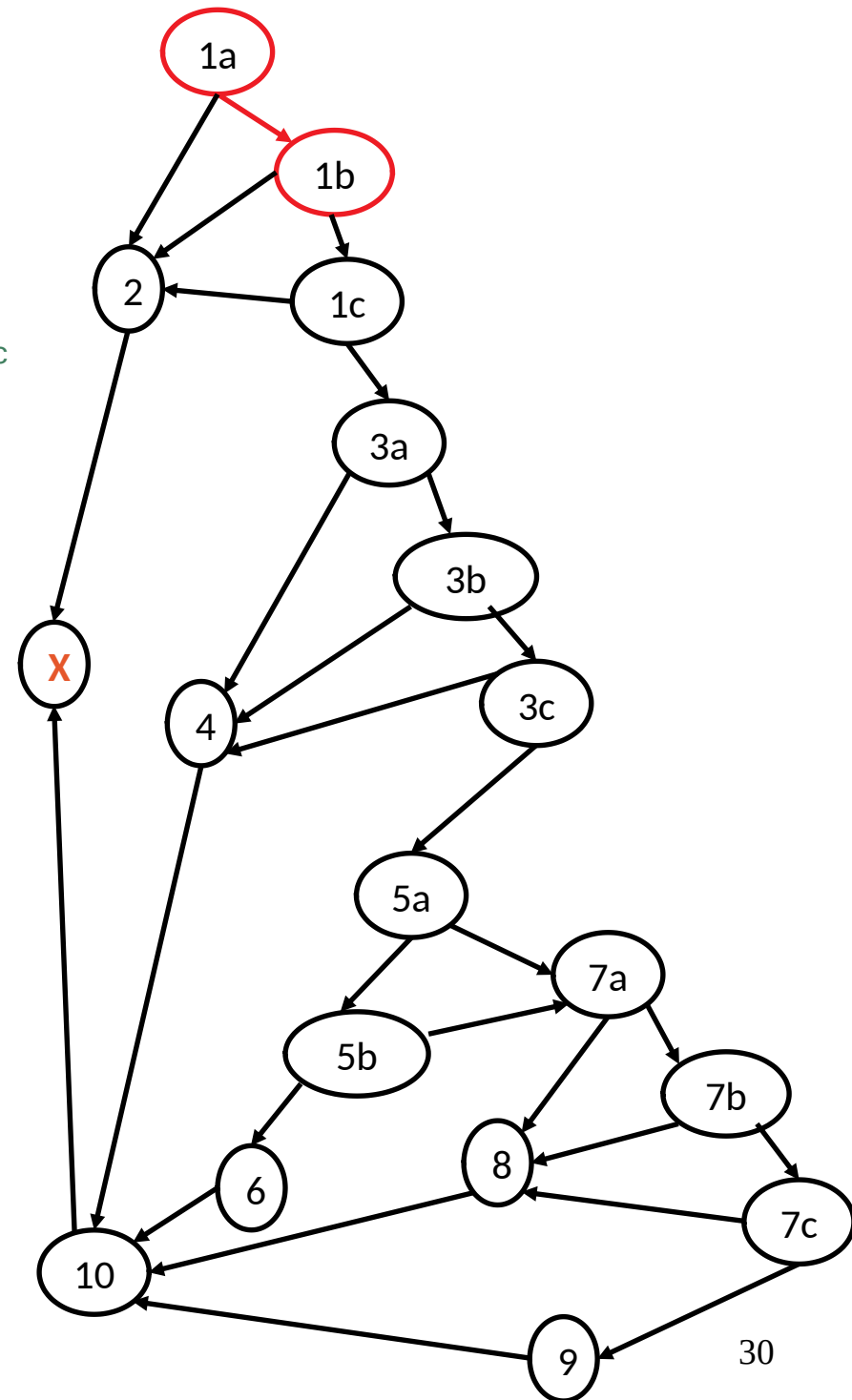
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

LA > 0



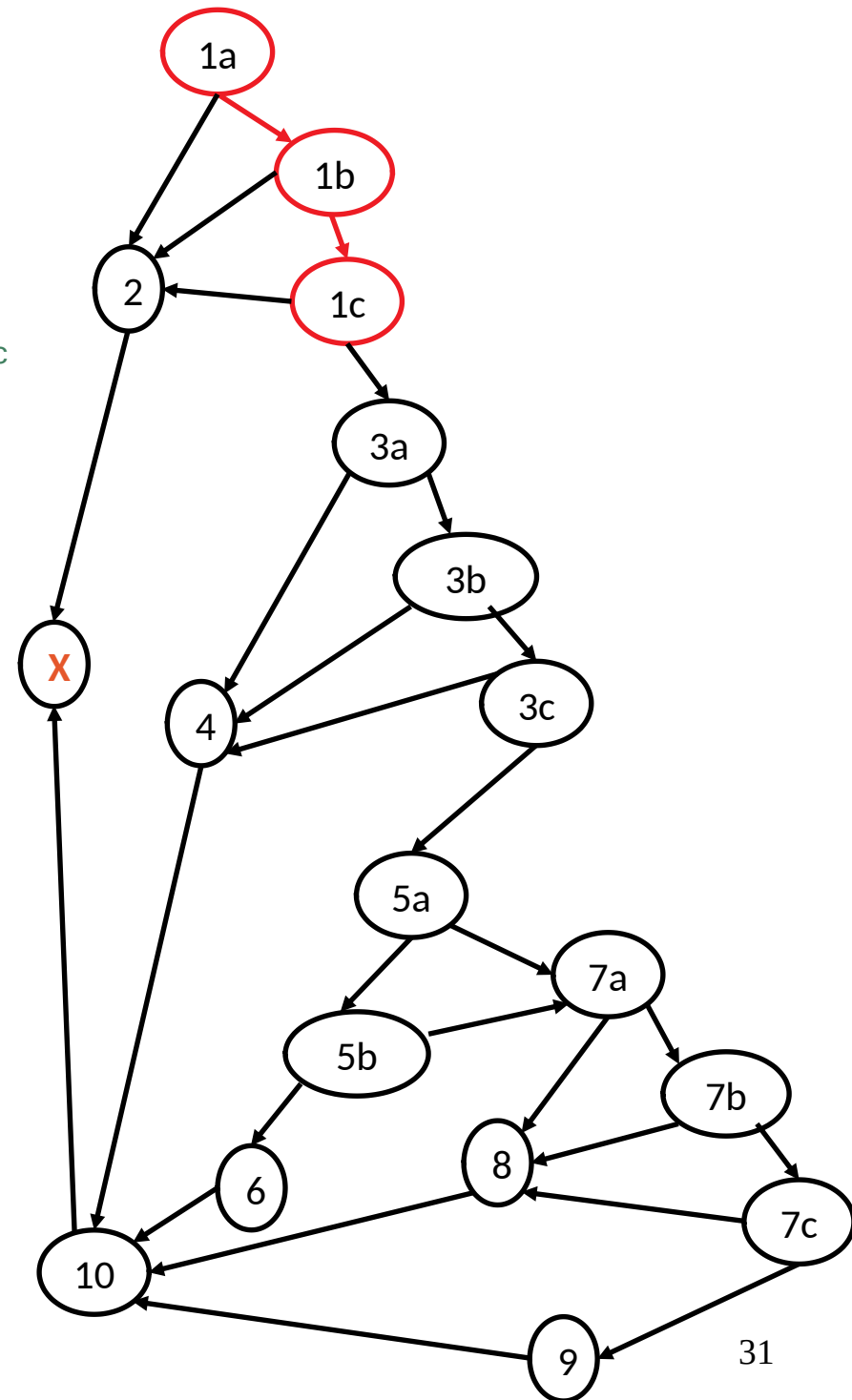
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0)$



