

ENGENHARIA DE SOFTWARE III

Prof. Iara Carnevale de Almeida

Conforme Pressman e Maxim(2016), “usar **métricas** para desenvolver estratégias para a melhoria de processo de software e, como consequência, a qualidade no produto final”

É preciso avaliar duplicidade de código, métodos e classes, **complexidade ciclomática elevada**, ausência de testes unitários entre outros pontos.

A **auditoria** é uma técnica eficaz para a identificação de falhas de segurança, uma vez que grande parte das vulnerabilidades das aplicações têm como causa falhas de implementação.

A **análise estática** de código é uma das práticas que analisa a qualidade do código-fonte. Esta verificação é realizada antes mesmo que haja execução do software.

Complexidade Ciclométrica (CC) é medida de complexidade de um software. O cálculo de complexidade ciclométrica está relacionado a quantos desvios um programa possui. Cada um dos comandos if, switch, for, while, pode ser considerado um desvio do fluxo principal do código.

Toda funcionalidade possui o valor padrão “1”.

Assim que são incluídos novos desvios este valor vai aumentando.

Deve-se procurar manter este valor baixo, para garantir que a manutenção de código seja mais fácil.

Passos necessários:

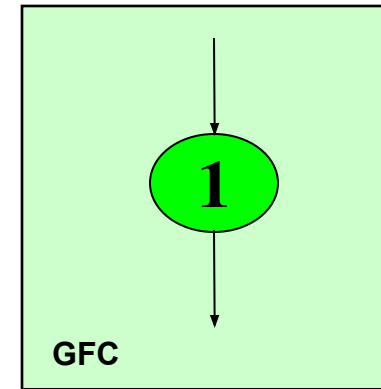
- Desenhar o grafo de fluxo de controle (GFC) para um fragmento de código.
- Computar a *complexidade ciclomática* para o GFC gerado.

Como gerar o GFC?

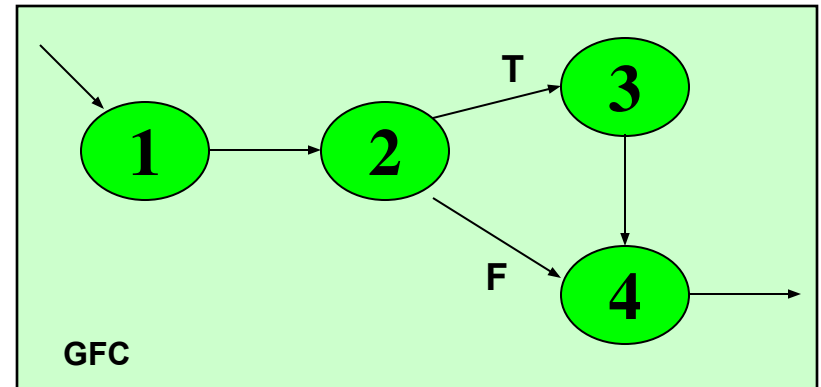
Grafo de Fluxo de Controle (GFC)

```
Statement1;  
Statement2;  
Statement3;  
Statement4;
```

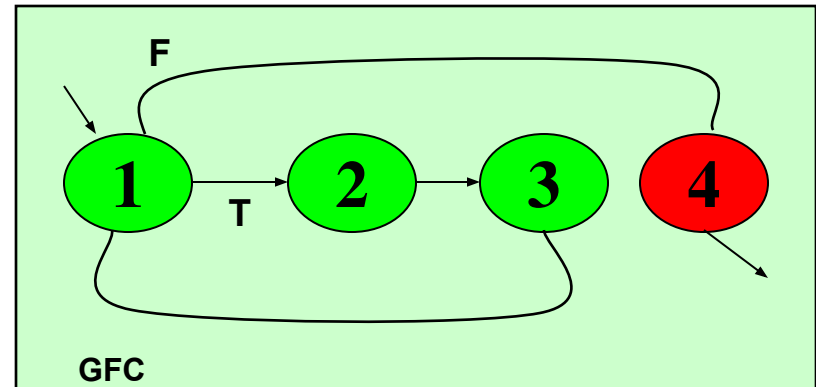
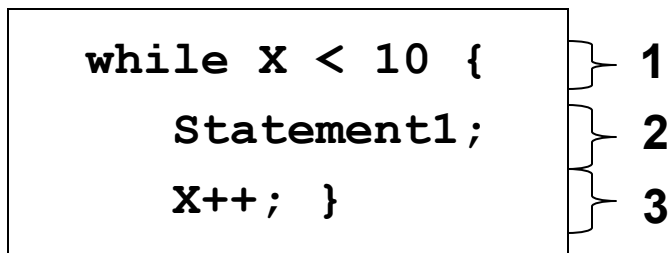
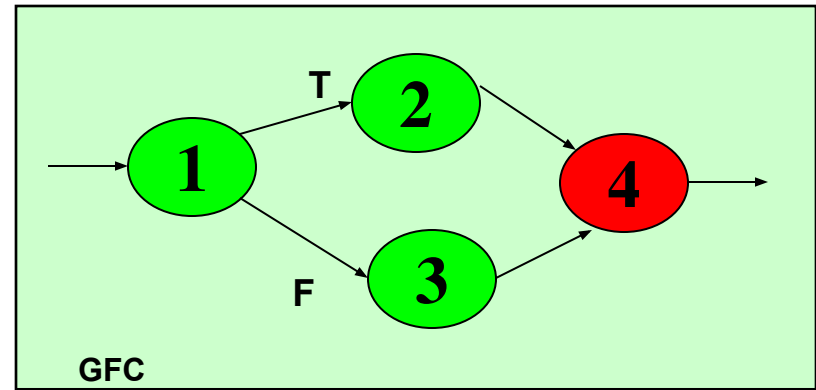
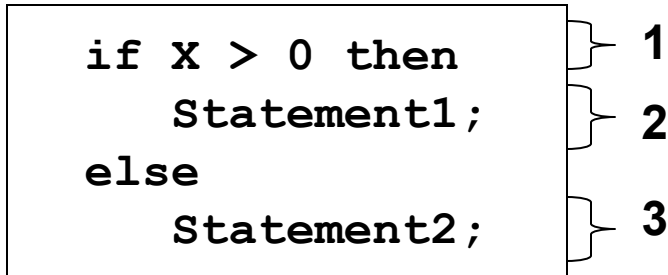
Diversos comandos podem ser representados por um único nodo.



```
Statement1;  
Statement2; 1  
  
if X < 10 then 2  
    Statement3; 3  
  
Statement4; 4
```



