Testes e Qualidade em Jogos

5º período

Professora: Michelle Hanne

PUC Minas

Complexidade Ciclomática

A **complexidade ciclomática** é uma métrica cujo objetivo é medir a complexidade de determinado **código fonte**. Quanto maior o **seu valor, maior a dificuldade de se entender, modificar e, consequentemente, testar o código fonte** (McCabe, 1976)

PUC Minas

Complexidade Ciclomática

Consiste na contagem do número de caminhos independentes que ele pode executar até o seu fim. Um caminho independente é aquele que apresenta pelo menos uma nova condição (possibilidade de desvio de fluxo) ou um novo conjunto de comandos a serem executados.

PUC Minas

Complexidade Ciclomática

"Caminho Linearmente Independente é qualquer caminho do programa que introduz pelo menos um novo conjunto de comandos de processamento ou uma nova condição. Quando definido em termos de grafo de fluxo, um caminho independente deve incluir pelo menos uma aresta que não tenha sido atravessada antes de o caminho ser definido" (Pressman, 2006).

PUC Minas

Cálculo da Complexidade Ciclomática

- **Existem diferentes formas de se calcular:**
  - Usando a notação de um grafo de fluxo;
  - Usando fluxograma;
  - Com análise estática do código, usando uma ferramenta que automatize essa tarefa.

Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

PUC Minas

Cálculo da Complexidade Ciclomática

Tendo um grafo de fluxo ou um fluxograma, temos três fórmulas equivalentes para se mensurar a complexidade ciclomática:

- **V(G) = R**, onde R é o número de regiões do grafo de fluxo.
- **V(G) = E – N + 2,** onde E é o número de arestas (setas) e N é o número de nós do grafo G.
- **V(G) = P + 1,** onde P é o número de nós-predicados contidos no grafo G (só funciona se os nós-predicado tiverem no máximo duas arestas saindo.) *Nós-predicado são àqueles que podem desviar o fluxo da execução: if, while, switch etc.*

Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

PUC Minas

Cálculo da Complexidade Ciclomática

- Exemplo usando a notação

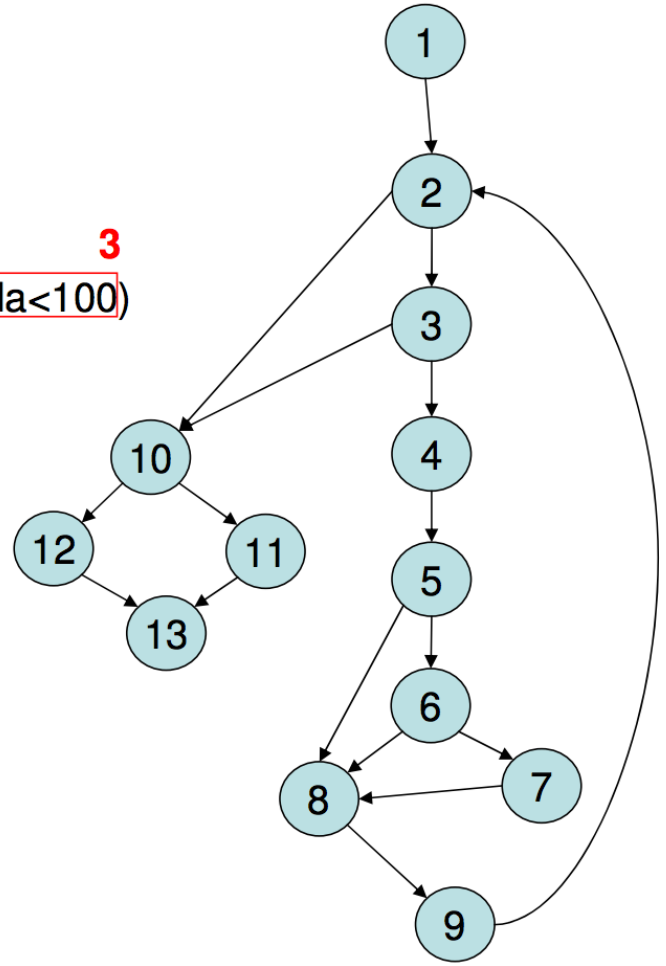
de grafo de fluxo:

V(G) = E – N + 2

onde **E** é o número de arestas (setas) e **N** é o número de nós do grafo G.

V(G) = 17 arestas/setas – 13 nós + 2 = 6

```
Procedimento média(valor[])
i=1;                                                    1
soma=0;
total.entrada=0;
total.válidas=0                                  2                  3
Faça-Enquanto (valor[i]≠-999 E total.entrada<100)
    incremente total.entrada de 1;                      4
    Se (valor[i]≥min E valor[i]≤max)                5       6
    Então
7           incremente total.válidas de 1;
            soma = soma + valor[i]
    Fim-Se
8   incremente i de 1;
Fim-Enquanto                                        9
Se total.válidas>0                                  10
Então média = soma/total.válidas;                   11
Senão média = -999;                                 12
Fim-Se                                              13
Fim média
```



Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

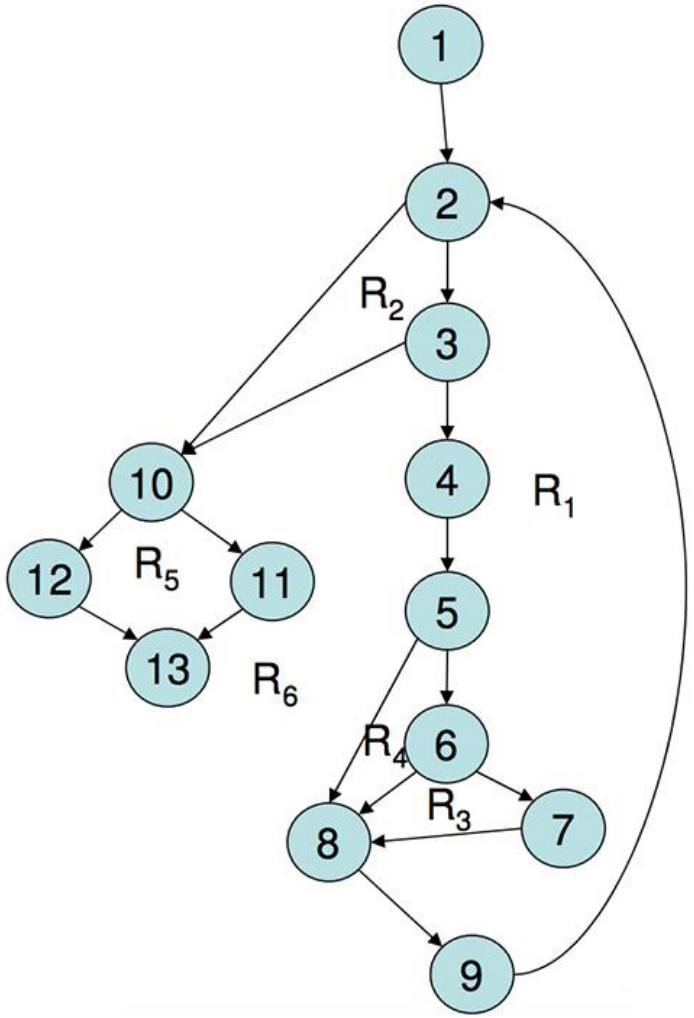
Cálculo da Complexidade Ciclomática

V(G) = R, onde R é o número de regiões do grafo de fluxo.

V(G) = P + 1, onde **P** é o número de nós-predicados contidos no grafo G.

V(G) = 5 nós-predicados + 1 = 6

No final temos, então, 6 caminhos independentes. Com isso, sabemos que precisamos ter uma gama de pelo menos 6 testes para garantir uma boa cobertura para esse código.



Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

PUC Minas

Cálculo da Complexidade Ciclomática

Os seis caminhos independentes são:

1) 1-2-10-12-13

2) 1-2-10-11-13

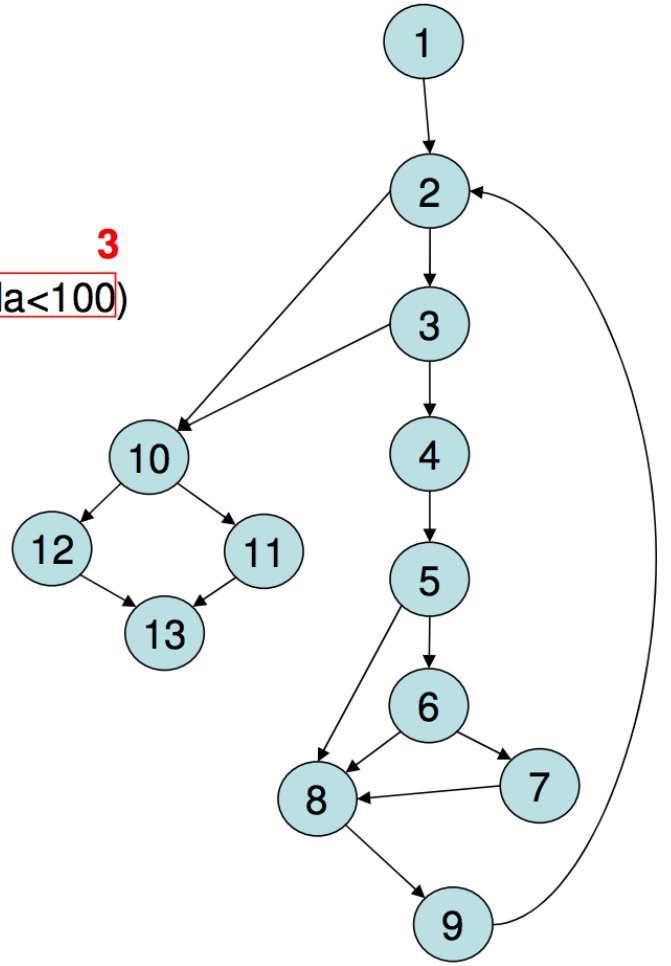
3) 1-2-3-10-11-13

4) 1-2-3-4-5-8-9-2-[...]

5) 1-2-3-4-5-6-8-9-2-[...]

6) 1-2-3-4-5-6-7-8-9-2-[...]

```
Procedimento média(valor[])
i=1;                                          1
soma=0;
total.entrada=0;
total.válidas=0
                                              2                    3
Faça-Enquanto (valor[i]≠-999 E total.entrada<100)
    incremente total.entrada de 1;            4
    Se (valor[i]≥min E valor[i]≤max)          5                    6
    Então
7           incremente total.válidas de 1;
            soma = soma + valor[i]
    Fim-Se
8   incremente i de 1;
Fim-Enquanto                                  9
Se total.válidas>0                            10
Então média = soma/total.válidas;             11
Senão média = -999;                           12
Fim-Se                                        13
Fim média
```



Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

PUC Minas

Quais os parâmetros aceitáveis para a complexidade dos meus métodos?

O fato de um método ter baixa complexidade não quer dizer que ele não pode ser melhorado ou até mesmo refatorado. Essa é a parte relativa da "coisa". Caberá a você e a sua equipe identificar esses pontos.

Complexidade	Avaliação
1-10	Método simples. **Baixo risco**.
11-20	Método razoavelmente complexo. **Moderado risco**.
21-50	Método muito complexo. **Elevado risco**.
51-N	Método de **altíssimo risco** e bastante instável.

Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

PUC Minas

Cálculo da Complexidade Ciclomática

Valores **mais altos de complexidade ciclomática resultam na necessidade de um maior número de casos de teste para testar de forma abrangente um bloco de código.**

- Ao reduzir a complexidade ciclomática - e, idealmente, também a rotatividade do código - você estará mitigando esses riscos.

  1- Preferir funções menores

  2- Evitar Argumentos de Bandeira em Funções - são parâmetros booleanos que você acrescenta a uma função.