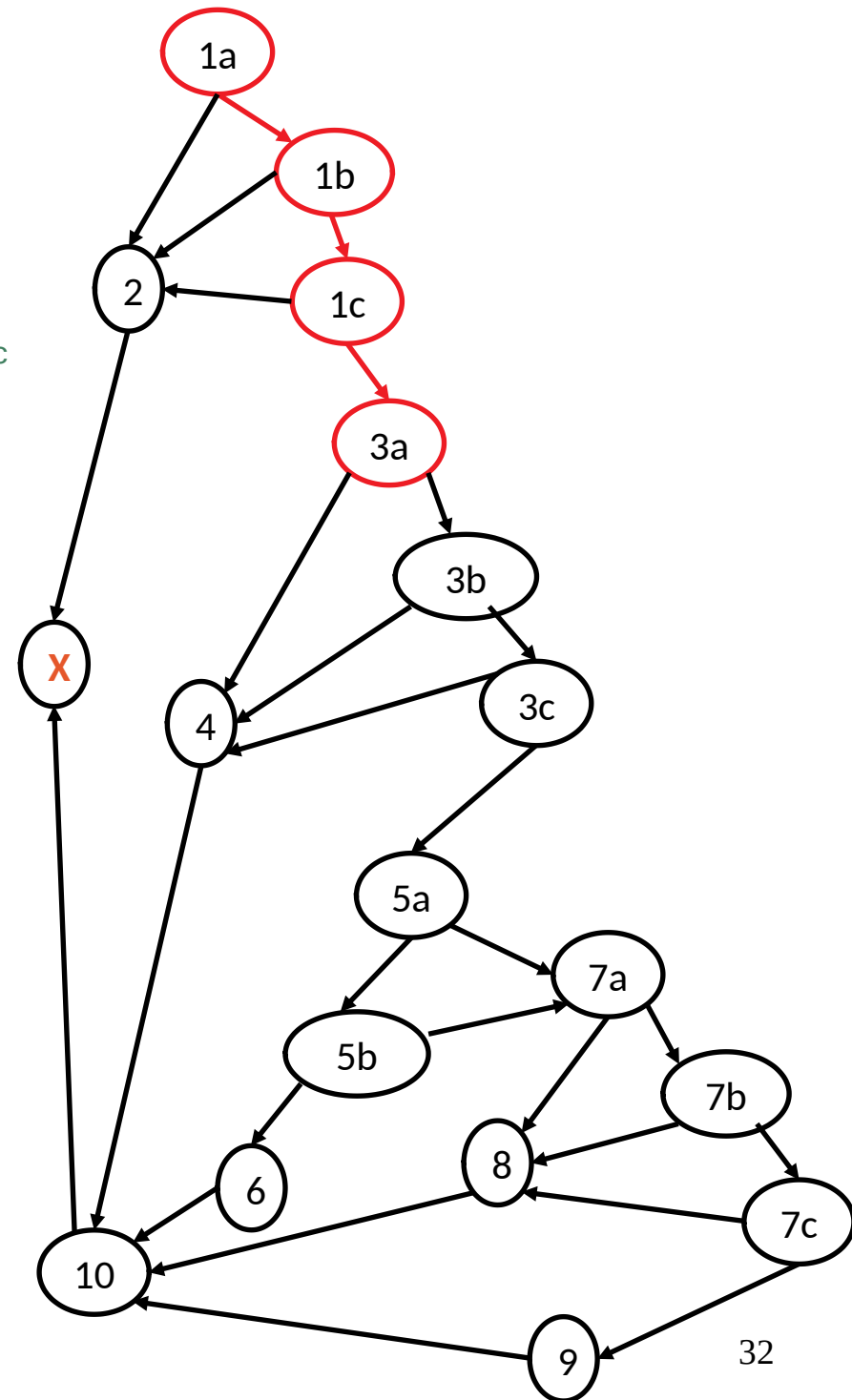
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0)$



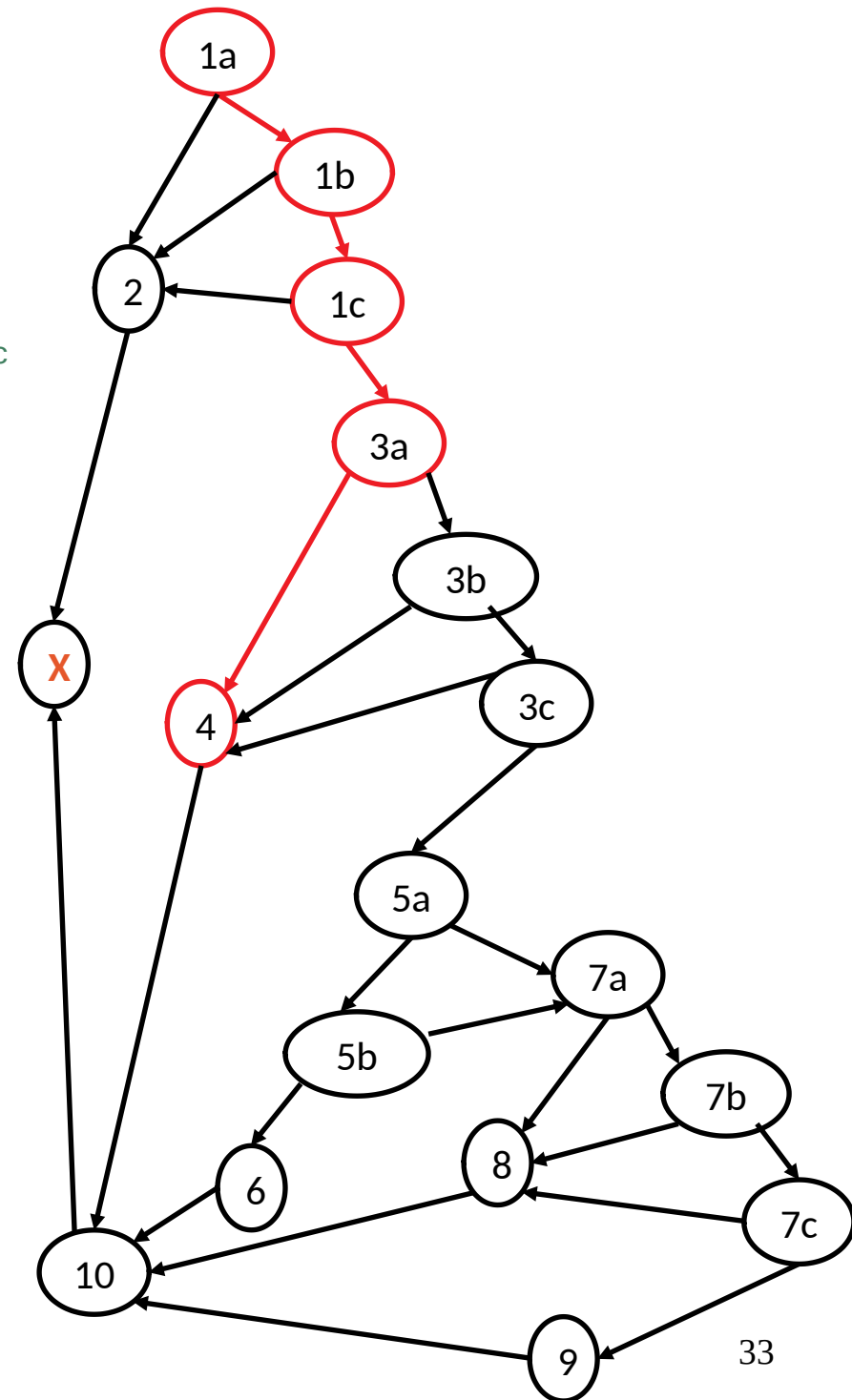
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge (LA \geq LB + LC)$



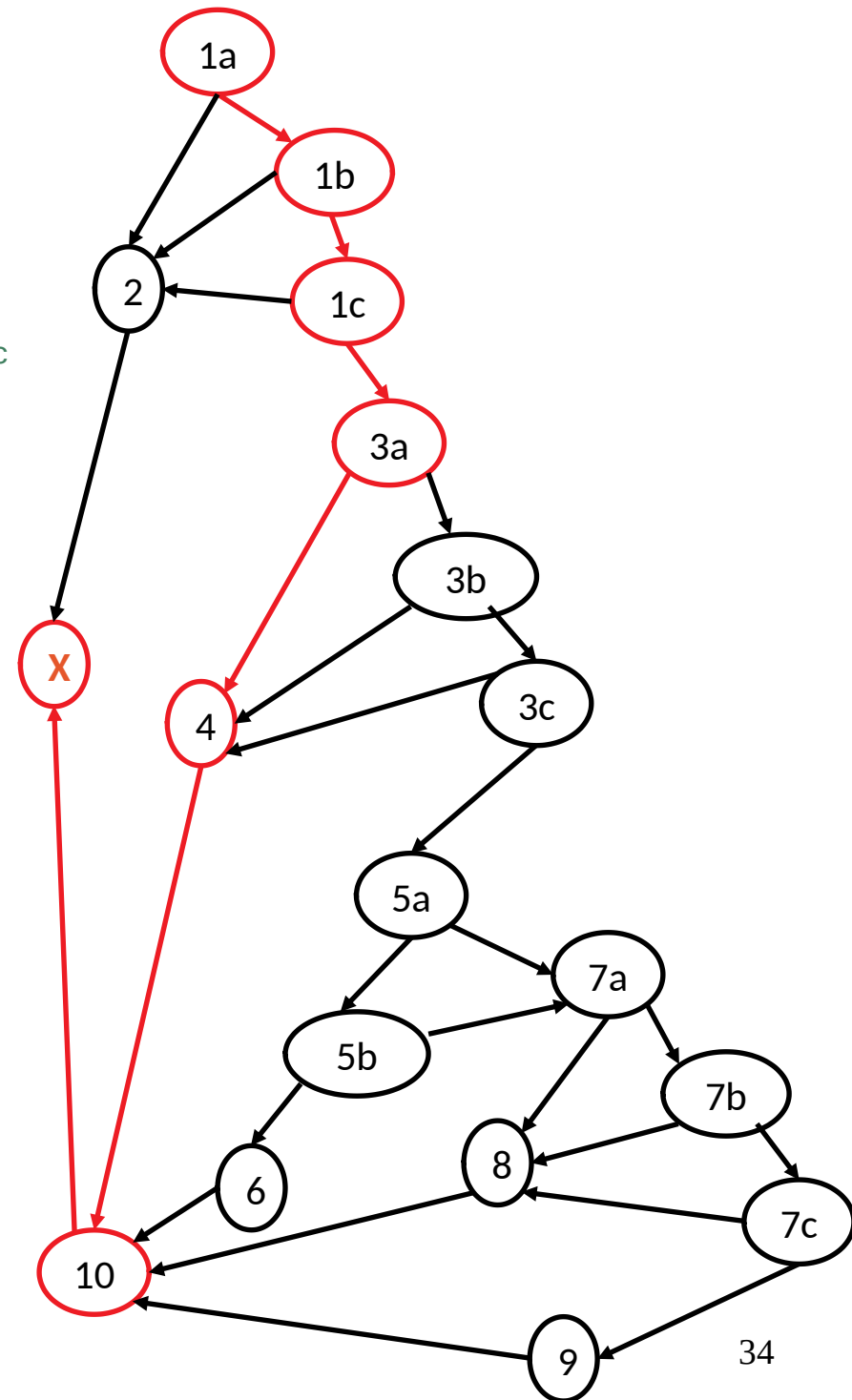
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge (LA \geq LB + LC)$



