MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Ouro Preto
Departamento de Computação e Sistemas
Campus João Monlevade

NOÇÕES DE ANÁLISE DE COMPLEXIDADE DE ALGORITMOS

Curso: Engenharia Elétrica/Sistemas de Informação — 1º semestre de 2024
Disciplina: Algoritmos e Estrutura de Dados I — Data: 10/04/24
Professor: Alexandre Magno de Sousa

1 FUNÇÕES DE CUSTO E ANÁLISE ASSINTÓTICA

1. Encontre a função de custo associada à complexidade de tempo para a instrução de interesse indicada em cada trecho de código a seguir.

(a)
```
void function(int n){
    if (n == 1)
        return;
    for (i = 1; i <= n; i++){
        for (j = 1; j <= n; j++){
            printf("*"); // <-- instrução de interesse
            break;
        }
    }
}
```

Solution:
O laço de repetição mais interno chega a executar apenas uma única vez por causa do comando `break` dentro do laço.
Por causa disso, não é necessário mapear o laço mais interno em um somatório, pois ele executa APENAS uma única vez.
Assim, o somatório que representa esse trecho de código é

$$
\begin{aligned}
f(n) &= \sum_{i=1}^{n} 1 \\
f(n) &= n
\end{aligned}
$$

(b)
```
int count = 0;
for (i = 0; i < n; i++) {
   if (i % 2 == 0) {
      for (j = 0; j < n; j++)
         count++;  // <-- instrução de interesse
   }
   else {
      for (j = 0; j < 10; j++)
         count++;  // <-- instrução de interesse
   }
}
```

Solution:
A instrução `if` testa se i é par ou ímpar, dessa forma o laço dentro do "**então**"

executa somente quando i é par e o laço do "**senão**" executa somente quando i é ímpar. Isso significa que o laço do "**então**" executa apenas metade das vezes ($n/2$) e o laço do "**senão**" também executa metade das vezes ($n/2$).
Diante disso, o teto do somatório mais externo será $n/2-1$, aqui o "-1" é devido a condição do laço utilizar apenas relação "$<$" sem igualdade.

$$
\begin{aligned}
f(n) &= \sum_{i=0}^{\frac{n}{2}-1}\left(\sum_{j=0}^{n-1}1+\sum_{j=0}^{9}1\right)\\
f(n) &= \frac{n^2}{2}+5n
\end{aligned}
$$

(c)
```c
for(i = 0; i < n; i++){
   for (j = n; j > i+1; j--){
      printf("plus");  // <-- instrução de interesse
      printf("minus"); // <-- instrução de interesse
   }
}
```

Solution:
Existe uma correção para ser feita para o laço de repetição mais interno quando o mapeamento do laço para a construção do somatório for realizada.
Perceba que o laço possui um controle do contador em formato decrescente onde $j=n$ e enquanto $j>i+1$ o laço continua em execução. Ou seja, a contagem será $n, n-1, n-2, n-3, \ldots, 3, 2$.
Por isso, o laço mais interno executa $n-1$ vezes, assim a correção para o somatório mais interno será $j=i+2$ para iniciar o contador do somatório e o teto será n.
Assim, tem-se

$$
\begin{aligned}
f(n) &= \sum_{i=0}^{n-1}\sum_{j=i+2}^{n}2\\
f(n) &= n^2-n
\end{aligned}
$$

(d)
```c
int count = 0;
for (i = 0; i < n; i++) {
   for (j = i; j < 2*n; j++) {
      count++;  // <-- instrução de interesse
   }
}
```

Solution:

$$f(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{2n-1} 1$$

$$f(n) = \frac{3n^2}{2} + \frac{n}{2}$$

(e)
```
for (i = 0; i < n; i++) {
   for (j = i+1; j < n/2; j++) {
      printf("fee"); // <-- instrução de interesse
      printf("fi");  // <-- instrução de interesse
   }
}
```

Solution:

$$f(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n/2-1} 2$$

Correção do somatório: o contador do somatório interno `j = i + 1` executa até `n/2 - 1`, porém, quando `i` $\geq$ `n/2`, `j = n/2 + 1`, que é maior do que o seu respectivo teto de `n/2 - 1` o que torna a contagem do somatório interno impossível. Diante disso, o somatório mais externo tem que ser corrigido para que o somatório interno possa ser calculado.
Assim, o teto do somatório externo deverá ser `n/2 - 1`. Então, tem-se

$$f(n) = \sum_{i=0}^{n/2-1} \sum_{j=i+1}^{n/2-1} 2$$

Resultado:

$$f(n) = \frac{n^2}{4} - \frac{n}{2}$$

(f) Para o trecho de código a seguir, assuma que n é uma potência de 2, ou seja 2^m

```
for (i = 1; i < n/4; i *= 2) {
   printf("stop short"); // <-- instrução de interesse
}
for (i = 0; i < 2*n; i += 2) {
   printf("more stuff"); // <-- instrução de interesse
}
```

