

Lista 2 – 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 2 - Gabarito

Complexidade Ciclomática

1) O algoritmo abaixo implementa uma função que encontra o maior valor dentro de uma lista.

```

1 def max(lista):
2     temp = lista[0] # Inicializa temp com o primeiro elemento da lista
3     for i in range(1, len(lista)): # Percorre os elementos do índice 1 até o final
4         if temp < lista[i]: # Se encontrar um elemento maior que temp
5             temp = lista[i] # Atualiza temp com esse valor
6     return temp # Retorna o maior valor encontrado

```

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 max(lista)

I. Representação da função em fluxo de controle

Passos do fluxo de controle:

1. Início da função.
2. Inicialização da variável temp.
3. Início do laço for.
4. Verificação da condição if temp < lista[i].
 - Se verdadeiro: Executa temp = lista[i].
 - Se falso: Passa direto para a próxima iteração.
5. Retorno da variável temp.

Observação sobre o passo 2: Vamos supor que a inicialização da variável temp fosse temp = [] ao invés de temp = lista[0]:

- Neste caso, não seria contado como um nó porque: A inicialização de temp não afetaria nenhuma decisão no código. O laço for e o if não dependeriam do valor inicial de temp. O fluxo de controle não mudaria por causa dessa atribuição.

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 $P = 1$ (é um bloco único de código sem chamadas externas a outras funções ou estruturas separadas).

Nós (N):

1. N1: Início da função.
2. N2: Inicialização de temp.
3. N3: Início do laço for.
4. N4: Verificação do if.
5. N5: Ação dentro do if (temp = lista[i]).
6. N6: Retorno final.

Número total de nós: $N = 6$.

Arestas (E):

1. N1 -> N2: Do Início para a inicialização de temp: 1 aresta.
2. N2 -> N3: Da inicialização de temp para o laço for: 1 aresta.
3. N3 -> N4: Do laço for para o if: 1 aresta.
4. N4 -> N5: Condição verdadeira no if para temp = lista[i]: 1 aresta.
5. N4 -> N3: Condição falsa no if para a próxima iteração do laço: 1 aresta.
6. N5 -> N3: De temp = lista[i] para a próxima iteração do laço: 1 aresta.
7. N3 -> N6: Saída do laço para o retorno: 1 aresta.

Número total de arestas: $E = 7$.

III. Aplicando a fórmula

Agora, usamos a fórmula da complexidade ciclomática:

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

Substituímos os valores:

$$M = 7 - 6 + 2(1) \rightarrow M = 3$$

IV. Interpretando o resultado

- A complexidade ciclomática da função é 3.
- Isso significa que há 3 caminhos independentes no grafo de fluxo de controle:

Caminho 1 – Lista vazia:

- O laço for não executa nenhuma iteração, pois a lista não contém elementos.
- O fluxo segue diretamente para o retorno.
- Nesse caso, o comportamento depende da implementação: se temp foi inicializado com algum valor padrão, esse valor é retornado. Em implementações reais, pode até lançar exceção (como ValueError em max([]) no Python).

Caminho 2 – Nenhuma atualização de temp:

- O laço percorre todos os elementos da lista.
- A condição if temp < lista[i] nunca é verdadeira (todos os elementos são menores ou iguais a temp).
- O programa retorna o valor inicial de temp sem alterações.

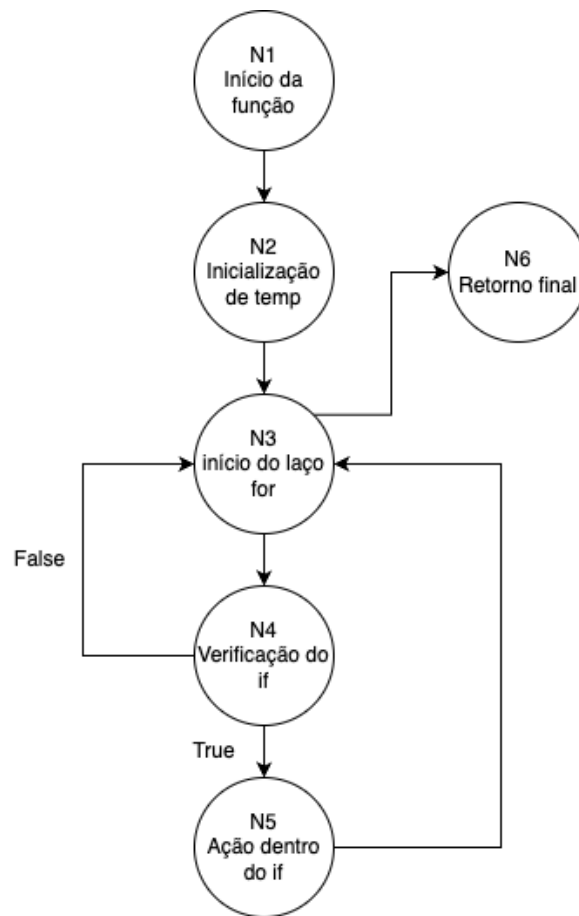
Caminho 3 – Pelo menos uma atualização de temp:

- Em uma ou mais iterações, a condição if temp < lista[i] é verdadeira.
- temp é atualizado uma ou várias vezes até armazenar o maior valor encontrado.
- No final, retorna o maior elemento da lista.