Grafo de Fluxo de Controle (GFC)



Nodo 4 é um Nodo Lógico, veja próximo slide ...

Grafo de Fluxo de Controle (GFC)

- Um GFC deve ter
 - 1 arco de entrada, e
 - 1 arco de saída.
- Todos os nodos devem ter
 - Ao menos um arco de entrada, e
 - Ao menos um arco de saída
- Um **Nodo Lógico** representa a ausência de comando e pode ser usado como um ponto de junção. Representa um fechamento lógico.

Exercício: compreender ...

```
min = A[0];  
I = 1;  
  
while (I < N) {  
    if (A[I] < min)  
        min = A[I];  
    I = I + 1;  
}  
print min
```

Qual o objetivo deste programa?

O que a variável N representa?

Quais são os nodos para o GFC?

Exercício: elaborar o GFC

```
min = A[0];  
I = 1;  
  
while (I < N) {  
    if (A[I] < min)  
        min = A[I];  
    I = I + 1;  
}  
print min
```

Diagram illustrating the flow of the code with numbered steps:

- 1
- 2
- 3
- 4
- 5
- 6

Exercício: elaborar o GFC

```
min = A[0];
```

```
I = 1;
```

```
while (I < N) {
```

```
    if (A[I] < min)
```

```
        min = A[I];
```

```
    I = I + 1;
```

```
}
```

```
print min
```

1

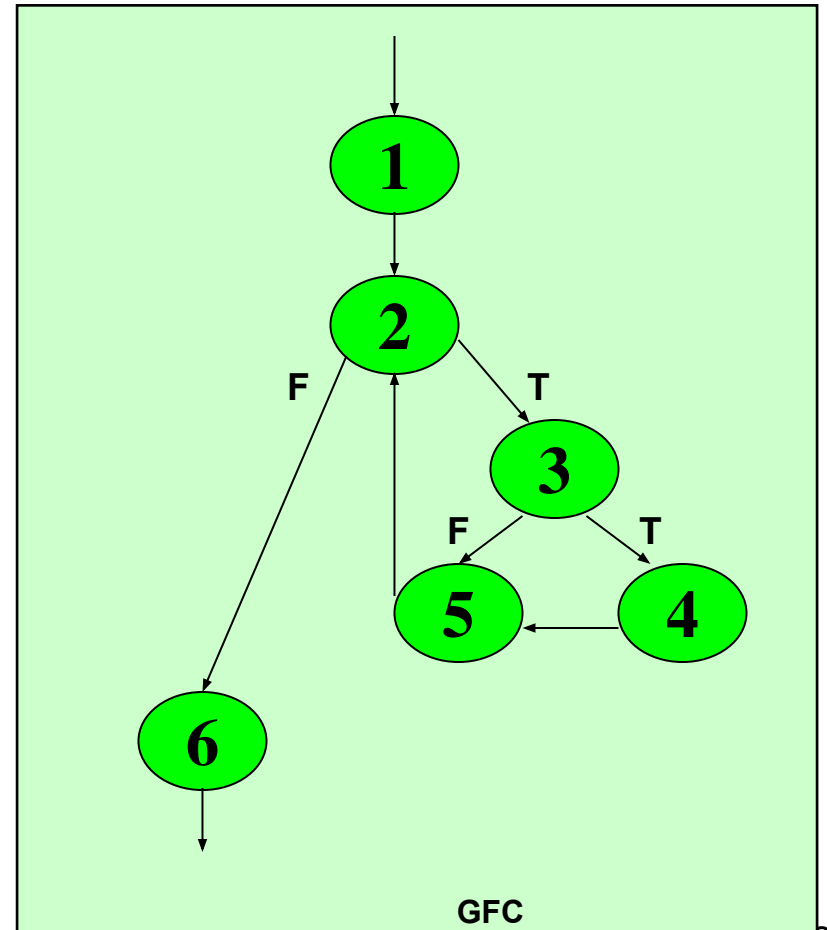
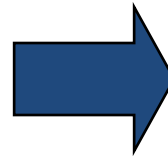
2