  3- Reduzir o número de estruturas de decisão - Use Design Patterns

  4 - Livre-se do código duplicado)

  5- Remover Código Obsoleto

  6- Não Reinvente a Roda

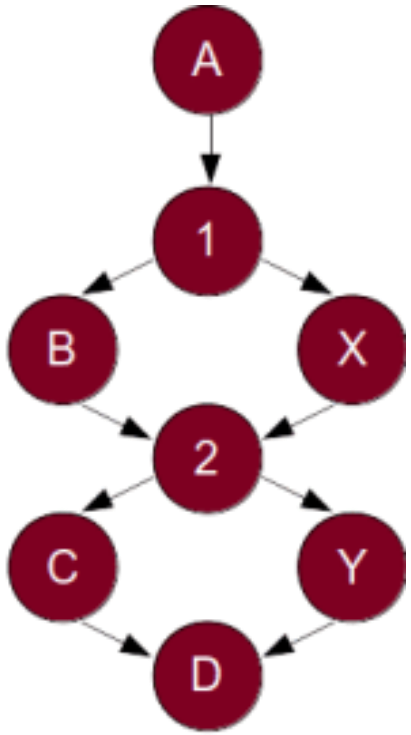
Fonte: https://carlosschults.net/pt/reduzir-complexidade-ciclomatica/

PUC Minas

Exemplo Complexidade Ciclomática

- O exemplo mostra graficamente um algoritmo onde as **letras representam uma instrução simples e os números representam um "IF" (decisão)**. Repare que de uma instrução simples parte apenas uma seta (aresta) enquanto de uma instrução de decisão partem duas.

```
print("A")
if (condition1)
  print("X")
else
  print ("B")
if (condition2)
  print("Y")
else
  print ("C")
print("D")
```

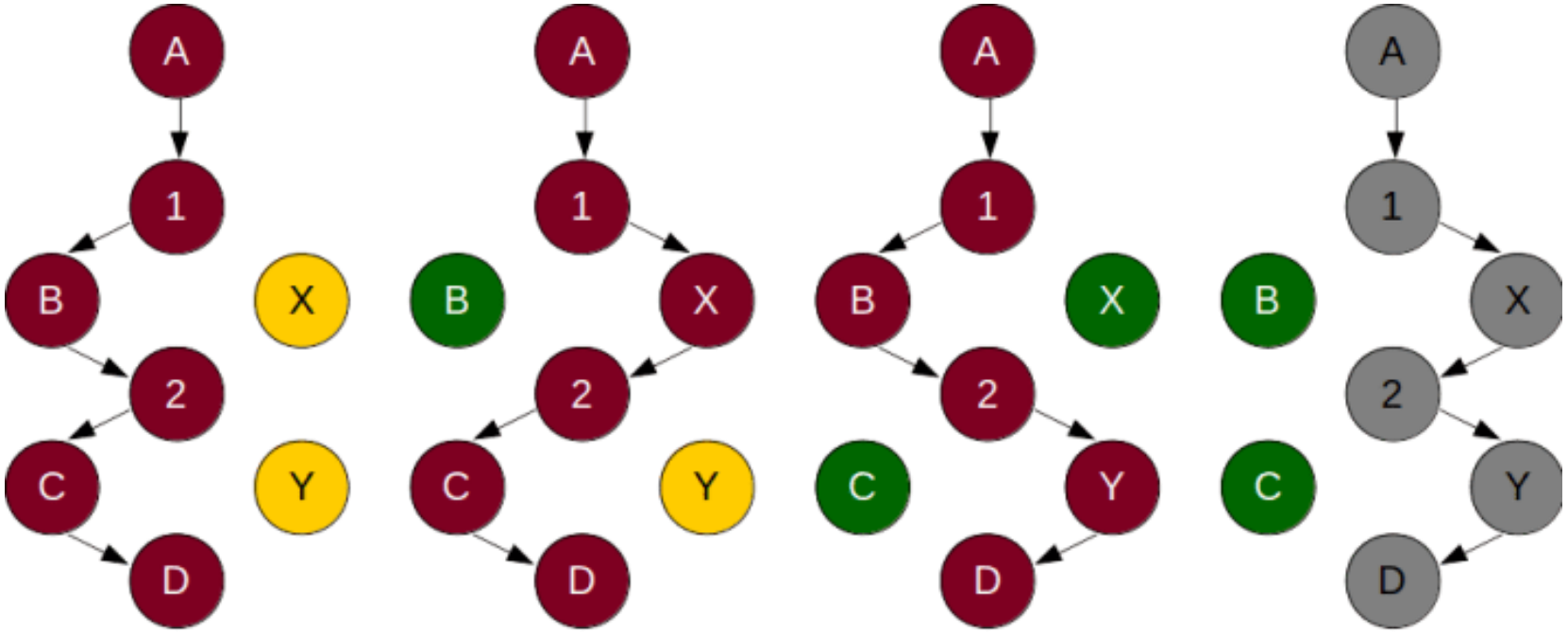


Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

PUC Minas

Exemplo Complexidade Ciclomática

- O mesmo algoritmo é desmembrado em seus caminhos possíveis (NPATH) que são 4, porém somente os três primeiros são linearmente independentes. O último caminho não passa em nenhuma instrução nova pela qual já não tenha sido passada antes.



Independentes: A-1-B-2-C-D A-1-X-2-C-D A-1-B-2-Y-D

Não Independente: A-1-X-2-Y-D

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

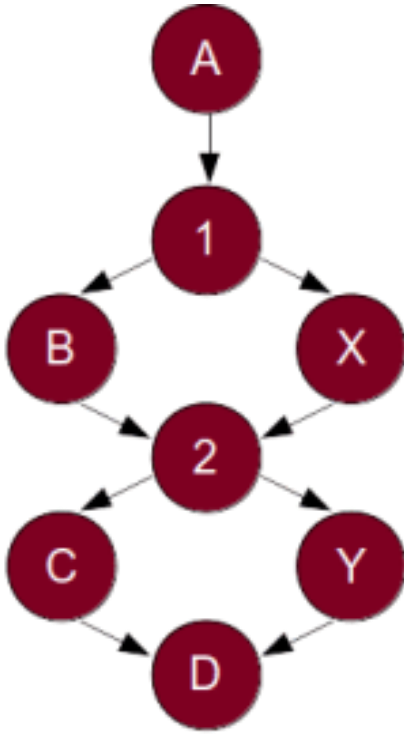
Exemplo Complexidade Ciclomática

- No exemplo acima, apesar de haver 4 caminhos possíveis (NPATH), podemos dizer que a complexidade **ciclomática do algoritmo representado graficamente é 3**, ou seja, é igual ao número de caminhos linearmente independentes.