Exemplo - Triângulo

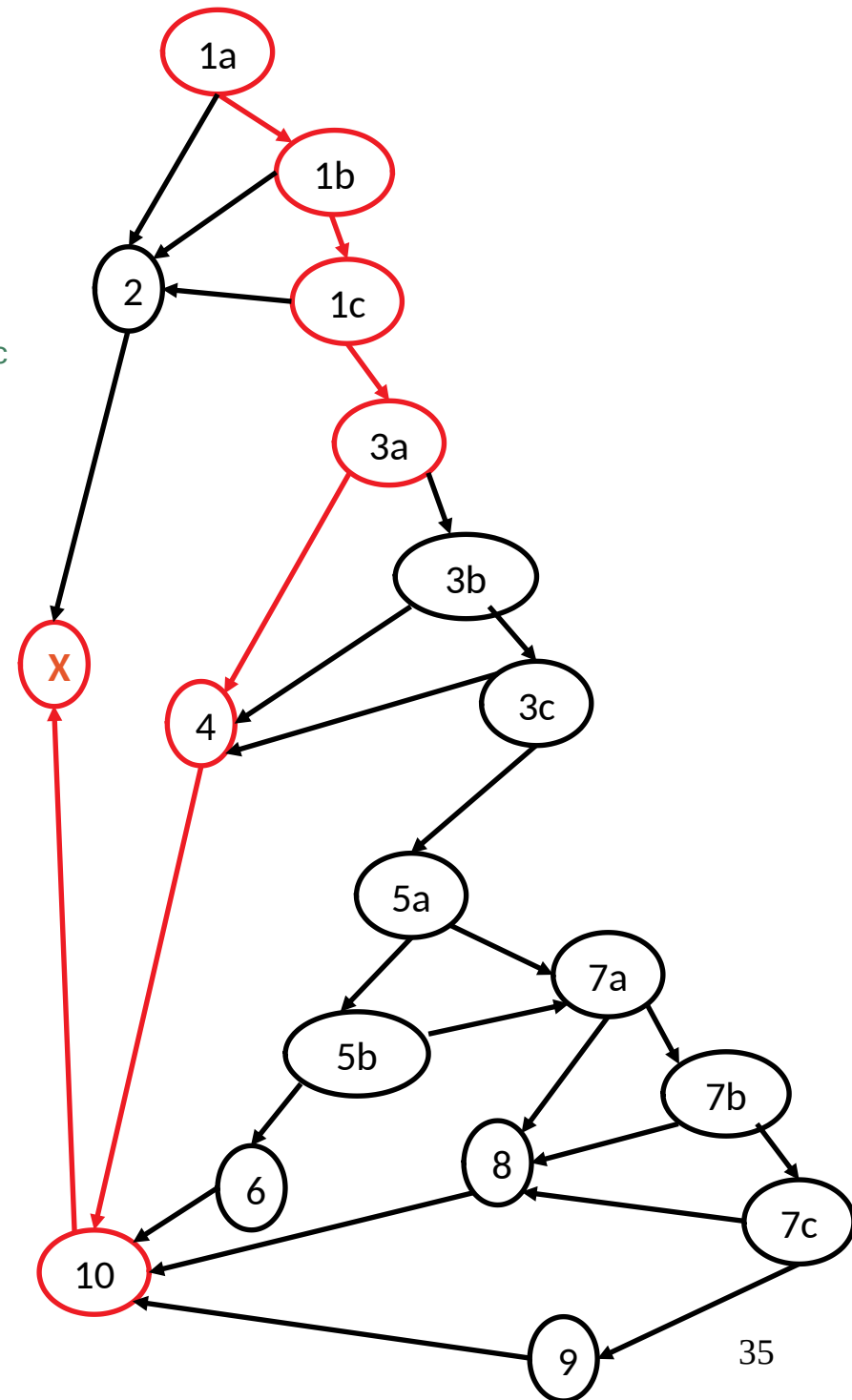
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge (LA \geq LB + LC)$

?



Exemplo - Triângulo

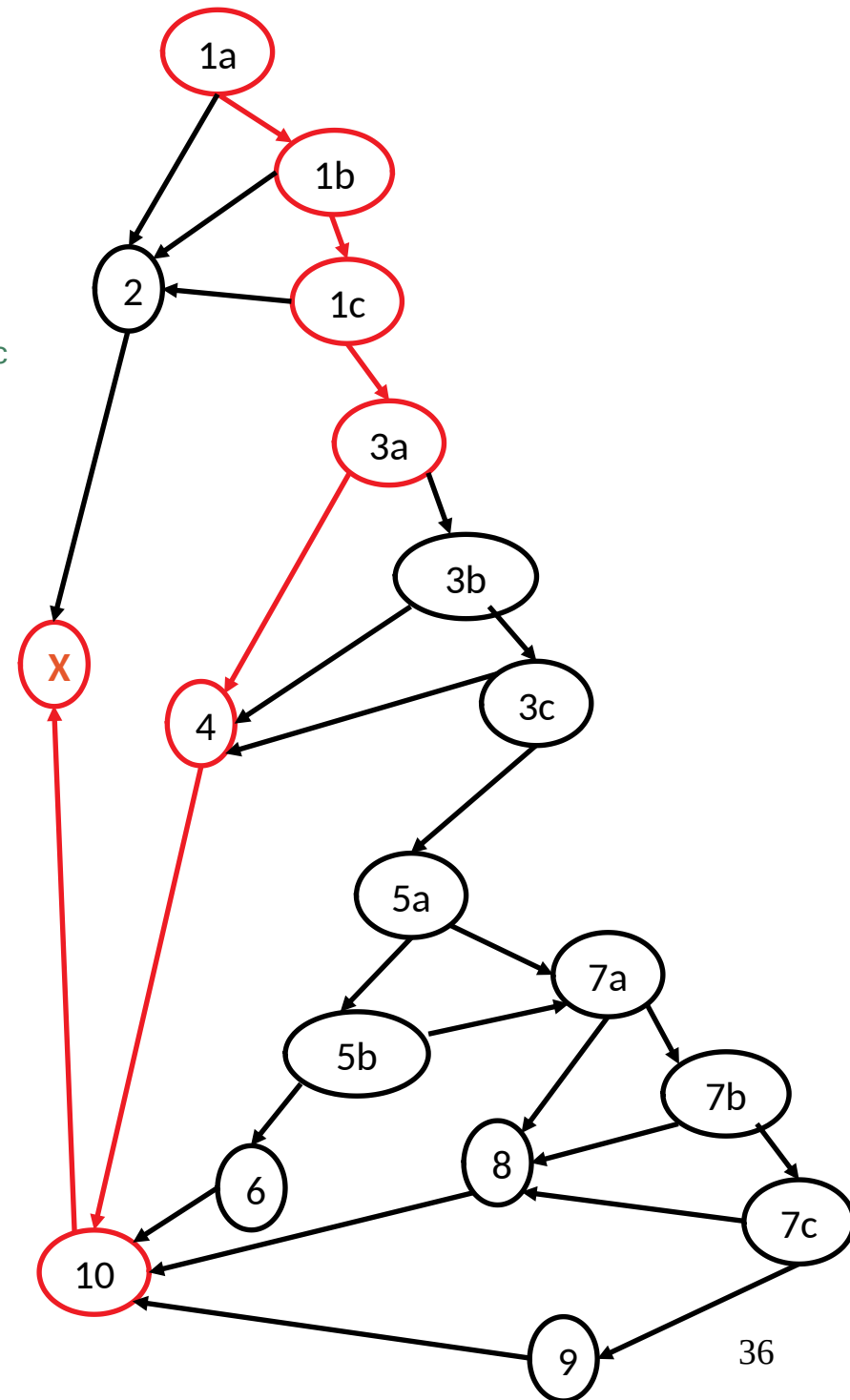
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho: 1a - 1b - 1c - 3a - 4 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge (LA \geq LB + LC)$