3

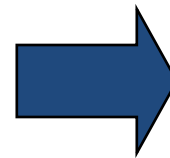
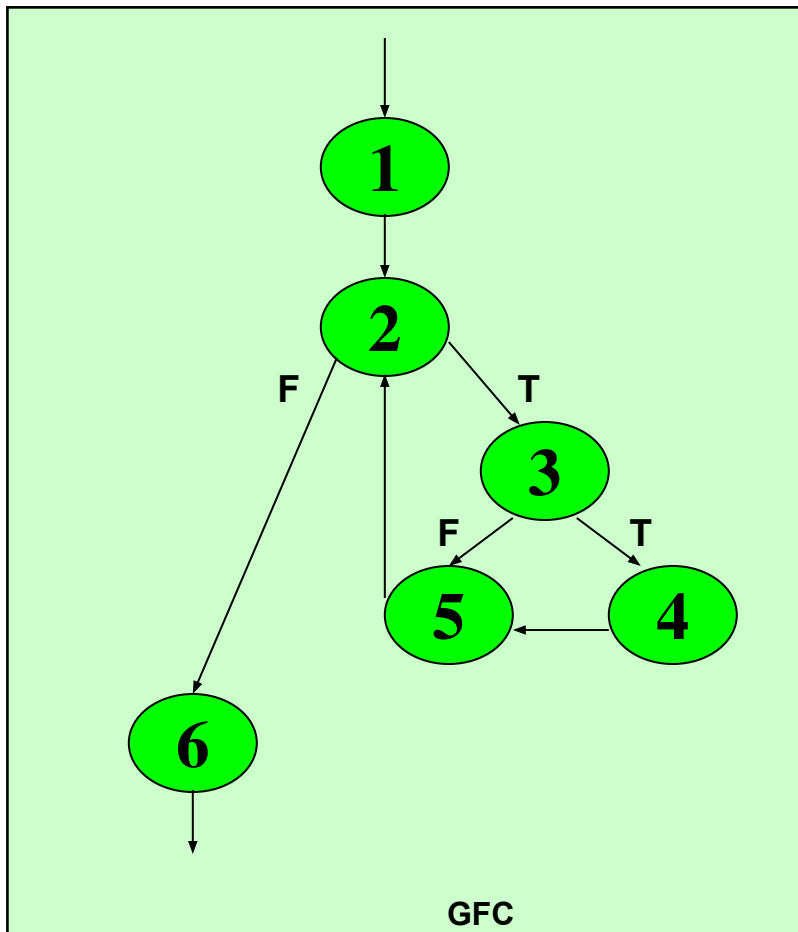
4

5

6



Complexidade Ciclomática

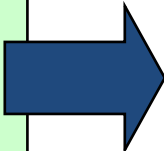


Núm.Regões do grafo
OU
Núm.Predicados + 1
OU
 $\text{Núm.Setas} - \text{Núm.nós} + 2$

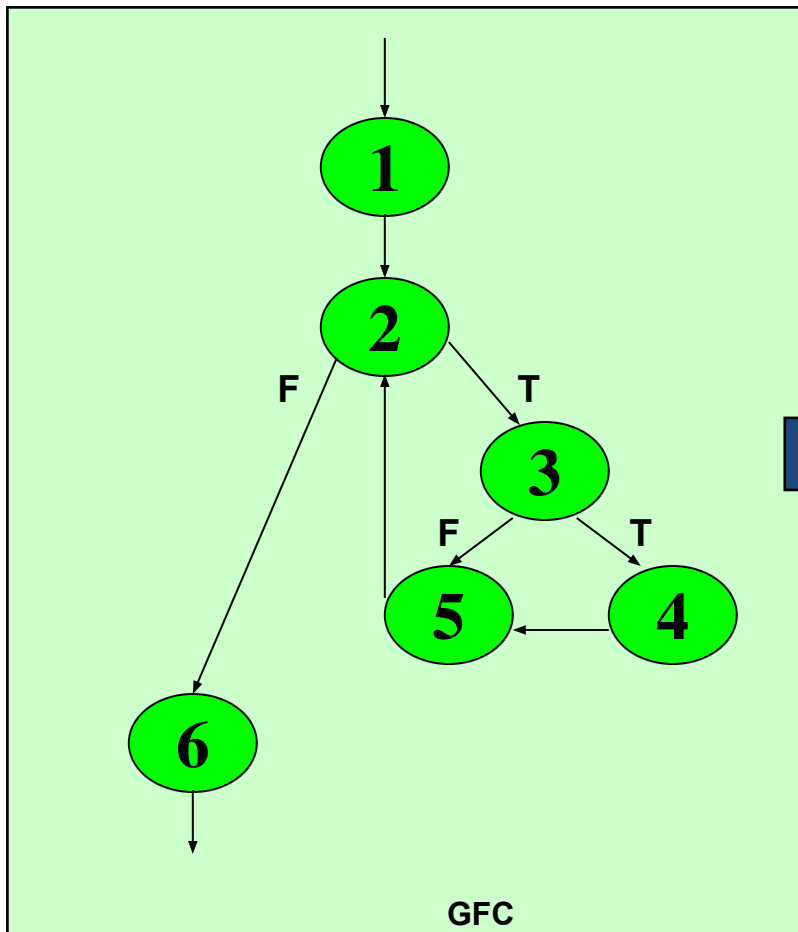
Complexidade Ciclomática

Região: é a região fechada em um GFC. Não esquecer a região mais externa!