V(G) = E – N + 2

V(G) = 9 arestas – 8 nós + 2 = 3

```
print("A")
if (condition1)
  print("X")
else
  print ("B")
if (condition2)
  print("Y")
else
  print ("C")
print("D")
```



Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

PUC Minas

Exemplo da Sorveteria - Requisitos

- Imagine uma sorveteria que define o preço de seus sorvetes da seguinte forma:

1. Há dois tipos de sorvete: Comum, cujo preço é R$15 e Premium cujo preço é R$20.

2. O sorvete pode ser vendido em copinho ou casquinha. O copinho adiciona R$1 ao preço enquanto a casquinha adiciona R$2.

3. A cobertura pode ser simples (apenas uma) custando R$1 ou especial (duas coberturas ou mais) custando R$2.

O cálculo do preço do sorvete pode ser representado pelo seguinte código escrito em java (implementação 1):

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria - Requisitos

PUC Minas

```
public int precoSorvete(boolean premium, boolean casquinha, int
coberturas) {
  int preco = 0;
  if (premium) {
    preco = 20;
  } else {
    preco = 15;
  }
  if (casquinha) {
    preco = preco + 2;
  } else {
    preco = preco + 1;
  }
  if (coberturas > 1){
    preco = preco + 2;
  } else {
    preco = preco + 1;
  }
  return preco;
 }
```

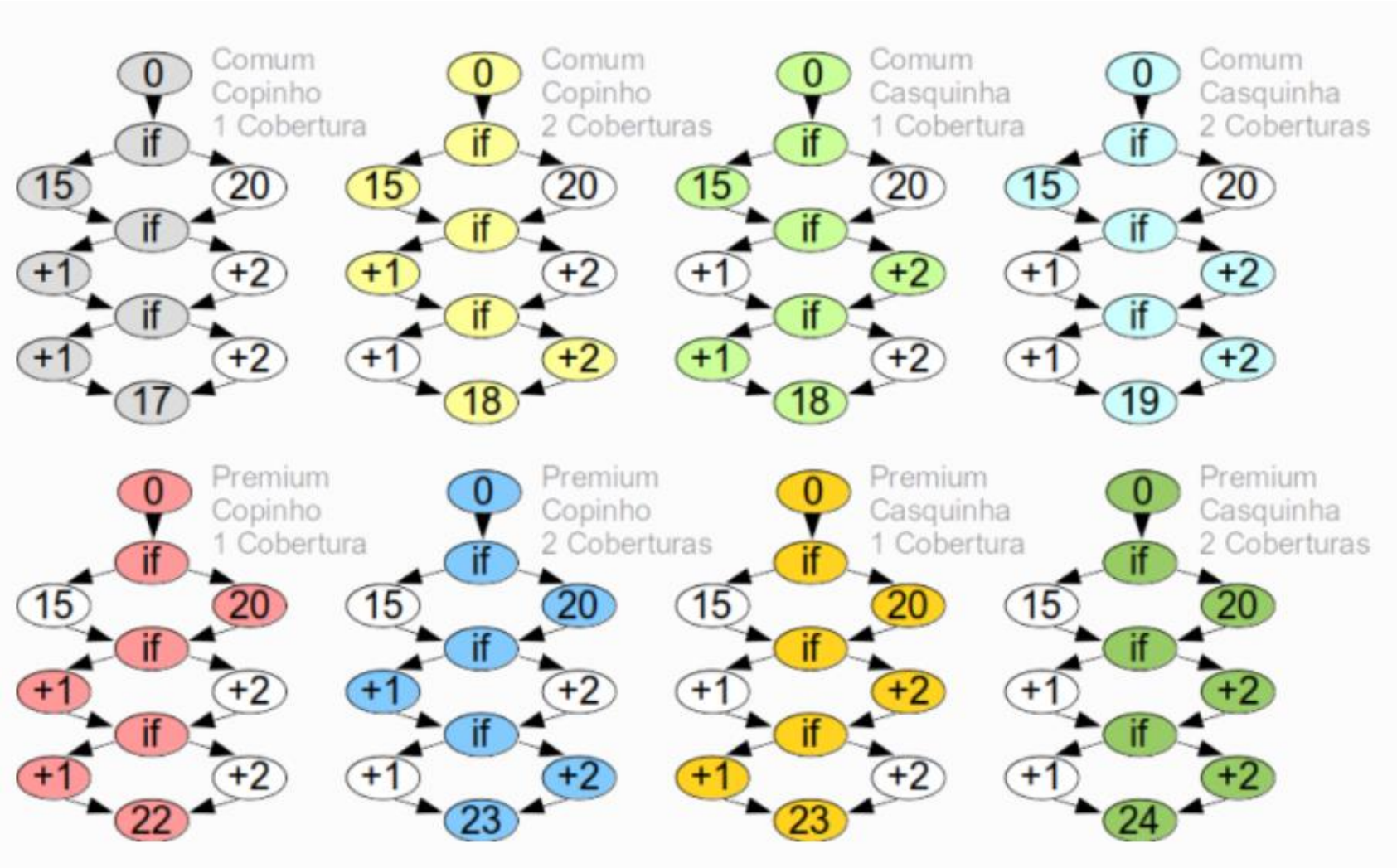
O método espera três parâmetros baseados na definição do preço do sorvete onde o cliente deve informar três coisas: tipo do sorvete (se é premium ou não), recipiente (se quer na casquinha ou não) e cobertura (uma ou mais). A partir destas três informações o algoritmo calcula o preço do sorvete.