Resumo dos caminhos independentes:

- Caminho 1: A lista está vazia, o laço for não executa nenhuma iteração e o programa retorna diretamente o valor inicial de temp.
- Caminho 2: O laço percorre todos os elementos e não atualiza temp, retornando o valor inicial.
- Caminho 3: O laço encontra ao menos um número maior que temp e atualiza seu valor (podendo ser uma ou várias vezes). No final, retorna o maior valor encontrado.

Desenhando o grafo de fluxo:



2) O algoritmo abaixo retorna o maior e o menor valor dentro de uma lista de números.

```

1 def max_min(lista):
2     temp = [lista[0], lista[0]] # Inicializa temp com o primeiro elemento como maior e menor
3     for i in range(1, len(lista)): # Percorre a lista do índice 1 até o final
4         if temp[0] < lista[i]: # Se encontrar um número maior que temp[0]
5             temp[0] = lista[i] # Atualiza temp[0] com esse valor (maior número)
6         elif temp[1] > lista[i]: # Se encontrar um número menor que temp[1]
7             temp[1] = lista[i] # Atualiza temp[1] com esse valor (menor número)
8     return temp # Retorna uma lista [maior_valor, menor_valor]
  
```

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 max_min(lista)

I. Representação da função em fluxo de controle

Passos do fluxo de controle:

1. Início da função.
2. Inicialização da variável temp com [lista[0], lista[0]].
3. Início do laço for.
4. Verificação da condição if temp[0] < lista[i].
 - Se verdadeiro: Executa temp[0] = lista[i].
 - Se falso: Passa para a verificação do elif.
5. Verificação da condição elif temp[1] > lista[i].
 - Se verdadeiro: Executa temp[1] = lista[i].
 - Se falso: Passa direto para a próxima iteração.
6. Retorno da variável temp.

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 $P = 1$ (é um bloco único de código sem chamadas externas a outras funções ou estruturas separadas).

Nós (N):

1. N1: Início da função.
2. N2: Inicialização de temp.
3. N3: Início do laço for.
4. N4: Verificação do if.
5. N5: Ação dentro do if (temp[0] = lista[i]).
6. N6: Verificação do elif.
7. N7: Ação dentro do elif (temp[1] = lista[i]).
8. N8: Retorno final.

Número total de nós: $N = 8$.

Arestas (E):

1. N1 -> N2: Do Início para a inicialização de temp: 1 aresta.
2. N2 -> N3: Da inicialização de temp para o laço for: 1 aresta.
3. N3 -> N4: Do laço for para a verificação do if: 1 aresta.
4. N4 -> N5 (se a condição if temp[0] < lista[i] for verdadeira): da verificação do if para a ação dentro do if, onde o valor de temp[0] é atualizado.
5. N4 -> N6 (se a condição if temp[0] < lista[i] for falsa): da verificação do if para a verificação do elif temp[1] > lista[i].
6. N6 -> N7 (se a condição elif temp[1] > lista[i] for verdadeira): da verificação do elif para a ação dentro do elif, onde o valor de temp[1] é atualizado.
7. N6 -> N3 (se a condição elif temp[1] > lista[i] for falsa): da verificação do elif de volta para o início do laço for para a próxima iteração.
8. N5 -> N3: da ação dentro do if de volta para o início do laço for para a próxima iteração.
9. N7 -> N3: da ação dentro do elif de volta para o início do laço for para a próxima iteração.
10. N3 -> N8: do início do laço for para o retorno final.

Número total de arestas: $E = 10$.

III. Aplicando a fórmula

Agora, usamos a fórmula da complexidade ciclomática:

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

Substituímos os valores:

$$M = 10 - 8 + 2(1) \rightarrow M = 4$$

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 – Lista vazia:

- O laço for não executa nenhuma iteração.
- O fluxo segue direto para o retorno.
- Dependendo da implementação, pode retornar o valor inicial de temp ou lançar uma exceção (no Python, max_min([]) normalmente falharia).

Caminho 2 – Nenhuma atualização de temp:

- O laço percorre todos os elementos da lista.
- Nem a condição do if (temp[0] < lista[i]) nem a do elif (temp[1] > lista[i]) são verdadeiras em nenhuma iteração.
- O vetor temp permanece igual ao inicial e é retornado sem alterações.

Caminho 3 – Atualização somente do máximo:

- Em pelo menos uma iteração, a condição do if (temp[0] < lista[i]) é verdadeira.

- `temp[0]` é atualizado (uma ou várias vezes) durante a execução do laço.
- A condição do `elif` nunca é verdadeira, então `temp[1]` não sofre alterações.
- O retorno final contém o maior valor encontrado e o mínimo inicial.

Caminho 4 – Atualização somente do mínimo ou de ambos (máximo e mínimo):

- Em pelo menos uma iteração, a condição do `elif` (`temp[1] > lista[i]`) é verdadeira, atualizando `temp[1]`.
- Além disso, em outros momentos, o `if` pode também ser verdadeiro, atualizando `temp[0]`.
- No final, o retorno contém o par `[máximo, mínimo]` atualizado de acordo com os elementos da lista.

Resumo dos caminhos independentes:

- Caminho 1: A lista está vazia, o laço não executa e o programa retorna o valor inicial de `temp` (ou lança exceção).
- Caminho 2: O laço percorre todos os elementos e não atualiza `temp`, retornando o valor inicial.
- Caminho 3: O laço encontra pelo menos um valor maior que `temp[0]` e atualiza apenas o máximo.
- Caminho 4: O laço encontra pelo menos um valor menor que `temp[1]` (e possivelmente também valores maiores), atualizando mínimo e/ou máximo antes de retornar.

Desenhando o grafo de fluxo:



PUC Minas