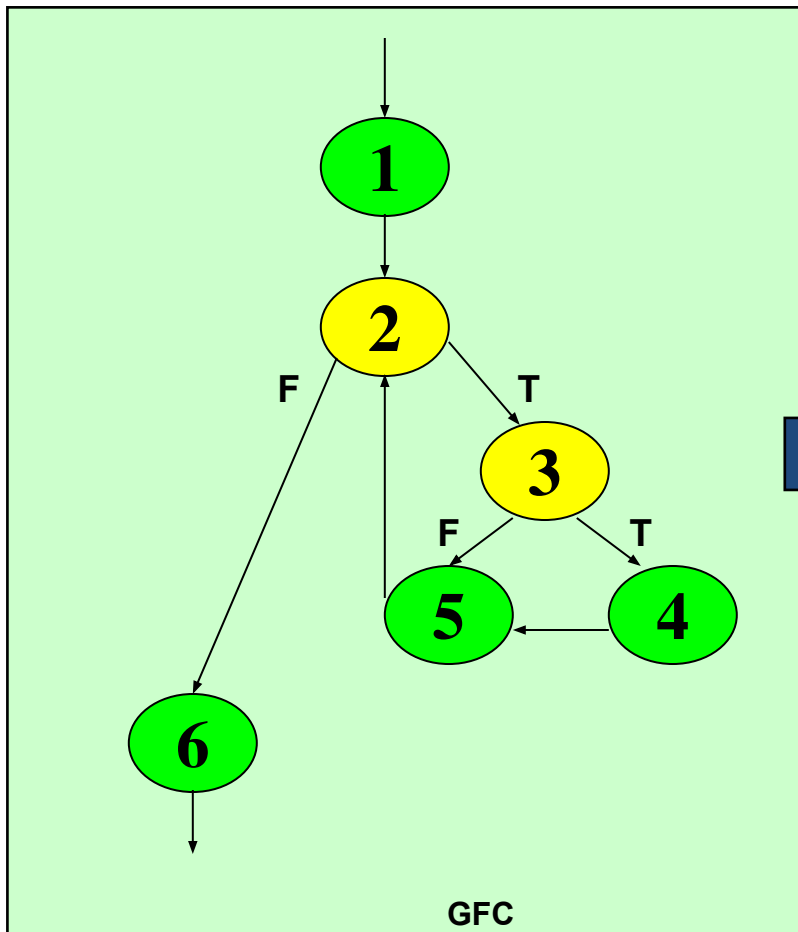
Regiões deste GFC?

- 
1. *toda região*
 2. *entre os nodos 2, 3 e 5*
 3. *entre os nodos 3, 4 e e 5*

Logo, a complexidade ciclomática é 3.



Complexidade Ciclômática



Predicados: Nodos que correspondem ao comando condicional do programa, são os com saídas múltiplas.

GFC possui **2** predicados (nodos 2 e 3)

Complexidade ciclômática
 $= 2 + 1$
 $= 3$

Steve McConnell no livro "Code Complete" criou uma tabela com os seguintes valores limites para CC:

- **0-5**: código está, provavelmente, ok;
- **6-10**: procure simplificar o código;
- **>10**: quebre o código em dois e insira uma chamada da primeira parte para a segunda.


```
public int identificaTriangulo2(int a, int b,  
int c) {  
    if ((a < b + c) && (b < a + c) && (c < b +  
a)) {  
        if ((a == b) && (b == c))  
            return (int)tipos.EQUILATERO;  
        else  
            if ((a != b) && (a != c) && (b != c))  
                return (int)tipos.ESCALENO;  
            else return (int)tipos.ISOSCELES;  
        }  
    return (int) tipos.INVALIDO;  
}
```

Uma ferramenta
muito utilizada
para gerar
métricas de
código.

Calcula CC !