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Combinações

PUC Minas

Temos oito combinações possíveis destes parâmetros gerando oito opções de compra do sorvete



As oito combinações correspondem aos oito caminhos possíveis do algoritmo de cálculo de preço. Uma forma de obter uma cobertura de testes de 100% seria fazer um caso de teste para cada caminho

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

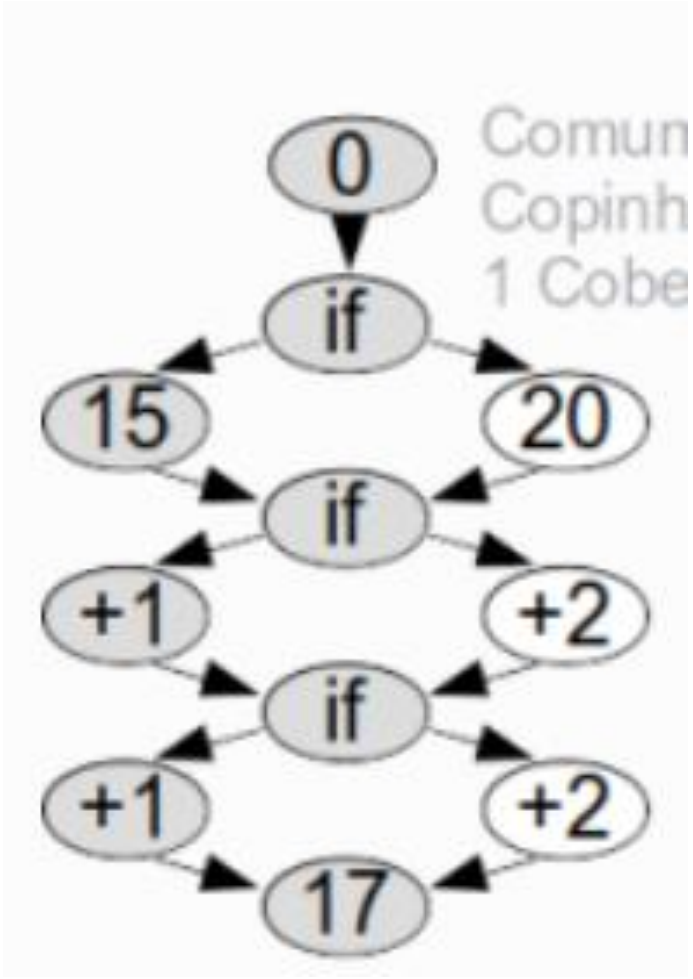
Exemplo da Sorveteria – Complexidade Ciclomática

O crescimento do número de casos de teste será exponencial e em breve será impossível manter uma cobertura alta para este algoritmo. Por isso, a complexidade ciclomática é um indicador bem melhor para sugerir o número de casos de teste necessários em um teste de unidade.

V(G) = E – N + 2

V(G) = 13 arestas – 11 nós + 2 = 4

Complexidade Ciclomática = 4



Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

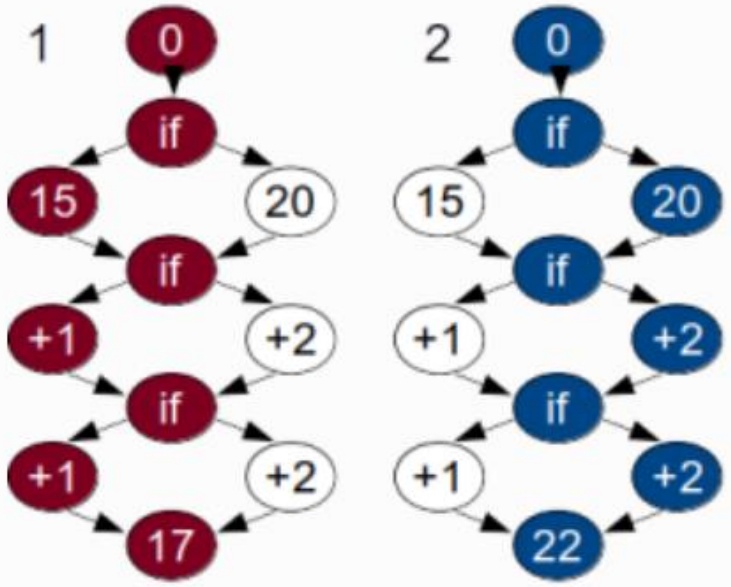
Exemplo da Sorveteria – Casos de Testes

PUC Minas

Percebe-se que apenas dois casos de teste são suficientes para cobrir 100% das linhas de código. Isso se deve a particularidade de como o requisito foi implementado. O algoritmo em questão tem um caminho principal e um caminho alternativo. **Dessa forma apenas dois cenários acabaram passando por todas as suas linhas.**

```
assertEquals(17,
  sorveteria.precoSorvete
  (false, false, 1));

assertEquals(24,
  sorveteria.precoSorvete
  (true, true, 2));
```



ops > Sorveteria.java (100.0%)

ops

src/main/java/siviotti/ops/example/cc/Sorveteria.java

```
1   package siviotti.ops.example.cc;
2
3   public class Sorveteria {
4
5           public int precoSorvete(boolean premium, boolean casquinha, int coberturas) {
6                   int preco = 0;
7                   if (premium) {
8                           preco = 20;
9                   } else {
10                          preco = 15;
11                  }
12                  if (casquinha) {
13                          preco = preco + 2;
14                  } else {
15                          preco = preco + 1;
16                  }
17                  if (coberturas > 1){
18                          preco = preco + 2;
19                  } else {
20                          preco = preco + 1;
21                  }
22                  return preco;
23          }
24
25
26  }
```