Solution:

$$f(n) \quad = \quad \left(\sum_{i=1}^{n/4-1} 1 \right) + \left(\sum_{i=0}^{2n-1} 1 \right)$$

Correção do 1º somatório: realizando a correção no teto do primeiro somatório, pois o incremento de `i` é potência de 2 por meio da instrução "`i*=2`" e também no valor inicial do contador do laço para "`i = 0`", tem-se

$$f(n) \quad = \quad \left(\sum_{i=0}^{\log_2(\frac{n}{4})-1} 1 \right) + \left(\sum_{i=0}^{2n-1} 1 \right)$$

Os valores gerados pelo contador do primeiro laço são potências de 2. Por exemplo, para $n = 32$ a seguinte sequência para os valores de `i` são: $1, 2, 4$ (faça um programa com esse trecho de código e veja quantas vezes ele imprime "**stop short**"). Veja que são apenas 3 números. Como são potências de 2, utilizamos logaritmos para encontrar o teto correto para o somatório. Por exemplo,

$$\log_2 32/4 = \log_2 8 = 3,$$

por isso, precisamos fazer a correção do valor de início do contador do laço, bem como o valor do teto do primeiro somatório.
Resultado final:

$$f(n) \quad = \quad 2n + \log_2 n - 2$$

ATENÇÃO: caso você tenha dúvidas do porquê utilizamos logaritmos aqui na correção do primeiro somatório, veja a solução para encontrar o custo do número de comparações do algoritmo da Busca Binária na seção "*5 Busca Binária: Slide*" da apostila intitulada "*Apostila de Análise de Complexidade - Slides 29, 30, 32 e 33*".

(g)
```
for (i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
        for (k = j+1; k < n; k++)
            count++;  // <-- instrução de interesse
```

Solution:

$$f(n) \quad = \quad \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n-1} 1$$

$$f(n) \quad = \quad \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3}$$

(h) Para o trecho de código a seguir, assuma que n é uma potência de 3, ou seja 3^m

```
for (i = 1; i < n; i++) {
    count++; // <-- instrução de interesse
}
for (j = 1; j < n; j *= 3) {
    count++; // <-- instrução de interesse
}
```

Solution:

$$f(n) = \left(\sum_{i=1}^{n-1} 1\right) + \left(\sum_{j=1}^{n-1} 1\right)$$

Semelhante à correção que fizemos na solução da letra (f), para aplicar a correção do somatório deve-se levar em consideração a instrução `j*=3` e a informação de que $n = 3^k$.

Assim, sabe-se que a sequência de valores de `i` são potências de 3. Então, tem-se que

$$f(n) = \left(\sum_{i=1}^{n-1} 1\right) + \left(\sum_{j=0}^{\log_3(n)-1} 1\right)$$

Resultado final:

$$f(n) = n + \log_3(n) - 1$$

(i) Para o trecho de código a seguir, assuma que n é uma potência de 2, ou seja 2^m

```
int count = 0;
for (i = n/2; i < n; i++)
    for (j = 1; j < n; j = 2*j)
        for (k = 1; k < n; k *= 2)
            count++;  // <-- instrução de interesse
```

Solution:

$$f(n) = \sum_{i=n/2}^{n-1} \sum_{j=1}^{n-1} \sum_{k=1}^{n-1} 1$$

Semelhante à correção que fizemos na solução da letra (f), para aplicar a correção do somatório devem ser levadas em consideração as instruções `j=2*j` e `k*=2`, e também a informação de que $n = 2^m$.

Assim, sabe-se que a sequência de valores de `j` e `k` são potências de 2. Então, tem-se que

$$f(n) = \sum_{i=n/2}^{n-1} \sum_{j=0}^{\log_2(n)-1} \sum_{k=0}^{\log_2(n)-1} 1$$

Resultado final:

$$f(n) = \frac{n}{2}(\log_2 n)^2 \quad \text{ou}$$
$$f(n) = \frac{n}{2}\log_2^2 n$$

(j) Para o trecho de código a seguir, assuma que n é uma potência de 2, ou seja 2^m

```
int count = 0;
for (i = n/2; i <= n; i++)
    for (j = 1; j + n/2 <= n; j++)
        for (k = 1; k <= n; k = k*2)
            count++; // <-- instrução de interesse
```

Solution:

$$f(n) = \sum_{i=n/2}^{n} \sum_{j=\frac{n}{2}+1}^{n} \sum_{k=1}^{n} 1$$

Semelhante à correção que fizemos na solução da letra (f), para aplicar a correção do somatório deve-se levar em consideração a instruções `k = k * 2` e também a informação de que $n = 2^m$.