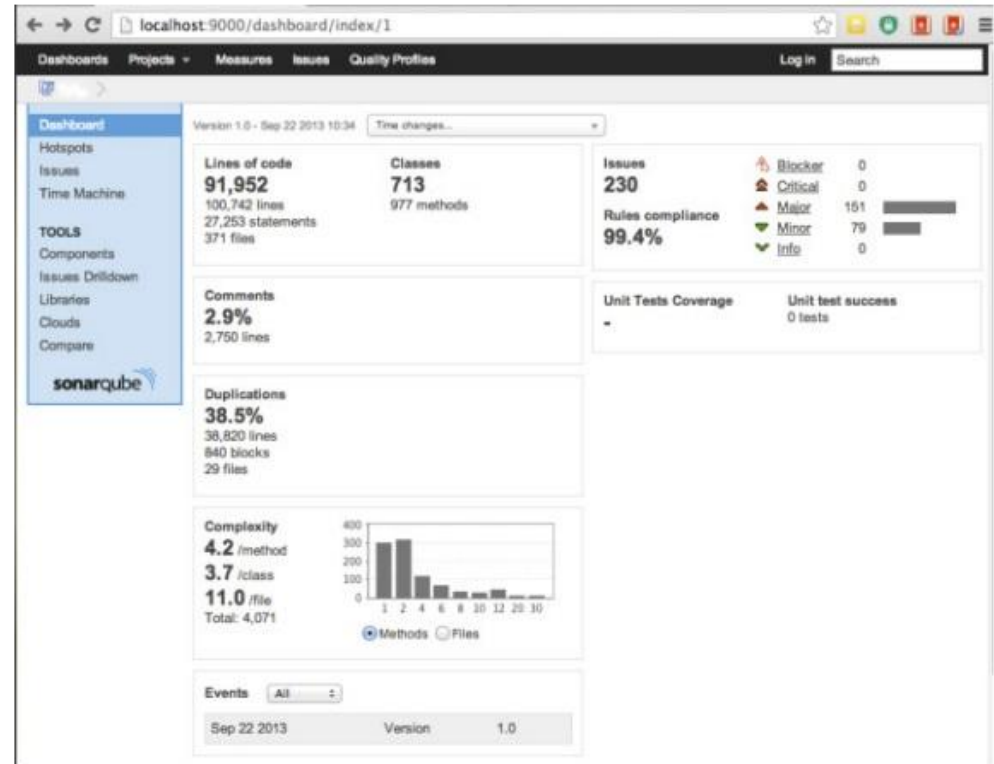


Figura 4 - Painel com resultado da análise de código

- **Análise dinâmica**
 - Execução monitorada do programa
 - Pode exigir instrumentação de código
 - Abordagens
 - **Análise de cobertura**
 - Teste de fluxo de controle
 - Teste de fluxo de dados
 - **Análise de mutantes**

- Análise dinâmica
 - Análise de cobertura
 - Geração de casos de teste
 - Grafo de programa

- **Análise dinâmica**
 - Análise de cobertura - dado um conjunto de execuções do programa, quanto do código fonte foi exercitado (coberto)?

- **Análise dinâmica**
 - Geração de casos de teste - cria-se os casos de teste com o objetivo de atingir um determinado valor de cobertura
 - Métricas
 - Fluxo de controle (Grafo de Fluxo de Controle – GFC)
 - Fluxo de dados

- **Análise dinâmica**
 - Grafo de programa - define uma relação entre o caso de teste e a parte do programa exercitada por ele
 - Um caso de teste
 - corresponde a um caminho no grafo
 - corresponde a uma execução completa: do nodo inicial até o final

Uma técnica conhecida para descobrir o número de caminhos necessários para cobrir todos os arcos e nodos em um GFC. Passos:

1. Desenhe o GFC para o fragmento de código.
2. Compute a complexidade ciclomática C , para o GFC.
3. Descubra no máximo C caminhos que cobrem os nodos e arcos em um GFC, denotado Conjunto de Caminhos Básicos;
4. Construa os casos de teste para forçar a execução entre os caminhos do Conjunto de Caminhos Básicos.

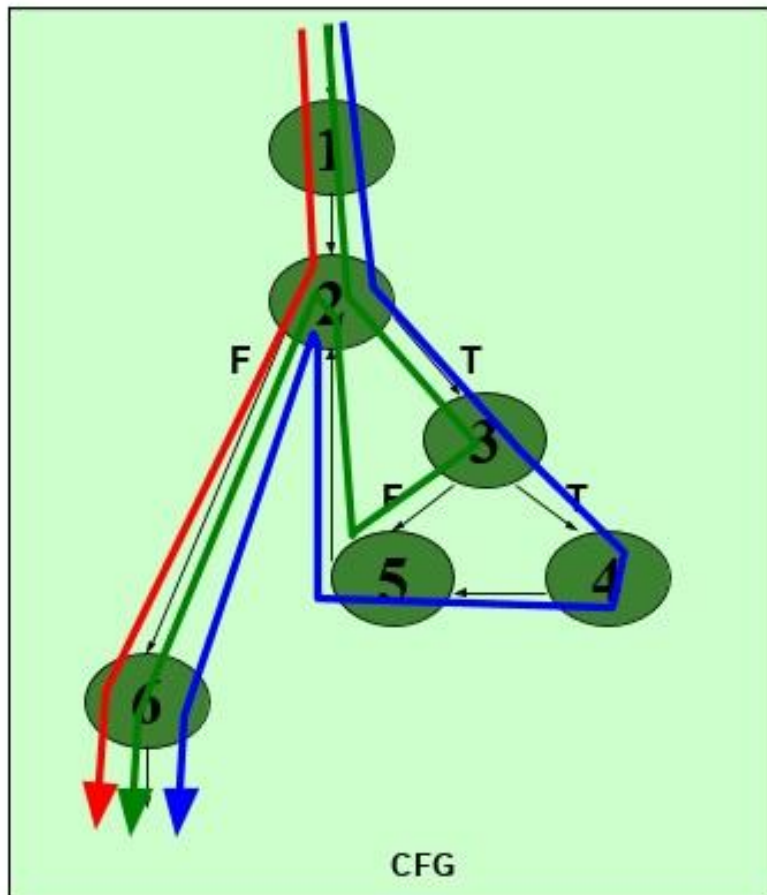
Teste de Caixa Branca: Caminho independente

Um caminho realizável no grafo do nodo inicial ao nodo final que ainda não foi visitado. Deve se mover em ao menos um arco que ainda não tenha sido visitado.

O objetivo é cobrir todos os comandos de um programa através de caminhos independentes.

O número de caminhos independentes a serem descobertos \leq número da complexidade ciclomática.