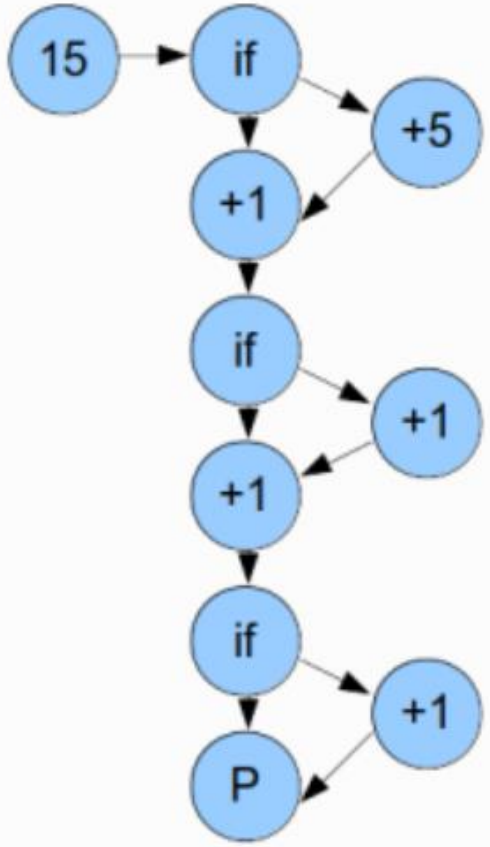
Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 2

PUC Minas

A implementação do requisito de cálculo de preço do sorvete com If/Else gerou a **necessidade de dois casos de teste**. Foi elaborada uma implementação ainda mais simples onde o caminho principal fica mais explícito. Repare que o grafo também se altera.

```
public int precoSorvete(boolean premium,
        boolean casquinha, int coberturas) {
    int preco = 15;
    if (premium) {
        preco = preco + 5;
    }
    preco = preco + 1; // pote
    if (casquinha) {
        preco = preco + 1;
    }
    preco = preco + 1; // cobertura
    if (coberturas > 1){
        preco = preco + 1;
    }
    return preco;
}
```

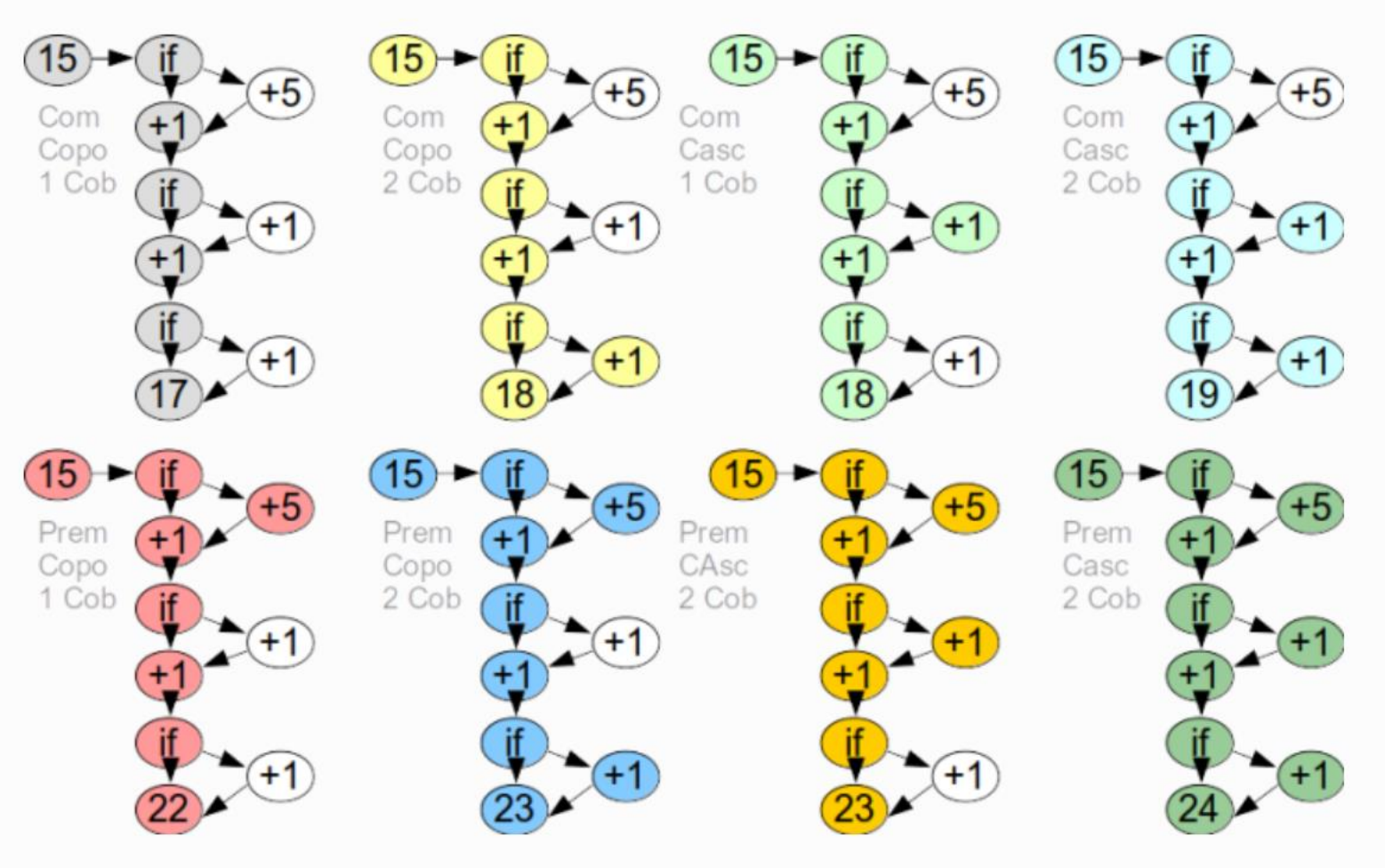


Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 2

PUC Minas

É possível ver que nesta implementação continuam existindo as oito possibilidades de sorvete e que a complexidade ciclomática continua 4.



