PUC Minas

Lista 3 – Complexidade Ciclomática

INFORMAÇÕES DOCENTE						
CURSO: ENGENHARIA DE SOFTWARE	DISCIPLINA: FUNDAMENTOS DE PROJETO E ANÁLISE DE ALGORITMOS	TURNO	MANHÃ	TARDE	NOITE	PERÍODO/SALA: 5º
					x	
PROFESSOR (A): João Paulo Carneiro Aramuni						

Lista 3 - Gabarito

Complexidade Ciclomática - Recursividade

1) O algoritmo abaixo implementa uma função recursiva para o cálculo fatorial.

```python
def fatorial(n):
    if n == 0 or n == 1:
        return 1
    return n * fatorial(n - 1)
```

1. Monte o grafo de fluxo de controle da função:
- Identifique os nós (representando os pontos de decisão e instruções da função).
- Identifique as arestas (representando as transições entre os nós).

2. Calcule a complexidade ciclomática da função usando a fórmula:

$$M = E - N + 2P$$

- Onde: E é o número de arestas no grafo.
- N é o número de nós no grafo.
- P é o número de componentes conexos (neste caso, $P = 1$, pois a função é uma unidade única).

3. Interprete o valor da complexidade ciclomática:
- Explique o que significa o valor obtido para o número de caminhos independentes no código.

4. Descreva os caminhos independentes possíveis no grafo de fluxo de controle para essa função.

Cálculo: Função fatorial(n)

I. Representação da função em fluxo de controle
Passos do fluxo de controle:
1. Início da função.
2. Verificação da condição if n == 0 or n == 1.
- Se verdadeiro: Retorna 1.
- Se falso: Passa para o próximo passo.
3. Chamada recursiva n * fatorial(n - 1).
4. Retorno do valor calculado.

II. Estruturando o Grafo de fluxo
Um grafo de controle representa os caminhos possíveis da execução:
- Nó: Representa um ponto de decisão ou instrução.
- Aresta: Representa a transição entre nós.
- Componentes conexos (P): A função é uma unidade única, então $P = 1$.

Nós (N):
1. N1: Início da função.
2. N2: Verificação do if n == 0 or n == 1.
3. N3: Retorno 1 (caso base da recursão).
4. N4: Chamada recursiva fatorial(n - 1).
5. N5: Cálculo do retorno return n * fatorial(n - 1).
Número total de nós: $N = 5$.

Arestas (E):
1. N1 -> N2: Do início para a verificação do if: 1 aresta.
2. N2 -> N3: Caso n == 0 or n == 1, retorna 1: 1 aresta.
3. N2 -> N4: Caso contrário, vai para a chamada recursiva: 1 aresta.
4. N4 -> N5: Retorna o resultado da chamada recursiva: 1 aresta.
5. N4 -> N1: Chamada recursiva gerando nova execução: 1 aresta.
Número total de arestas: $E = 5$.

III. Aplicando a fórmula

Agora, usamos a fórmula da complexidade ciclomática:
$M = E - N + 2P$
Substituímos os valores:
$M = 5 - 5 + 2(1) \rightarrow M = 2$

IV. Interpretando o resultado

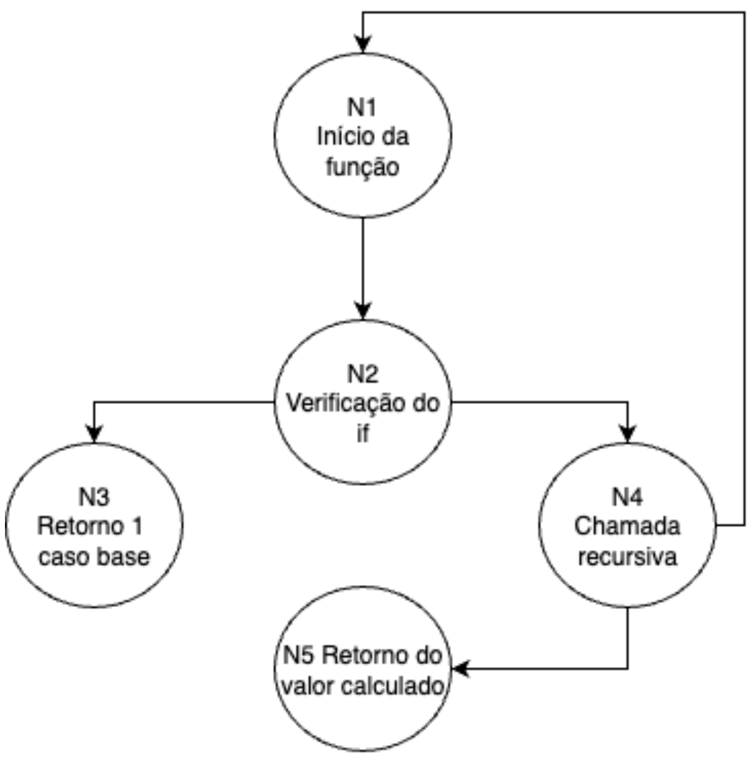
A complexidade ciclomática da função é 2.
Isso significa que há 2 caminhos independentes no grafo de fluxo de controle:

Caminho 1 - Caso base (n == 0 ou n == 1)
- A função recebe n = 0 ou n = 1.
- A condição if n == 0 or n == 1 é verdadeira.
- Retorna 1 imediatamente, sem chamadas recursivas.

Caminho 2 - Caso recursivo (n > 1)
- A função recebe um n > 1.
- A condição if n == 0 or n == 1 é falsa.
- A função chama n * fatorial(n - 1), gerando novas chamadas recursivas até chegar ao caso base.

Resumo dos caminhos independentes
Caminho 1: A função retorna 1 diretamente sem recursão (n == 0 ou n == 1).
Caminho 2: A função executa chamadas recursivas até alcançar o caso base (n > 1).

Desenhando o grafo de fluxo:



2) O algoritmo abaixo implementa uma função recursiva para o cálculo do número de Fibonacci.

```python
def fibonacci(n):
    if n == 0 or n == 1:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)
```

1. Monte o grafo de fluxo de controle da função:
- Identifique os nós (representando os pontos de decisão e instruções da função).
- Identifique as arestas (representando as transições entre os nós).

2. Calcule a complexidade ciclomática da função usando a fórmula:

$$M = E - N + 2P$$

- Onde: E é o número de arestas no grafo.
- N é o número de nós no grafo.
- P é o número de componentes conexos (neste caso, P = 1, pois a função é uma unidade única).

3. Interprete o valor da complexidade ciclomática:
- Explique o que significa o valor obtido para o número de caminhos independentes no código.

4. Descreva os caminhos independentes possíveis no grafo de fluxo de controle para essa função.

Cálculo: Função fibonacci(n)
I. Representação da função em fluxo de controle
Passos do fluxo de controle:
1. Início da função.
2. Verificação da condição if n == 0 or n == 1.
- Se verdadeiro: Retorna 1.
- Se falso: Passa para o próximo passo.
3. Chamadas recursivas fibonacci(n - 1) + fibonacci(n - 2).
4. Retorno do valor calculado.

II. Estruturando o Grafo de Fluxo
Um grafo de controle representa os caminhos possíveis da execução:
- Nó: Representa um ponto de decisão ou instrução.
- Aresta: Representa a transição entre nós.
- Componentes conexos (P): A função é uma unidade única, então P = 1.

Nós (N):
1. N1: Início da função.
2. N2: Verificação if n == 0 or n == 1.
3. N3: Retorno 1 (caso base da recursão).
4. N4: Chamada recursiva fibonacci(n - 1).
5. N5: Chamada recursiva fibonacci(n - 2).
6. N6: Cálculo do retorno return fibonacci(n - 1) + fibonacci(n - 2).

Número total de nós: N = 6.

Arestas (E):
1. N1 -> N2: Do início para a verificação do if: 1 aresta.
2. N2 -> N3: Caso n == 0 or n == 1, retorna 1: 1 aresta.
3. N2 -> N4: Caso contrário, faz chamada recursiva fibonacci(n - 1): 1 aresta.
4. N2 -> N5: Caso contrário, faz chamada recursiva fibonacci(n - 2): 1 aresta.
5. N4 -> N6: Resultado da chamada recursiva de fibonacci(n - 1): 1 aresta.
6. N5 -> N6: Resultado da chamada recursiva de fibonacci(n - 2): 1 aresta.
7. N4 -> N1: Chamada recursiva gerando nova execução: 1 aresta.
8. N5 -> N1: Chamada recursiva gerando nova execução: 1 aresta.

Número total de arestas: E = 8.

III. Aplicando a fórmula

Agora, usamos a fórmula da complexidade ciclomática:

$M = E - N + 2P$

Substituímos os valores:

$M = 8 - 6 + 2(1)$ -> $M = 4$

IV. Interpretando o resultado

A complexidade ciclomática da função é 4.