5, 2, 2



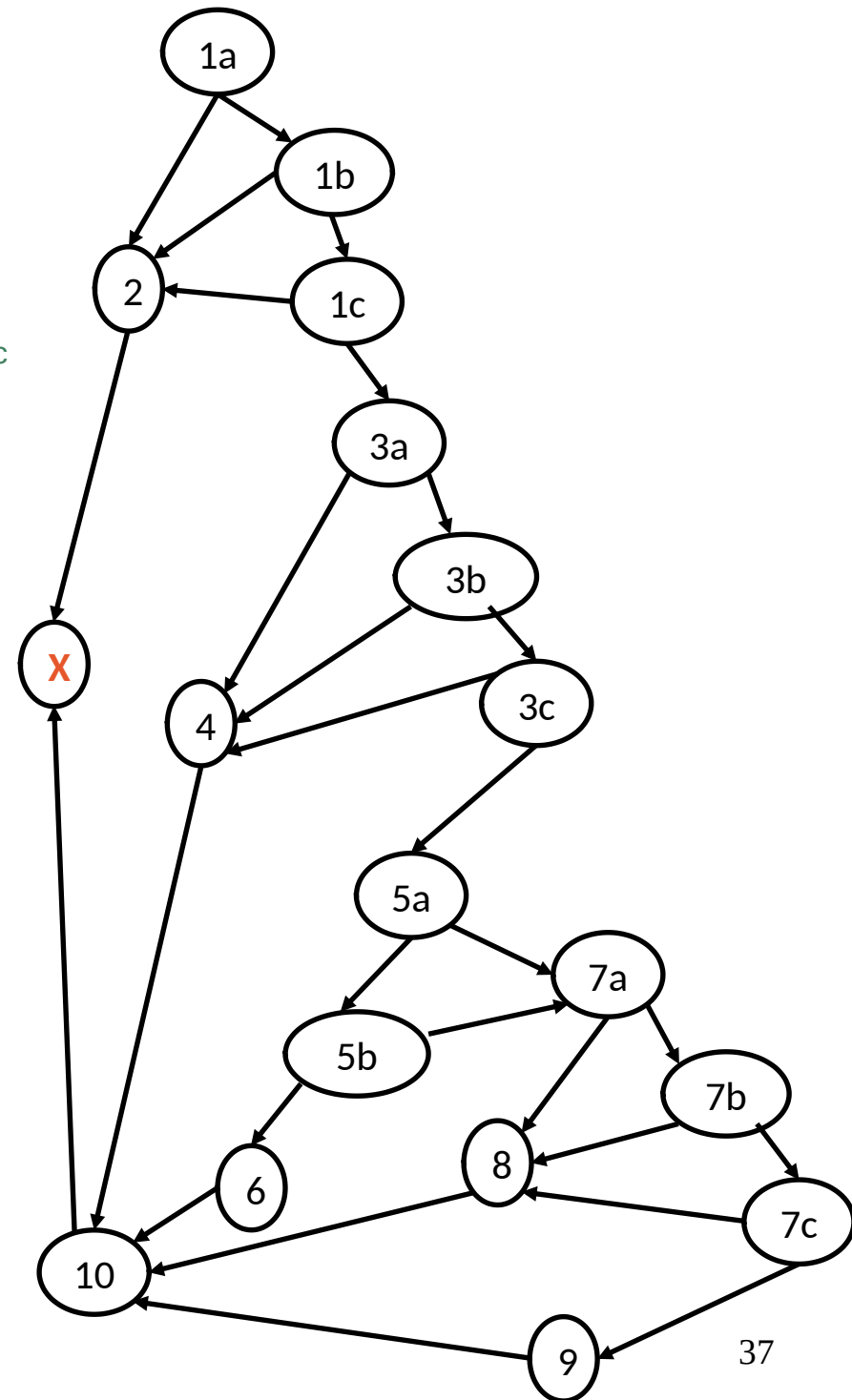
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 7c - 9 - 10 - X



Exemplo - Triângulo

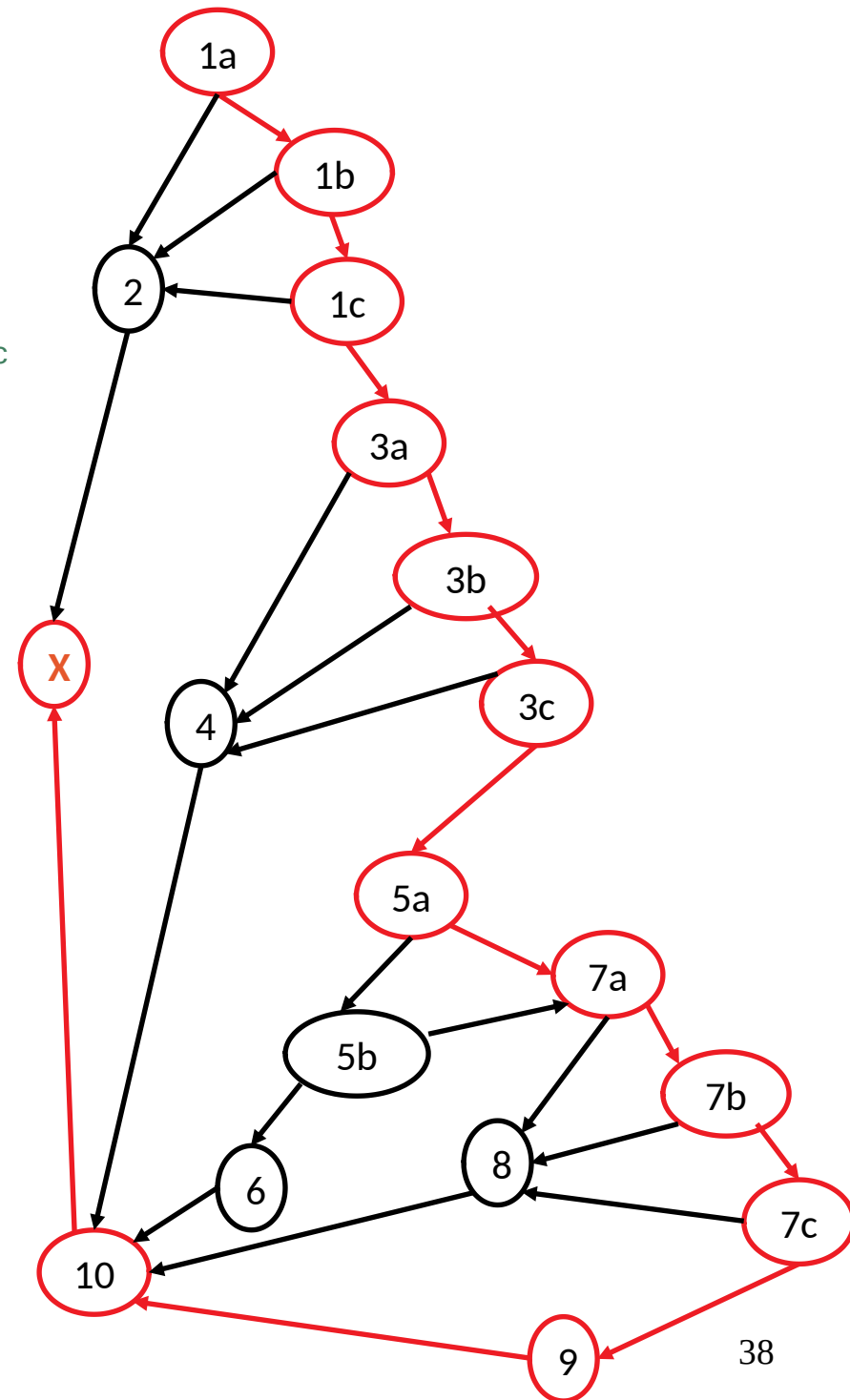
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 7c - 9 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge (LA \neq LB) \wedge (LB \neq LC) \wedge (LA \neq LC)$



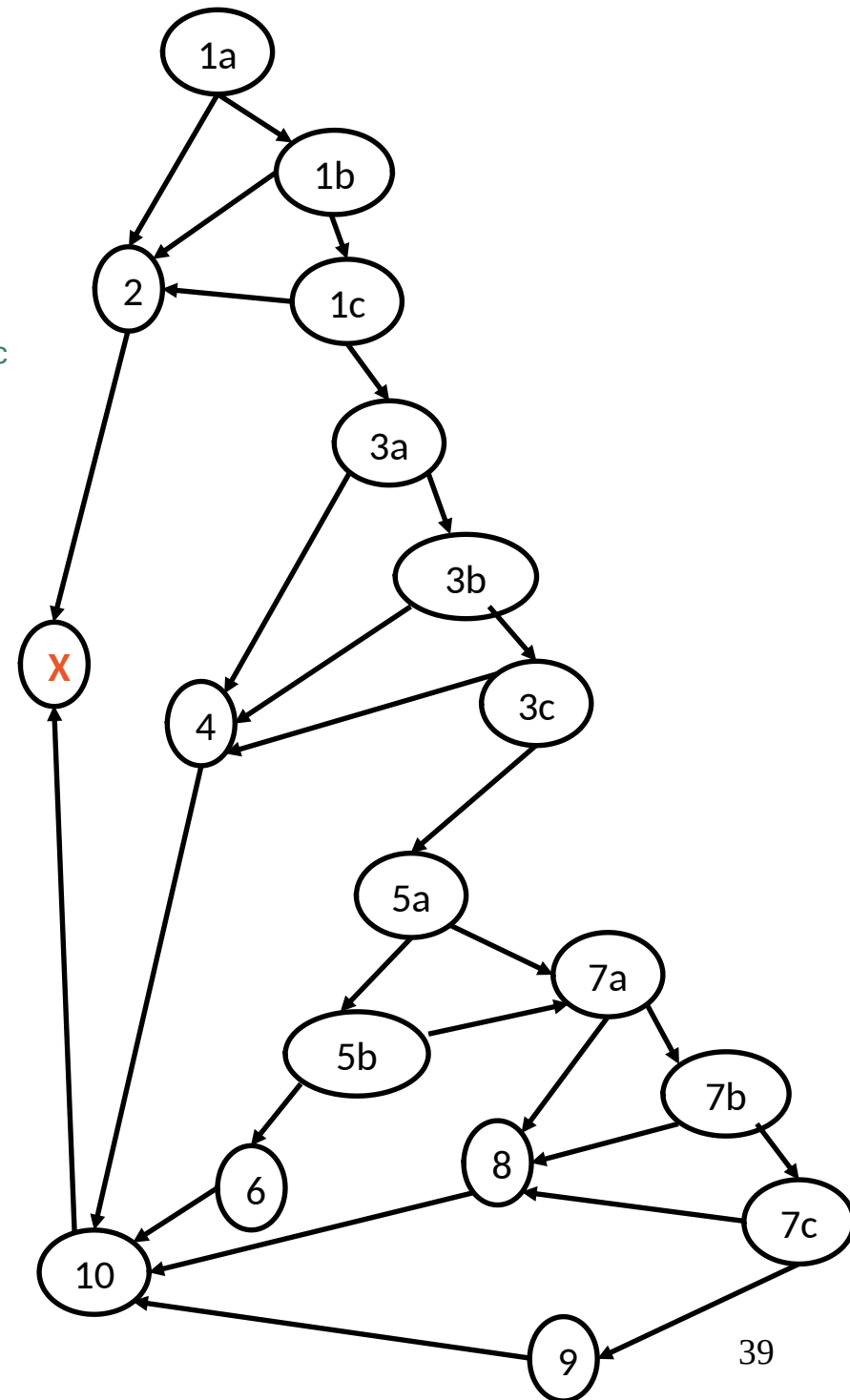
Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X



Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

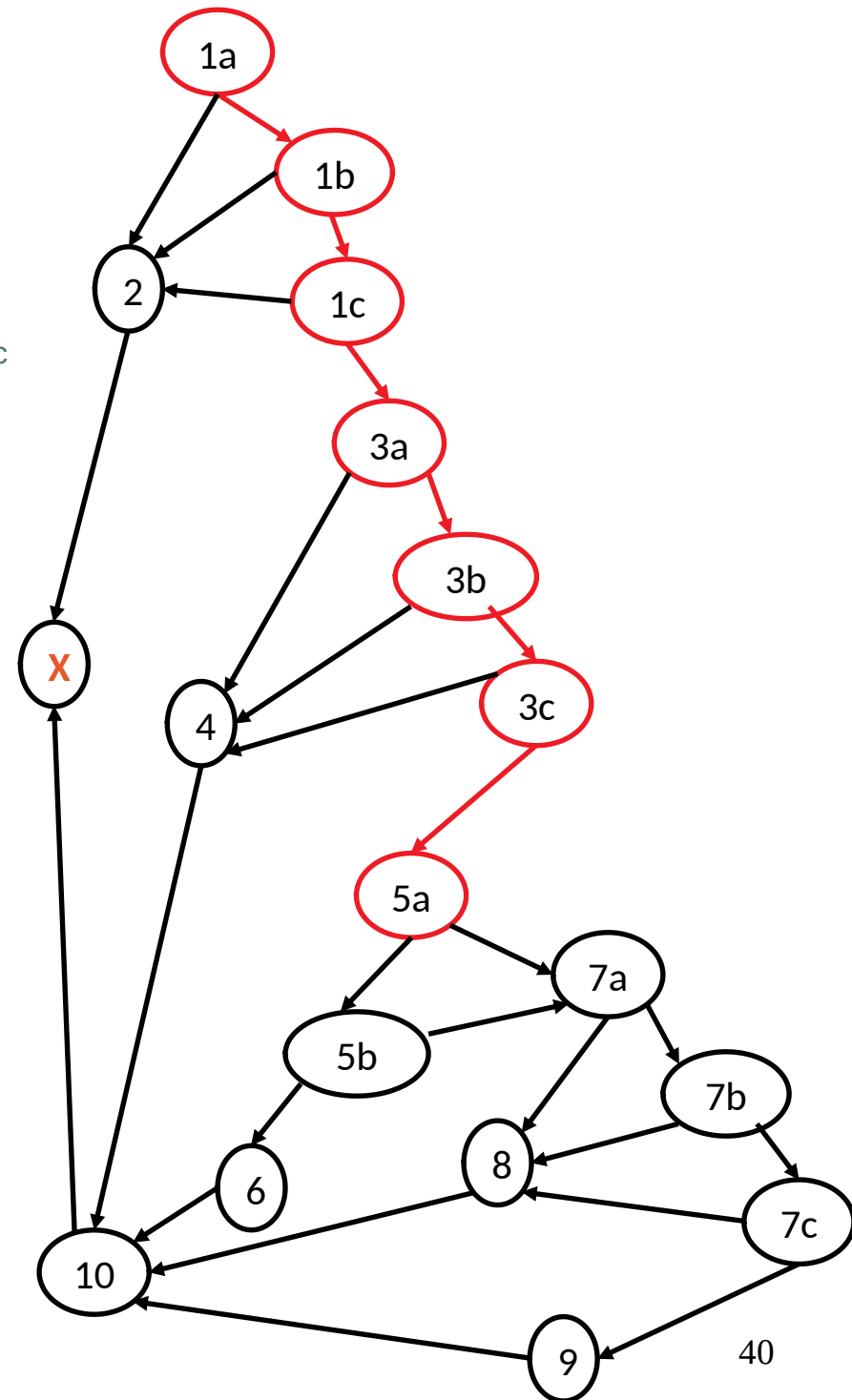
if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$

$(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$



Exemplo - Triângulo

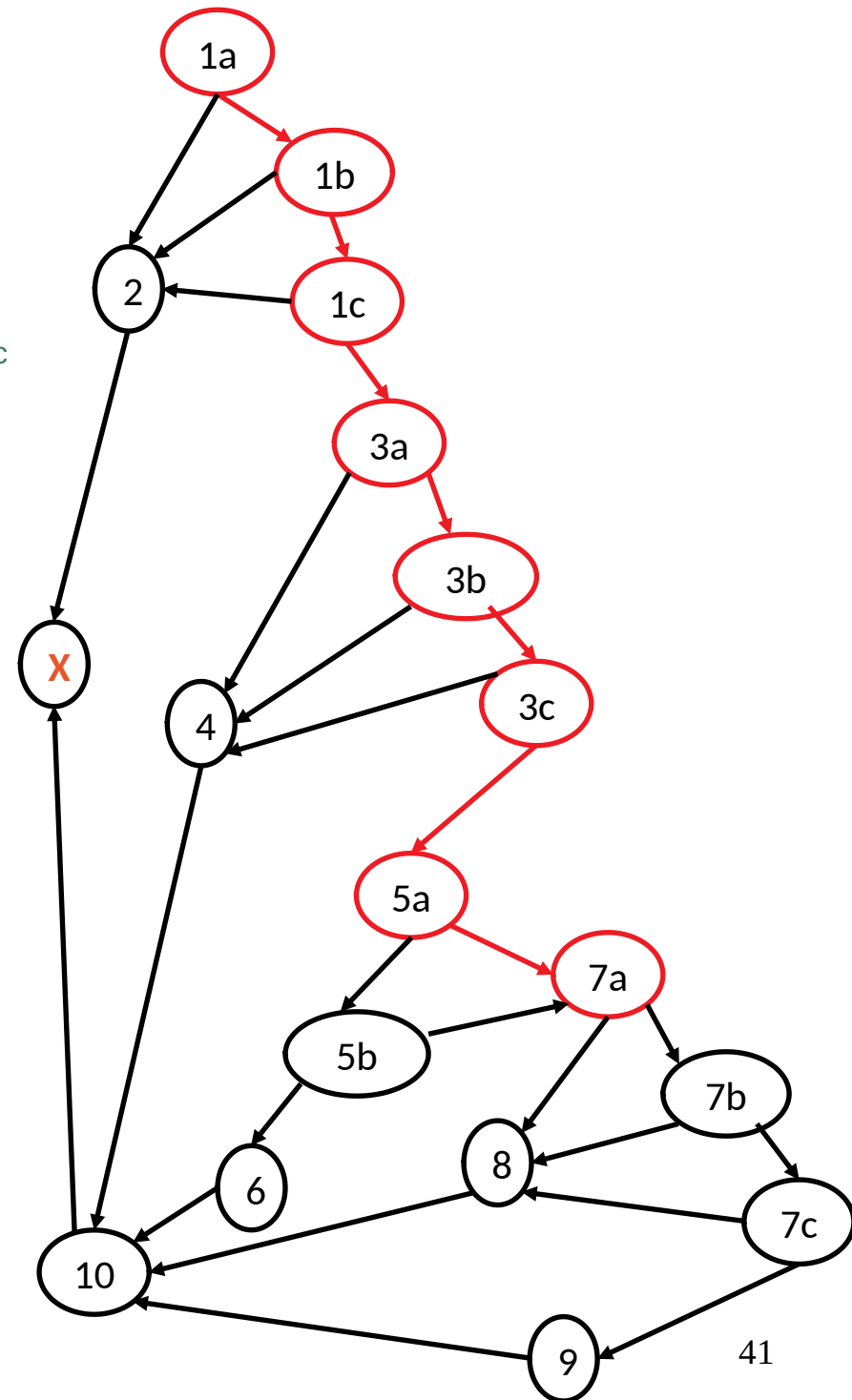
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge$