V(G) = E – N + 2

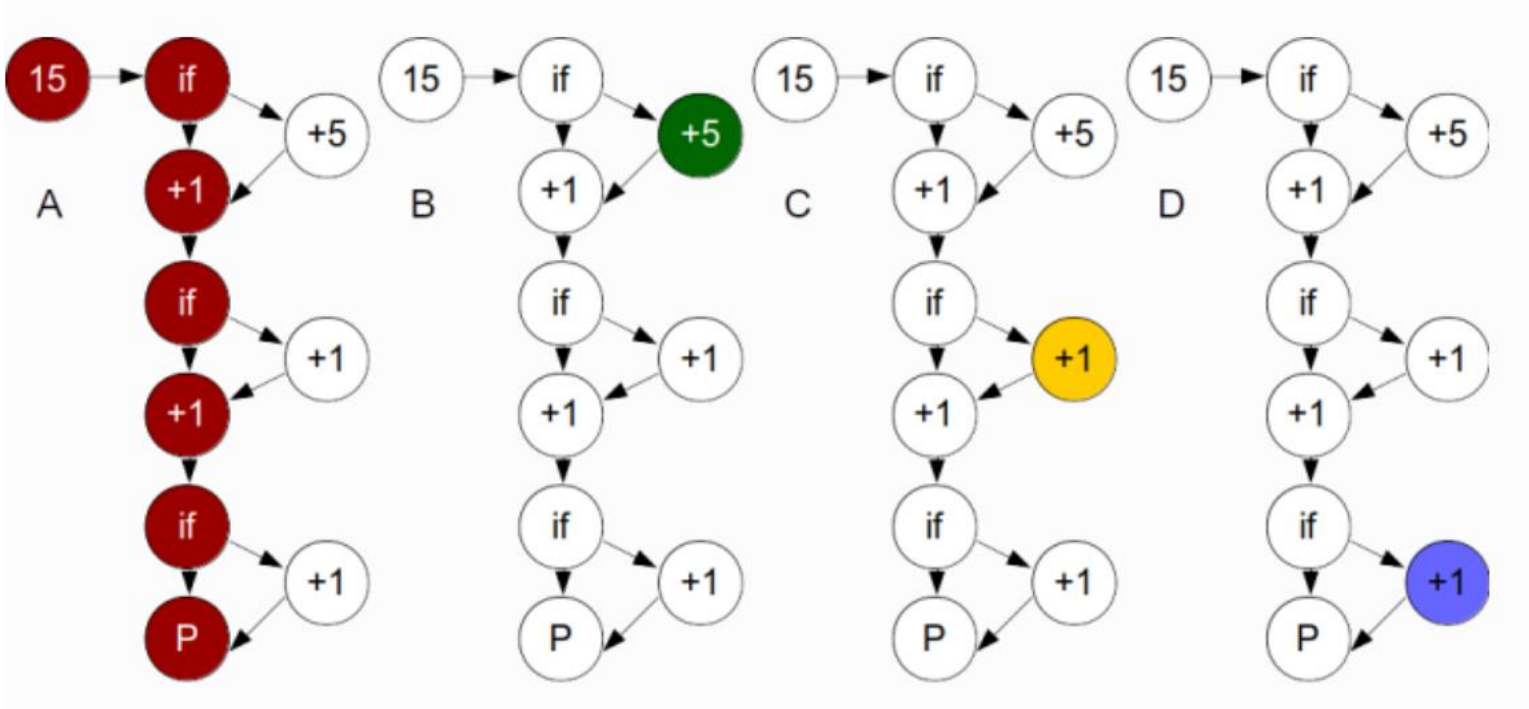
V(G) = 12 arestas – 10 nós + 2 = 4

Complexidade Ciclomática = 4

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 2

Temos 4 caminhos linearmente independentes:



V(G) = E – N + 2

V(G) = 12 arestas – 10 nós + 2 = 4

Complexidade Ciclomática = 4

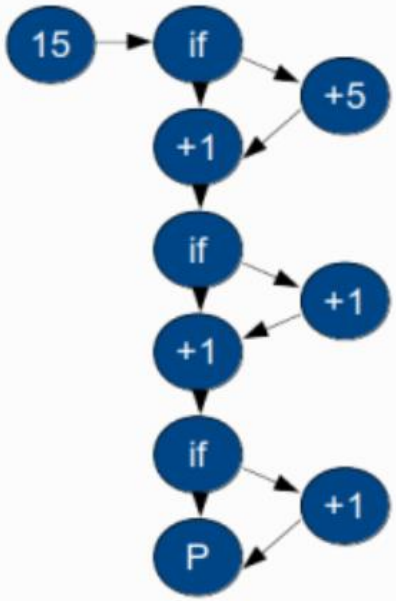
Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 2

PUC Minas

Cenário de Teste – apenas 1 cenário cobre 100% dos testes

```
ssertEquals(24,
  sorveteria.precoSorvete
  (true, true, 2));
```



ops > Sorveteria2.java (100.0%)

ops

src/main/java/siviotti/ops/example/cc/Sorveteria2.java

```
1   package siviotti.ops.example.cc;
2
3   public class Sorveteria2 {
4
5           public int precoSorvete(boolean premium, boolean casquinha, int coberturas) {
6                   int preco = 15;
7                   if (premium) {
8                           preco = preco + 5;
9                   }
10                  preco = preco + 1; // pote
11                  if (casquinha) {
12                          preco = preco + 1;
13                  }
14                  preco = preco + 1; // cobertura
15                  if (coberturas > 1){
16                          preco = preco + 1;
17                  }
18                  return preco;
19          }
20
21  }
22
```

Esta nova implementação gerou uma situação onde a quantidade de cenários de teste para cobertura de todas as linhas baixou para um. **Com um único cenário de teste (o sorvete mais caro) todas as linhas são percorridas.** Isso não significa que esta implementação só precisa de um cenário em seu teste de unidade, apenas que com apenas um (o certo) todas as linhas serão cobertas.