Teste de Caixa Branca: Exemplo (Passo 3)



- Complexidade Ciclomática = 3.
- São necessários no máximo 3 caminhos independentes para cobrir o GFC.
- Neste exemplo:
 - [1 - 2 - 6]
 - [1 - 2 - 3 - 5 - 2 - 6]
 - [1 - 2 - 3 - 4 - 5 - 2 - 6]

Caminho: [**1 – 2 – 6**]

Caso de Teste: $A = \{ 5, \dots \}$, $N = 1$

Saída esperada: 5

Caminho : [**1 – 2 – 3 – 5 – 2 – 6**]

Caso de Teste : $A = \{ 5, 9, \dots \}$, $N = 2$

Saída esperada : 5

Caminho : [**1 – 2 – 3 – 4 – 5 – 2 – 6**]

Caso de Teste : $A = \{ 5, 4, \dots \}$, $N = 2$

Saída esperada : 4

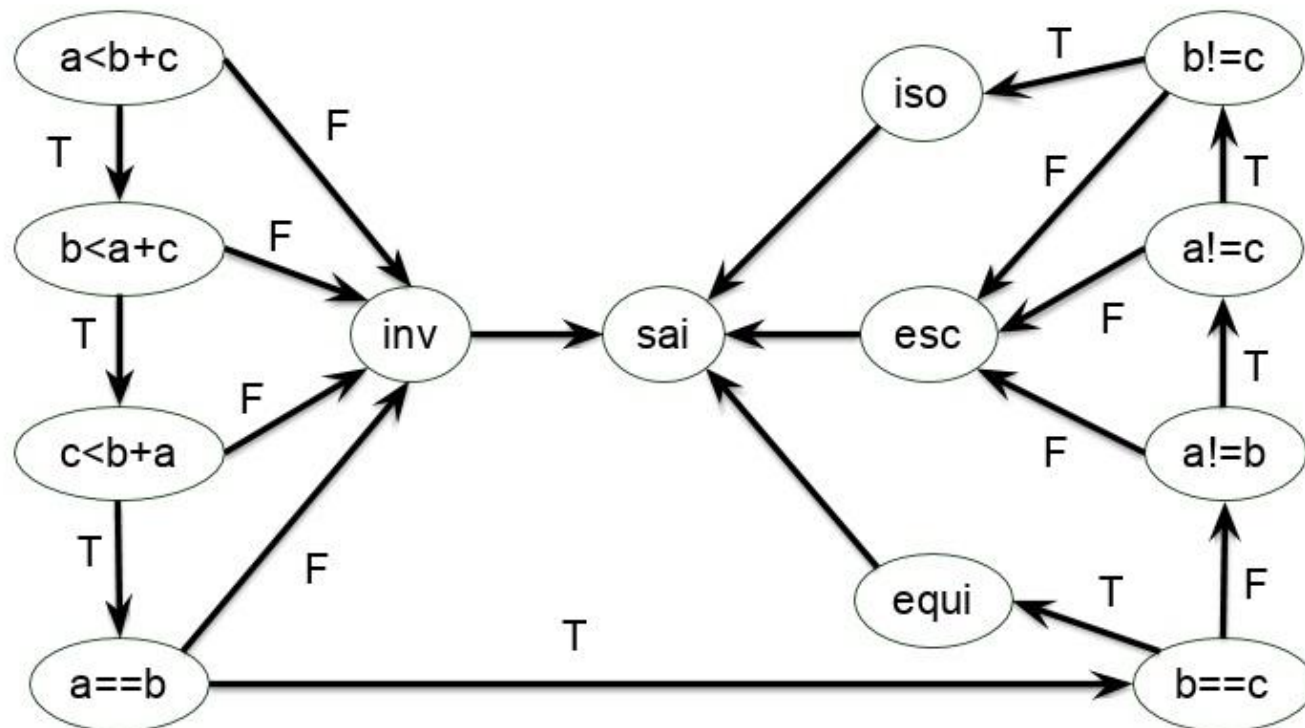
Estes testes resultam em uma cobertura completa de decisão e comandos do código

Exercícios

Exercício I - Triângulo

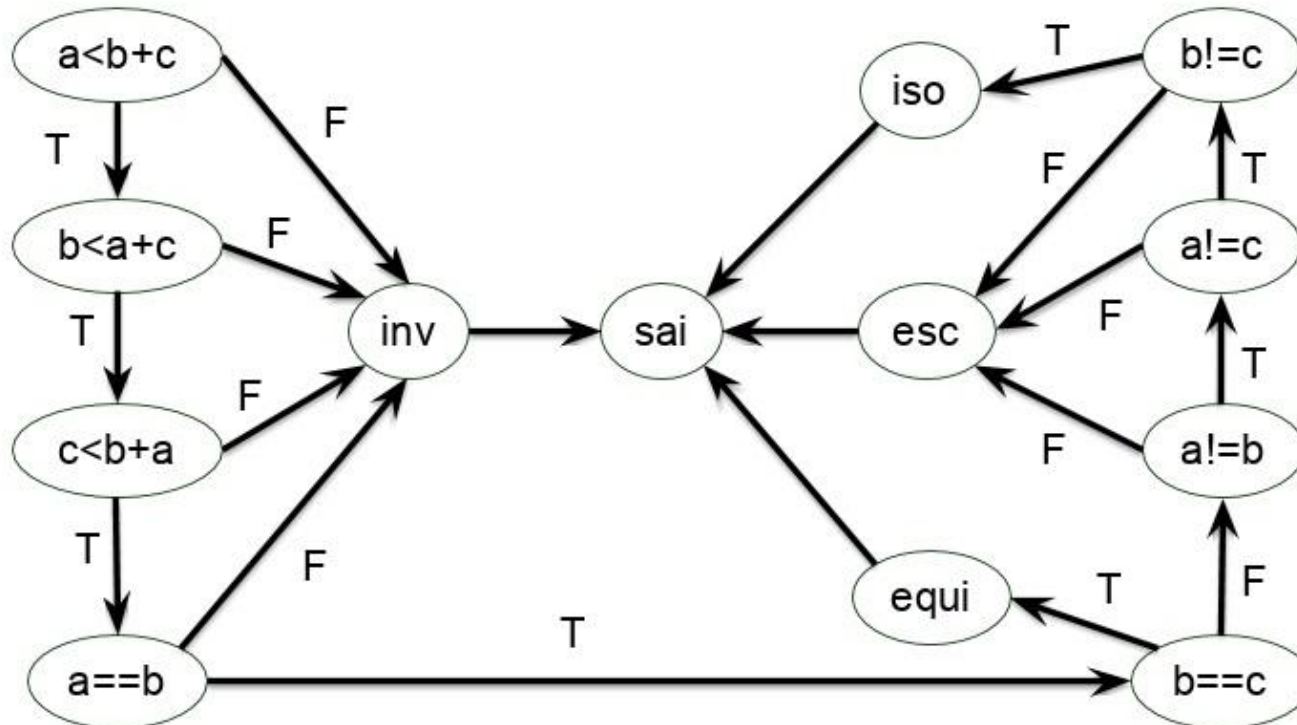
```
public int identificaTriangulo2(int a, int b, int c) {  
    if ((a < b + c) && (b < a + c) && (c < b + a)) {  
        if ((a == b) && (b == c))  
            return (int)tipos.EQUILATERO;  
        else  
            if ((a != b) && (a != c) && (b != c))  
                return (int)tipos.ESCALENO;  
            else return (int)tipos.ISOSCELES;  
        }  
    return (int) tipos.INVALIDO;  
}
```