Exemplo - Triângulo

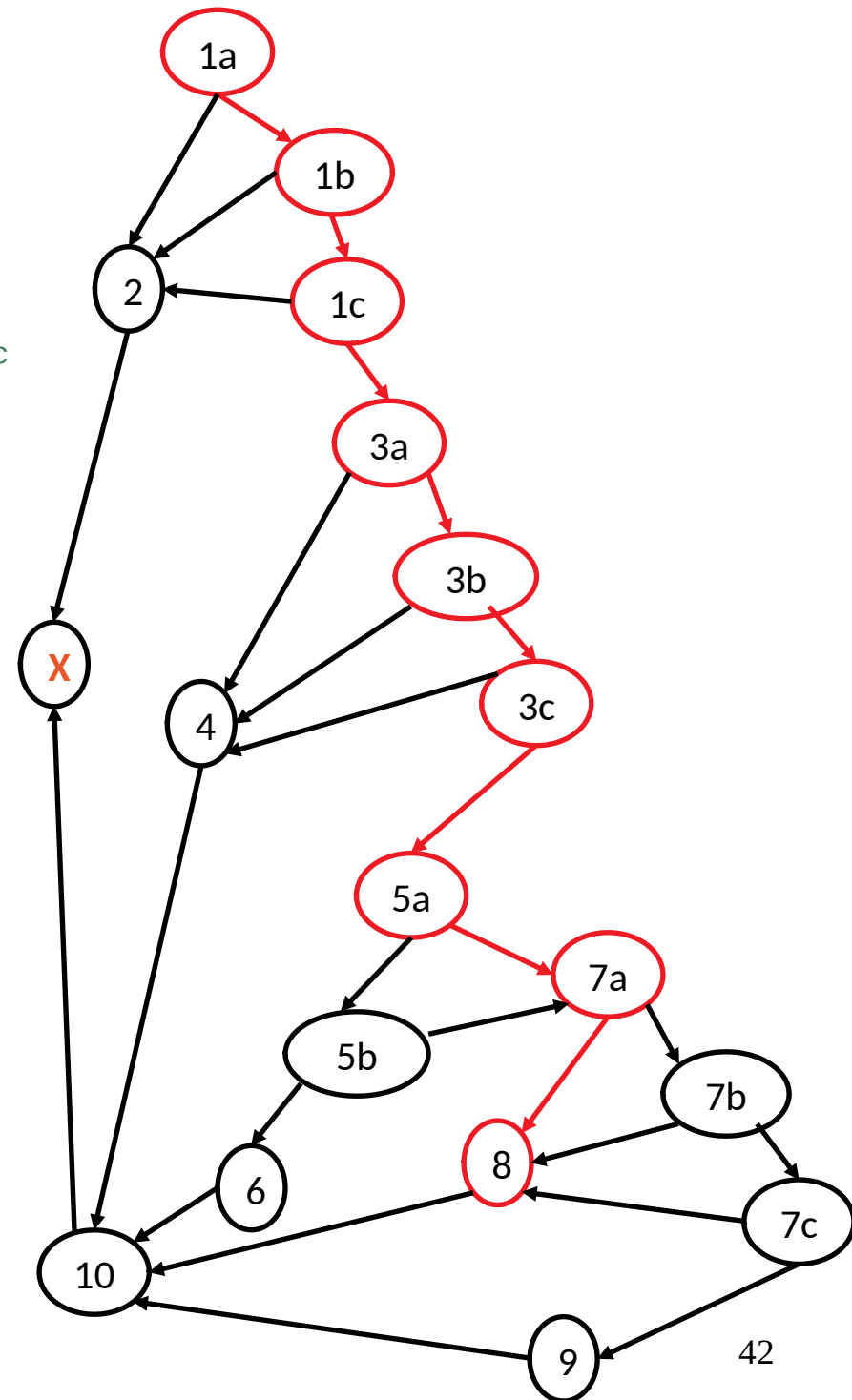
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge (LA == LB)$



Exemplo - Triângulo

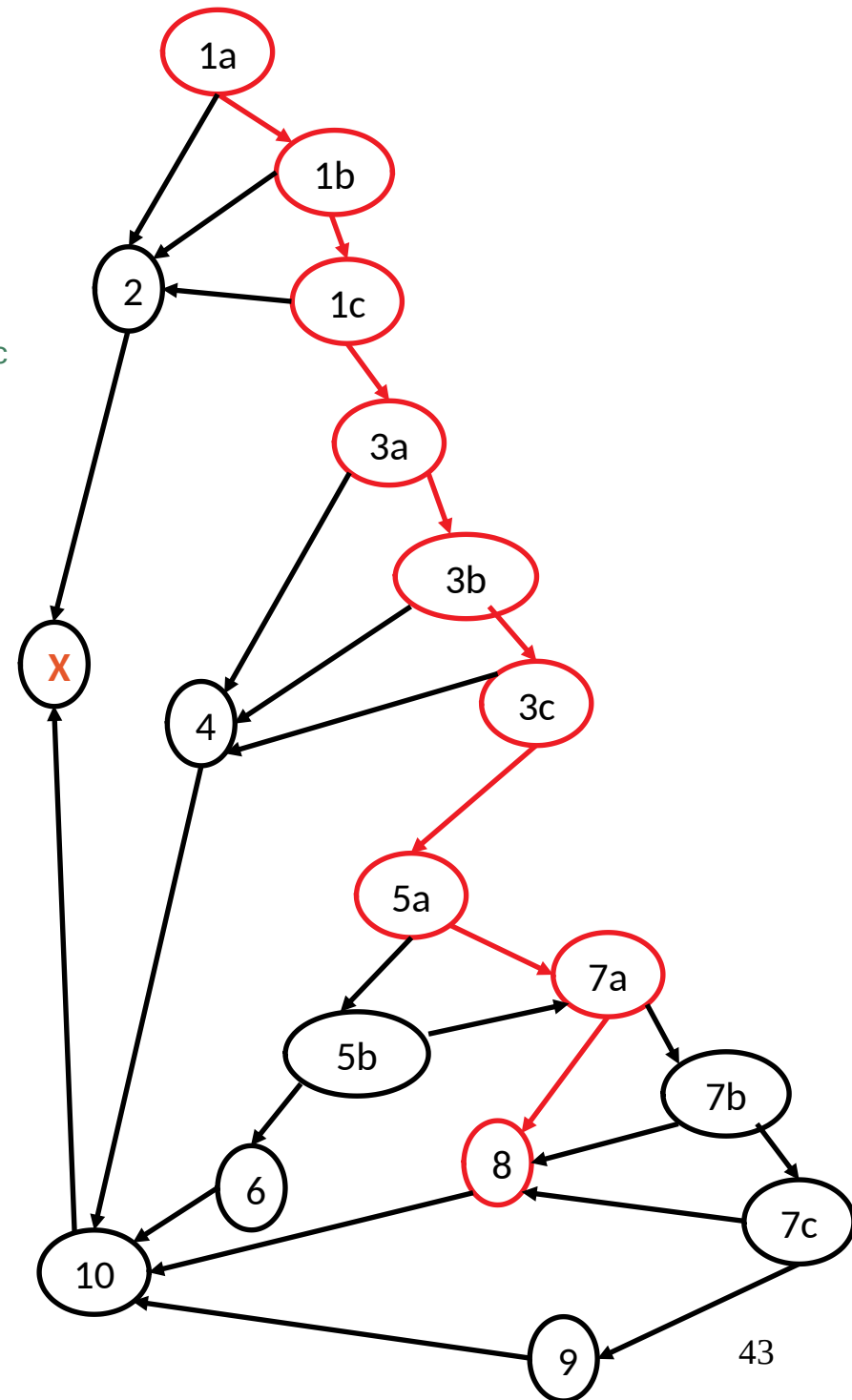
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge (LA == LB)$