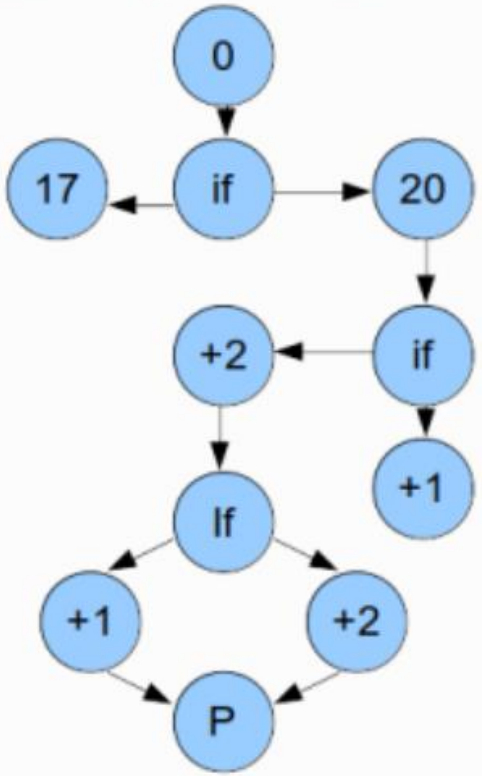
Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 3

PUC Minas

Em uma terceira implementação onde o requisito mudou um pouco os “IFs” estarão aninhados, tornando o código um pouco mais difícil de ler. Neste cenário somente os sorvetes premium têm casquinha e somente as casquinhas têm mais de uma cobertura. O comportamento e os caminhos possíveis (agora 4) são diferentes das duas anteriores, porém a **complexidade ciclomática ainda é 4.**

```
public int precoSorvete(boolean premium, boolean casquinha, int
coberturas) {
    int preco = 0;
    if (premium) { // só premiun tem casquinha
        preco = 20;
        if (casquinha) { // só casq tem cobe
            preco = preco + 2;
            if (coberturas > 1){
                preco = preco + 2;
            } else {
                preco = preco + 1;
            }
        } else {
            preco = preco + 1;
        }
    } else {
        preco = 15 + 1 + 1; // copo + 1 cob
    }
    return preco;
}
```



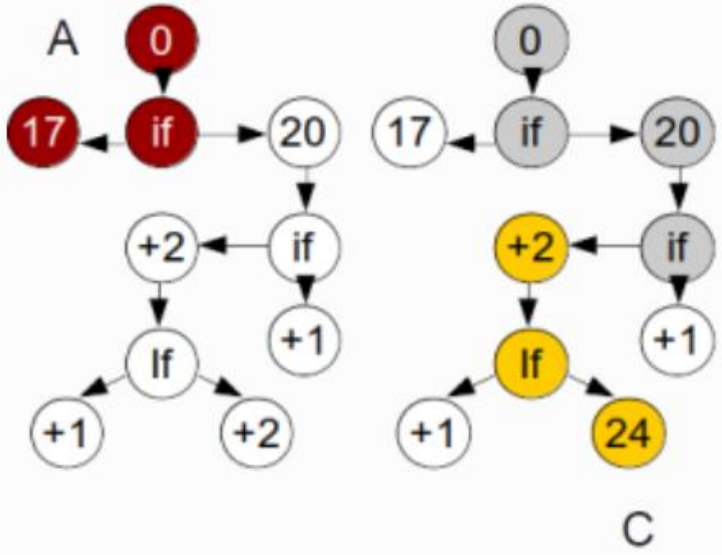
Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria – Implementação 3

PUC Minas

Pode-se ver que os mesmos dois cenários (mais barato 17 e mais caro 24) que cobriam 100% das linhas na implementação 1 agora cobrem somente 77,6% das linhas.

```
assertEquals(17,
  sorveteria.precoSorvete
  (false, false, 1));
assertEquals(24,
  sorveteria.precoSorvete
  (true, true, 2));
```



ops > Sorveteria3.java (77.8%)

ops
src/main/java/siviotti/ops/example/cc/Sorveteria3.java

```
1   package siviotti.ops.example.cc;
2
3   public class Sorveteria3 {
4
5       public int precoSorvete(boolean premium, boolean casquinha, int coberturas) {
6           int preco = 0;
7           if (premium) {
8               preco = 20;
9               if (casquinha) {
10                  preco = preco + 2;
11                  if (coberturas > 1){
12                      preco = preco + 2;
13                  } else {
14                      preco = preco + 1;
15                  }
16              } else {
17                  preco = preco + 1;
18              }
19          } else {
20              preco = 15 + 1 + 1; // Soverte + copo + 1 cobertura
21          }
22          return preco;
23      }
24
25
26  }
27
```

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

Exemplo da Sorveteria

PUC Minas

A complexidade ciclomática, cujo objetivo inicial era medir a complexidade de programas, apresenta forte relação com a testabilidade.

O valor de complexidade ciclomática é uma métrica mais razoável para elaboração de cenários de teste para um determinado código fonte**. Pode-se dizer que a complexidade ciclomática é o limite superior do número de cenários de teste necessários para cobrir 100% das linhas de código.**

Do ponto de vista prático e da qualidade de software, o valor de complexidade ciclomática pode ser usado como medida de qualidade de um código, assim como outras medidas, indicando sua complexidade

Fonte: https://artesoftware.com.br/2019/02/09/complexidade-ciclomatica/

PUC Minas

Cálculo da Complexidade Ciclomática

Com os caminhos definidos, você pode planejar os casos de teste para cada um deles.

Ferramentas para análise estática, garante maior produtividade. Dentre as várias vantagens, temos:

- Precisão;
- Possibilidade de incluir a análise na integração contínua;
- Geração de relatórios da evolução das métricas;
- Existem várias ferramentas, você pode pesquisar sobre “análise estática” na linguagem que você utiliza em seus projetos, certamente encontrará várias opções (com focos variados).
- https://www.sonarsource.com/products/sonarqube/
- https://www.jetbrains.com/resharper/

Fonte: https://www.treinaweb.com.br/blog/complexidade-ciclomatica-analise-estatica-e-refatoracao

Referências

McCabe, Thomas J. (1976). A Complexity Measure. Department of Defense, National Security Agency – EUA.

PRESSMAN, R, Engenharia de Software. 6a edição. São Paulo. McGraw-Hill, 2006.

SCHULTZ, Charles; BRYANT, Robert; LANGDELL, Tim. *Game Testing All in One*. Thomson Course Technology, 2005, capítulo 10.