Exercício I - Triângulo



E a complexidade ciclomática?

Exercício I - Triângulo



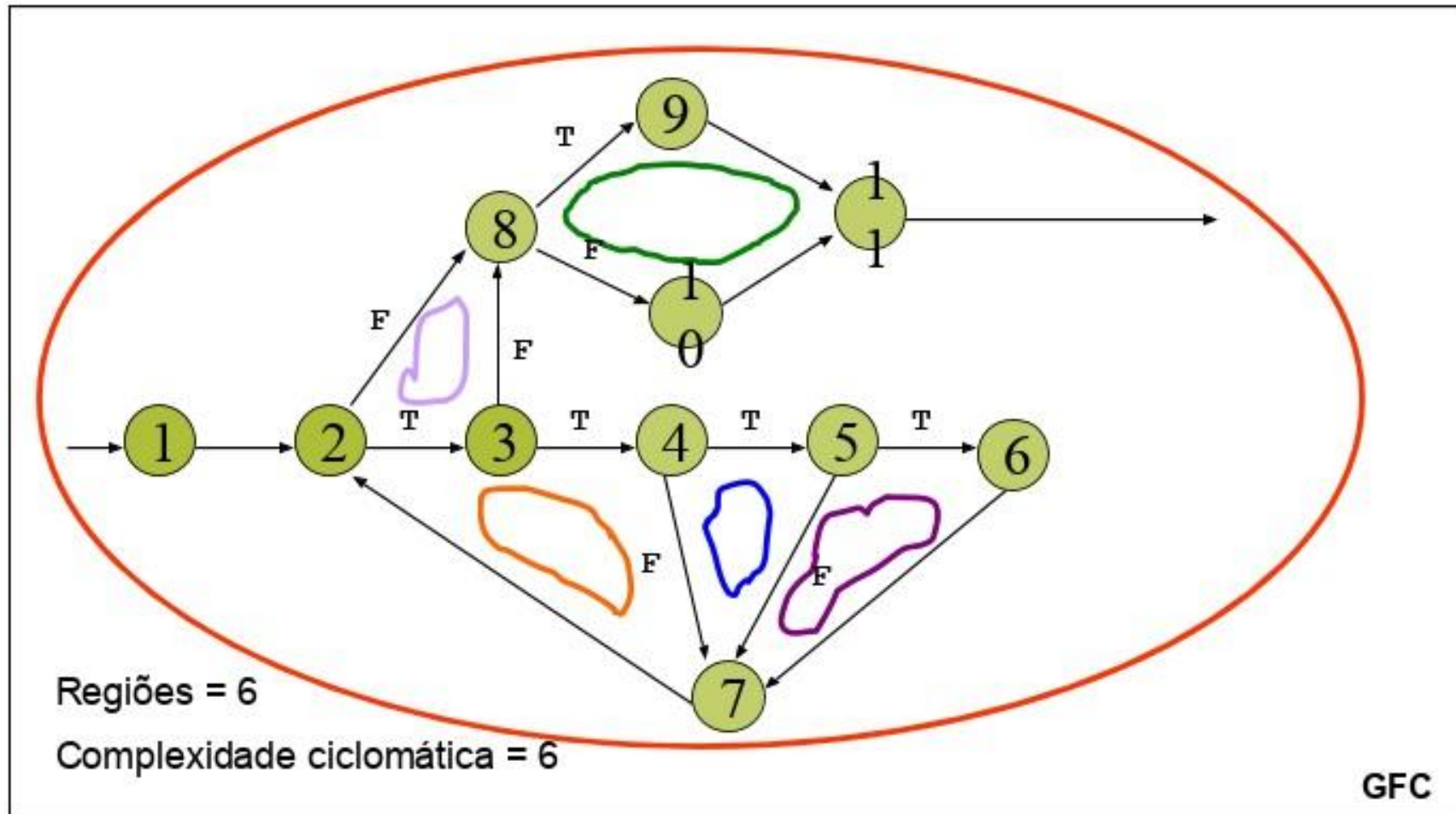
Predicados: 8
CC=8+1=9

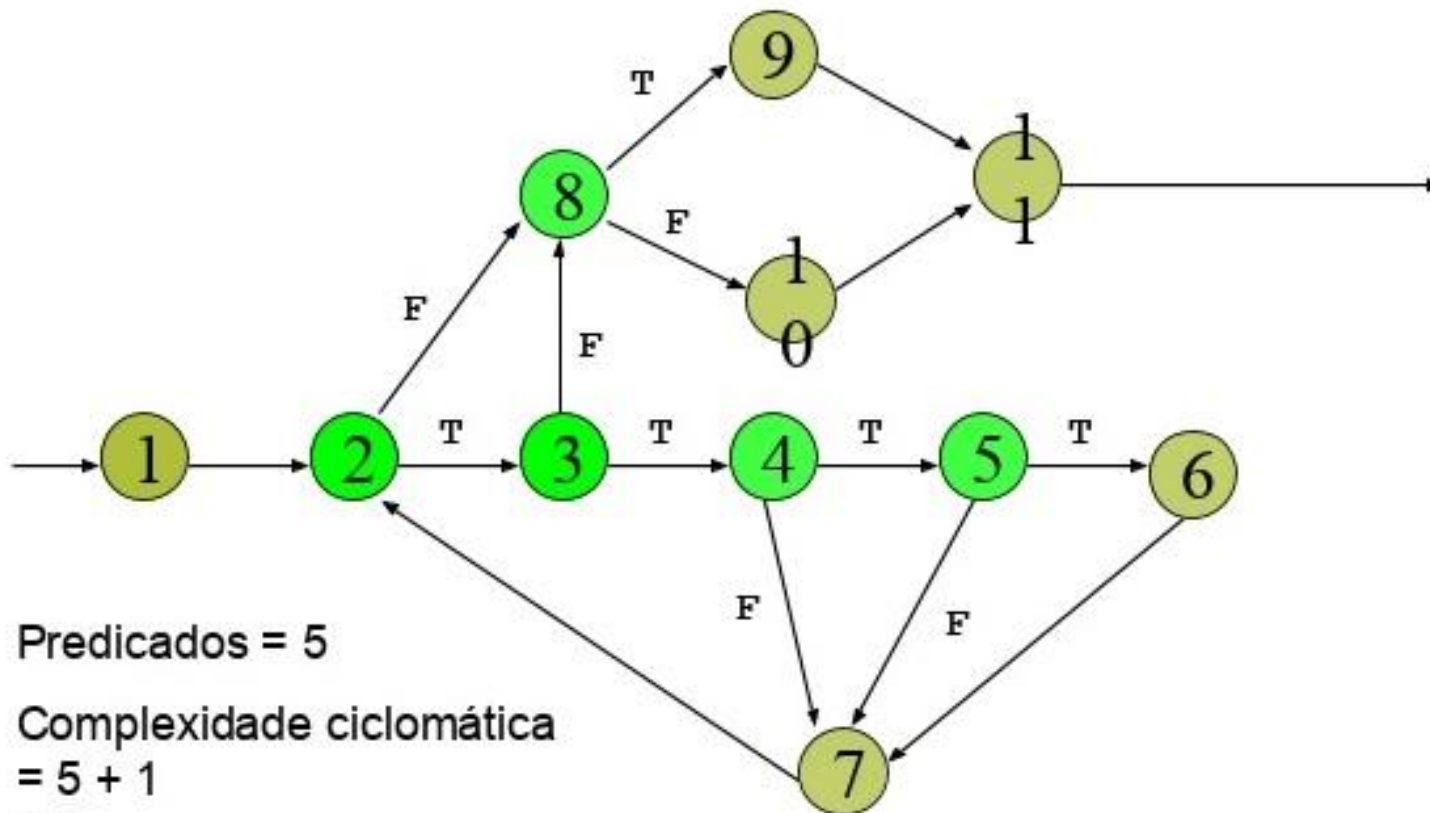
E a complexidade ciclomática?

Caminhos que podem ser testados?

```
int average (int[ ] value, int min, int max, int N) {  
    int i, totalValid, sum, mean;  
    i = totalValid = sum = 0;  
    while ( i < N && value[i] != -999 ) {  
        if (value[i] >= min && value[i] <= max) {  
            totalValid += 1; sum += value[i];  
        }  
        i += 1;  
    }  
    if (totalValid > 0)  
        mean = sum / totalValid;  
    else  
        mean = -999;  
    return mean;  
}
```

Quais são os estados?
Qual é o GFC? E a CC?





Predicados = 5

Complexidade ciclomática

= 5 + 1

= 6

Tem 6 caminhos independentes. Normalmente, caminho mais simples = mais fácil de construir um caso de teste.

Entretanto, alguns caminhos simples não são possíveis de serem feitos (não realizáveis). Exemplo: [1 – 2 – 8 – 9 – 11], impossível de ser executado, verifique olhando para o código correspondente.

Conjunto de caminhos básicos:

[1 – 2 – 8 – 10 – 11].

[1 – 2 – 3 – 8 – 10 – 11].

[1 – 2 – 3 – 4 – 7 – 2 – 8 – 10 – 11].

[1 – 2 – 3 – 4 – 5 – 7 – 2 – 8 – 10 – 11].

[1 – (2 – 3 – 4 – 5 – 6 – 7) – 2 – 8 – 9 – 11].

No último caso, (...) representa possível repetição.