Exemplo - Triângulo

```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

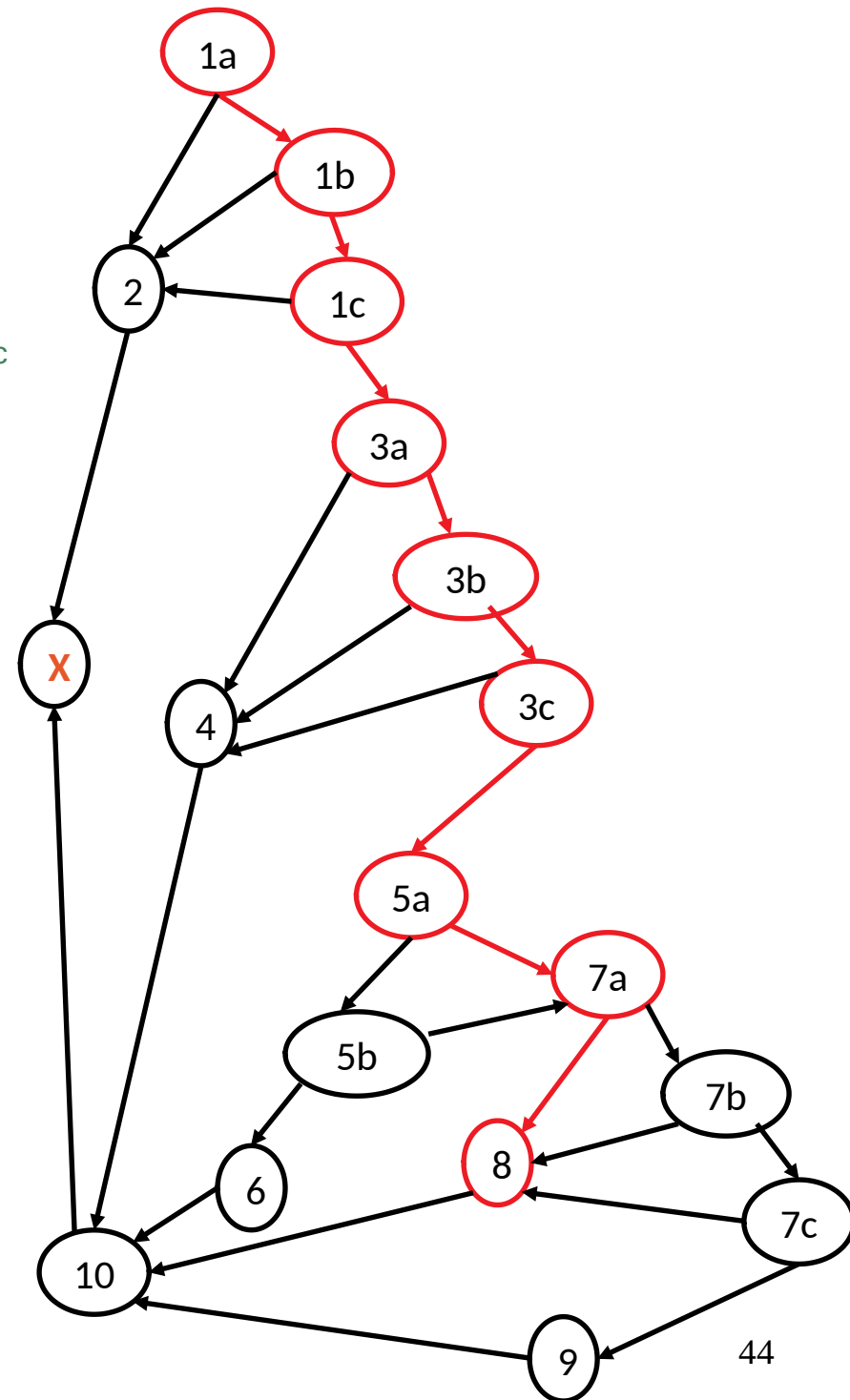
if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 8 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge (LA == LB)$

Caminho não-executável
(Infeasible path)



Execução Simbólica

- Achar soluções para sequências de restrições muito grandes e/ou complexas é uma tarefa que exige muito esforço, além de ser propensa a erros se for realizada manualmente
- A execução simbólica é uma solução simples e adequada para a geração de dados de teste
- A evolução dos solucionadores de restrições (Constraint Solvers) permitiu a automatização desta tarefa

Solucionadores de Restrições

- Constraint Solvers
- Propõem soluções para um conjunto de restrições impostas sobre uma ou mais variáveis
- Surgiram na década de 70/80 para apoiar ambientes e linguagens como o Prolog.
- Na década de 90 foram criados programas e bibliotecas para linguagens procedurais e orientadas a objetos como C, C++ e Java.

Solucionadores de Restrições

- Exemplos:
 - ECLiPSe: C/Prolog (1990)
 - Choco: Java (1999)
 - Geocode: C++ (2005)
 - Miniom: C++ (2006)

Exemplo - Triângulo

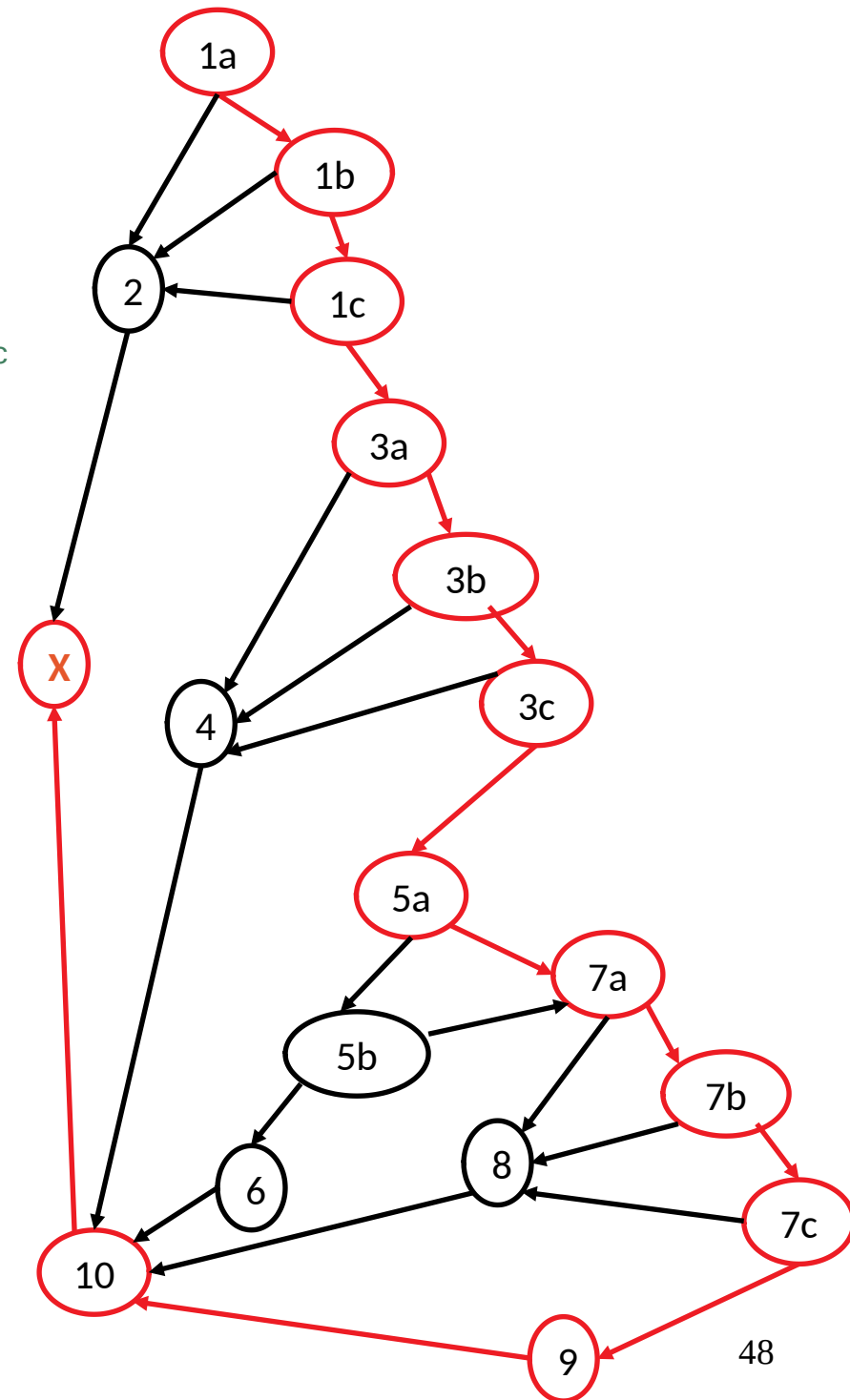
```
classificaTriangulo(int LA, int LB, int LC)
String resposta="";
if (LA<=0 || LB <=0 || LC <=0) //1a, 1b, 1c
    throw new LadoInvalidoException("inválido"); //2

if ( (LA>=LB+LC) || (LB>=LA+LC) || (LC>=LA+LB)) //3a,3b,3c
    resposta = "NAO FORMA TRIANGULO"; //4
else {
    if (LA==LB && LB==LC) //5a,5b
        resposta = "EQUILATERO"; //6
    else {
        if (LA==LB || LB==LC || LA==LC) //7a,7b,7c
            resposta = "ISOSCELES"; //8
        else
            resposta = "ESCALENO"; //9
    }
}
return resposta; //10
}
```

Caminho:

1a - 1b - 1c - 3a - 3b - 3c - 5a - 7a - 7b - 7c - 9 - 10 - X

$(LA > 0) \wedge (LB > 0) \wedge (LC > 0) \wedge$
 $(LA < LB + LC) \wedge (LB < LA + LC) \wedge (LC < LA + LB) \wedge$
 $(LA \neq LB) \wedge (LA \neq LB) \wedge (LB \neq LC) \wedge (LA \neq LC)$



Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Problema do triângulo
 - Entradas válidas
 - $LA > 0$
 - $LB > 0$
 - $LC > 0$
 - Para formar um triângulo:
 - $LA < LB + LC$
 - $LB < LA + LC$
 - $LC < LA + LB$

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Problema do triângulo
 - Para ser escaleno:
 - $LA \neq LB$
 - $LB \neq LC$
 - $LA \neq LC$

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Criação de variáveis

```
IntegerVariable LA = Choco.makeIntVar("LA", 1, 1000);
```

```
IntegerVariable LB = Choco.makeIntVar("LB", 1, 1000);
```

```
IntegerVariable LC = Choco.makeIntVar("LC", 1, 1000);
```

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Criação de expressões aritméticas:

//LB + LC

```
IntegerExpressionVariable E1 = Choco.plus(LB,LC);
```

//LA + LC

```
IntegerExpressionVariable E2 = Choco.plus(LA,LC);
```

//LA + LB

```
IntegerExpressionVariable E3 = Choco.plus(LA,LB);
```

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Criação das restrições:

```
Constraint C1 = Choco.lt(LA, E1); // LA < E1 (LB+LC)
```

```
Constraint C2 = Choco.lt(LB, E2); // LB < E2 (LA+LC)
```

```
Constraint C3 = Choco.lt(LC, E3); // LC < E3 (LA+LB)
```

```
Constraint C4 = Choco.neq(LA, LB); // LA != LB
```

```
Constraint C5 = Choco.neq(LB, LC); // LB != LC
```

```
Constraint C6 = Choco.neq(LA, LC); // LA != LC
```

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Adiciona restrições ao modelo criado:

```
Model M = new CPModel();
```

```
M.addConstraint(C1);
```

```
M.addConstraint(C2);
```

```
M.addConstraint(C3);
```

```
M.addConstraint(C4);
```

```
M.addConstraint(C5);
```

```
M.addConstraint(C6);
```

Solucionadores de Restrições

- Exemplo: biblioteca Choco
- Procura solução:

```
Solver S = new CPSolver();
```

```
S.read(M);
```

```
S.solve() ;
```

```
System.out.println("Lado A =" + s.getVar(LA).getVal());
```

```
System.out.println("Lado B =" + s.getVar(LB).getVal());
```

```
System.out.println("Lado C =" + s.getVar(LC).getVal());
```

Geração Automática de Dados de Teste

EXECUÇÃO SIMBÓLICA

+

SOLUCIONADORES DE RESTRIÇÕES

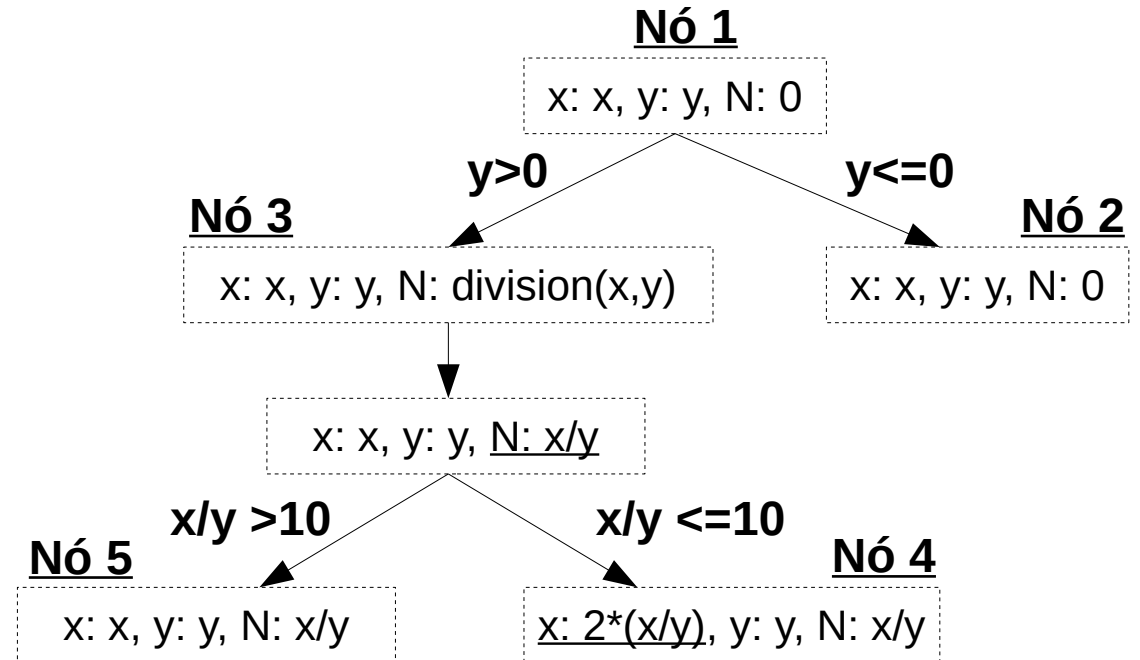
Geração Automática de Dados de Teste

- Execução Simbólica: gera sequências de restrições que devem ser satisfeitas para que um caminho seja executado
- Solucionadores de Restrições: encontram uma possível solução para a sequência de restrição gerada pela execução simbólica

Geração Automática de Dados de Teste

```
public int division(int x, int y)
    return x/y;
}
```

```
public int testMe(int x, int y)
    int N=0;
    if (y>0){
        N = division(x,y);
        if (N > 10)
            return N;
        else
        {
            x = 2*N;
            return x;
        }
    }
}
```



Para descobrir quais valores de entrada (x e y) são necessários para executar todos os caminhos, por exemplo, basta verificar a sequência de restrições e achar uma solução.

Caminho a: 1, 2.
Restrições: $y \leq 0$.
Possível solução $y=0$

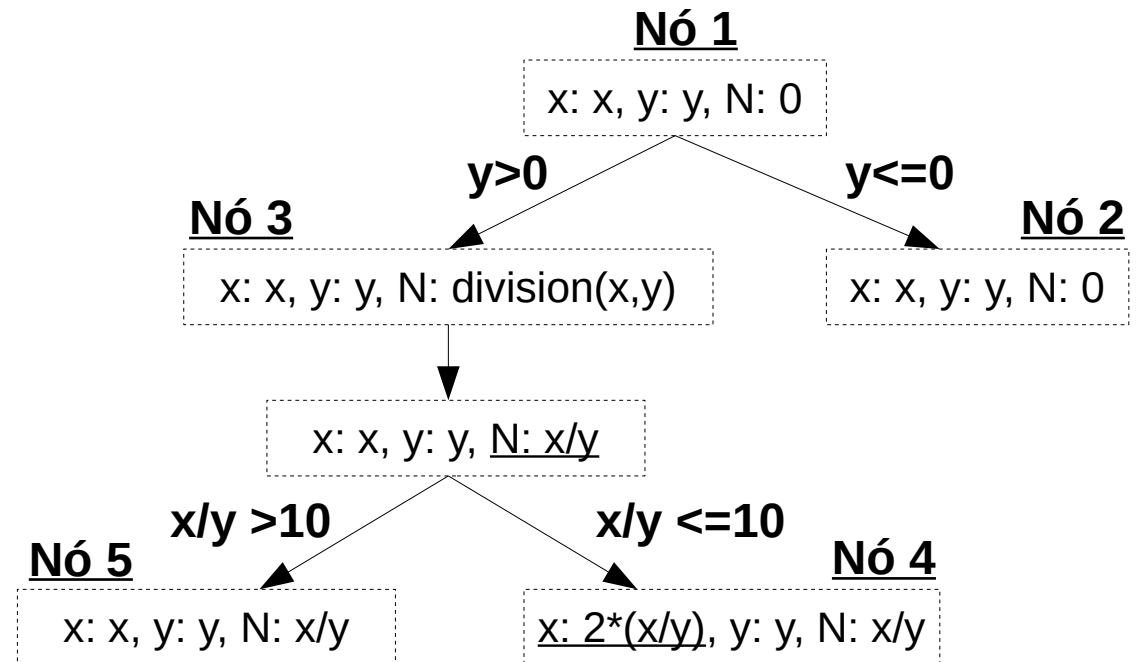
Caminho b: 1, 3, 5.
Restrições: $y > 0 \wedge x/y > 10$
Possível solução: $x=11, y=1$

Caminho c: 1, 3, 4.
Restrições: $y > 0 \wedge x/y \leq 10$
Possível solução: $x=10, y=1$ ⁵⁸

Geração Automática de Dados de Teste

```
public int division(int x, int y)
return x/y;
}
```

```
public int testMe(int x, int y)
int N=0;
if (y>0){
    N = division(x,y);
    if (N > 10)
        return N;
    else
    {
        x = 2*N;
        return x;
    }
}
```



Para descobrir quais valores de entrada (x e y) são necessários para executar todos os caminhos, por exemplo, basta verificar a sequência de restrições e achar uma solução.

Caminho a: 1, 2.

Restrições: $y \leq 0$.

Possível solução $y=0$

Caminho b: 1, 3, 5.

Restrições: $y > 0 \wedge x/y > 10$

Possível solução: $x=11, y=1$

Caminho c: 1, 3, 4.

Restrições: $y > 0 \wedge x/y \leq 10$

Possível solução: $x=10, y=1$

Limitações

- Execução Simbólica
 - Caminhos não executáveis
 - Explosão do número de caminhos
 - Antecipação de laços
 - Recursos externos
 - Ponteiros e estruturas complexas

Limitações

- Solucionadores de Restrições
 - Expressões complexas
 - Tempo de resposta
 - Sequências sem solução

Concolic testing

- Concolic = Concrete + Symbolic
- Mapeamento simbólico e concreto da execução de um programa com dados de teste iniciais escolhidos randomicamente
- A última decisão de desvio de fluxo de controle é negada e um valor concreto definido com base em execução simbólica e solucionadores de restrições para que o ramo oposto seja executado

Ferramentas (Symbolic e Concolic)

- DART
- Cute e JCute
- EXE
- Klee
- RANDOOP
- PEX

Referências

- J. C. King. Symbolic execution and program testing. Communications of ACM, 19(7):385-394, 1976.
- P. Godefroid. Test Generation Using Symbolic Execution. Annual Conference on Foundations of Software Technology and Theoretical Computer Science, pages 24-33, 2012.
- S. Galler and B. Aichernig. Survey on test data generation tools. International Journal on Software Tools for Technology Transfer, pages 1-25, 2013.

ACH2028 – Qualidade de Software

Aula 09 – Geração Automática de Testes Execução Simbólica

Prof. Marcelo Medeiros Eler
marceloeler@usp.br