Isso significa que há 4 caminhos independentes no grafo de fluxo de controle:

Caminho 1 - Execução direta para o caso base

- Se n == 0 ou n == 1, a execução segue este fluxo:

N1 → N2 (Início da função para verificação do if)

N2 → N3 (Caso base, retorna 1)

Caminho completo: N1 → N2 → N3

Caminho 2 - Cálculo do Fibonacci para fibonacci(n - 1)

- Quando n > 1, a função chama fibonacci(n - 1). Esse fluxo ocorre assim:

N1 → N2 (Início da função para verificação do if)
N2 → N4 (Chamada recursiva fibonacci(n - 1))
N4 → N6 (Resultado de fibonacci(n - 1) volta para N6)
Caminho completo: N1 → N2 → N4 → N6

Caminho 3 - Cálculo do Fibonacci para fibonacci(n - 2)

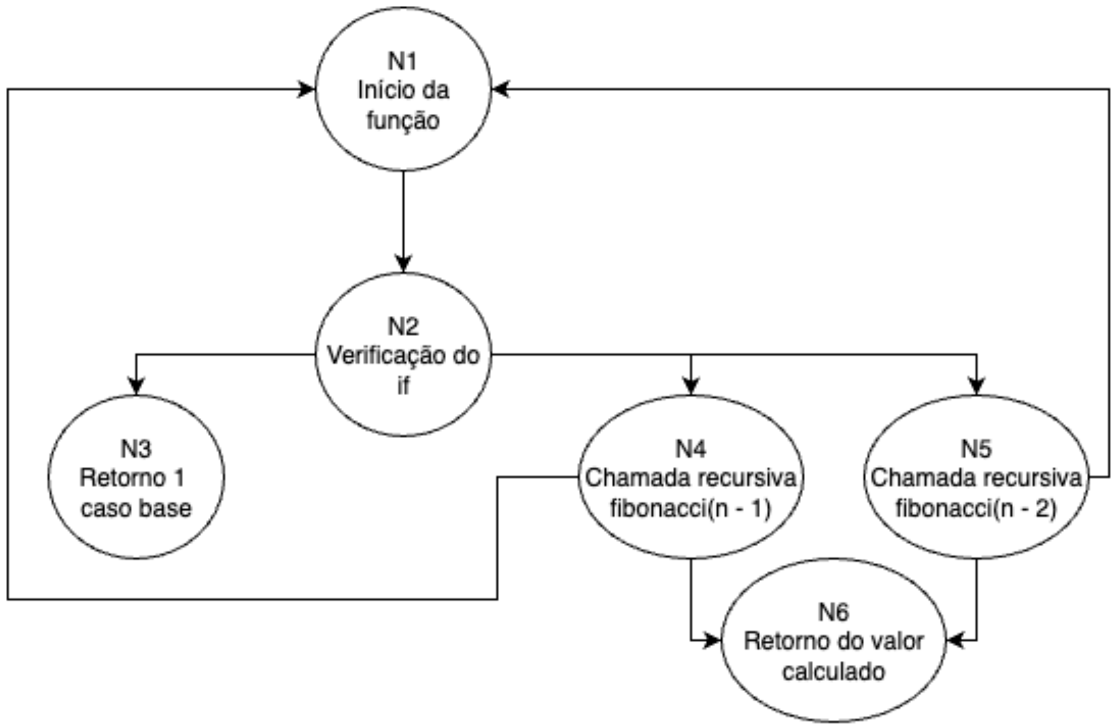
- Similar ao caso anterior, mas para fibonacci(n - 2):
N1 → N2 (Início da função para verificação do if)
N2 → N5 (Chamada recursiva fibonacci(n - 2))
N5 → N6 (Resultado de fibonacci(n - 2) volta para N6)
Caminho completo: N1 → N2 → N5 → N6

Caminho 4 - Soma dos resultados e retorno final

- O cálculo final retorna a soma de fibonacci(n - 1) + fibonacci(n - 2). O fluxo é:
N1 → N2 (Início da função para verificação do if)
N2 → N4 (Chamada fibonacci(n - 1))
N2 → N5 (Chamada fibonacci(n - 2))
N4 → N6 (Retorno de fibonacci(n - 1))
N5 → N6 (Retorno de fibonacci(n - 2))
N6 → Retorno final (Soma dos resultados)
Caminho completo: N1 → N2 → N4 → N5 → N6 → Retorno final

Resumo dos 4 caminhos
1. Caso base: N1 → N2 → N3
2. Chamada fibonacci(n - 1): N1 → N2 → N4 → N6
3. Chamada fibonacci(n - 2): N1 → N2 → N5 → N6
4. Retorno final: N1 → N2 → N4 → N5 → N6 → Retorno final

Desenhando o grafo de fluxo:


N1
Início da
função
N2
Verificação do
if
N3
Retorno 1
caso base
N4
Chamada recursiva
fibonacci(n - 1)
N5
Chamada recursiva
fibonacci(n - 2)
N6
Retorno do valor
calculado