Assim, sabe-se que a sequência de valores de `k` é potência de 2. Então, tem-se que

$$f(n) = \sum_{i=n/2}^{n} \sum_{j=\frac{n}{2}+1}^{n} \sum_{k=1}^{\log_2 n} 1$$

Resultado final:

$$f(n) = \frac{n^2}{4}\log_2 n + n\log_2 n + \log_2 n$$

(k)
```
int i = 1;
int s = 1;
while (s <= n) {
   i++;
   s += i;
   printf("*"); // <-- instrução de interesse
}
```

IMPORTANTE:

- Para àqueles que gostariam de ter uma ferramenta para auxiliar na solução dos somatórios, eu indico a ferramenta `WoflramAlpha`.
- Para utilizar a ferramenta, acesse o site `https://www.wolframalpha.com/` e clique no botão "**`MATH INPUT`**" logo abaixo do campo de entrada onde você pode digitar o somatório. Esse botão habilitará uma barra de tarefas embaixo do campo de entrada que permitirá que você digite os somatórios no formato padrão da matemática.
- Um exemplo para a construção do somatório da letra (g) pode ser visualizado no link `https://bit.ly/4aXQVj4`. A solução está na 1ª área abaixo do campo de entrada chamada "**`Sum`**" e solução expandida do somatório em formato de função pode ser vista na 4ª área chamada "**`Expanded form`**".

2. **Busca Bitônica**: Um vetor é bitônico se ele é formado por uma sequência crescente de inteiros seguida imediatamente por uma sequência decrescente de inteiros. Faça uma função que, dado um vetor bitônico de n inteiros distintos, determine se um dado inteiro está dentro do vetor. A função deverá utilizar aproximadamente $3\log_2 n$ comparações para o pior caso.

   DICA: utilize uma versão adaptada da busca binária para encontrar o valor máximo em $\log_2 n$ comparações, então utilize a busca binária para pesquisar em cada partição do vetor em $\log_2 n$ comparações para cada partição.

3. Assuma que cada uma das expressões a seguir representa a função de custo de complexidade por um algoritmo para resolver um problema de tamanho n. Selecione o termo dominante da expressão que tem o crescimento mais acentuado em função de n e especifique o limite assintótico superior mais próximo possível da função de custo de complexidade de algoritmo e complete a Tabela 1.

Tabela 1: Expressões de funções de custo de complexidade de algoritmos.

Expressão	**Maior Termo**	**Limite Superior**
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n\log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2\log_2 n + n(\log_2 n)^2$	$n^2\log_2 n$	$O(n^2\log_2 n)$
$n\log_3 n + n\log_2 n$	$n\log_3 n,\ n\log_2 n$	$O(n\log n)$
$3\log_8 n + \log_2\log_2\log_2 n$	$3\log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n\log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log_2 n)^2)$
$100n\log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003\log_4 n + \log_2\log_2 n$	$0.003\log_4 n$	$O(\log n)$

4. Para cada função a seguir encontre os pares de valores para c e n_0 para cada função de acordo com o limite assintótico superior e inferior, isto é, O-grande e Ω-grande:

   (a) $f(n) = 3n^3 + 2n^2 + n$.

   Solution:

   – Para $n_0 = 1$ e $c = 6$, $f(n) = O(n^3)$.

   – Para $n_0 = 1$ e $c = 2$, $f(n) = \Omega(n^3)$.

   (b) $f(n) = 2n^2 + 4n - 15$.

   Solution:

   – $2n^2 + 4n - 15 = O(n^2)$, para $n_0 = 9$ e $c = 2.26$.

   – $2n^2 + 4n - 15 = \Omega(n^2)$, para $n_0 = 2$ e $c = 1/4$.

   (c) $f(n) = 3n + 7$.

Solution:

- Para $n_0 = 1$ e $c = 10$, $f(n) = O(n)$.
- Para $n_0 = 1$ e $c = 3$, $f(n) = \Omega(n)$.

(d) $g(n) = 3n^2 + 1$.

Solution:

- Para $n_0 = 1$ e $c = 4$, $f(n) = O(n^2)$.
- Para $n_0 = 1$ e $c = 2$, $f(n) = \Omega(n^2)$.

(e) $f(n) = 5 \log_2 n$.

Solution:

- Para $n_0 = 1$ e $c = 11/2$, $f(n) = O(\log n)$.
- Para $n_0 = 1$ e $c = 9/2$, $f(n) = \Omega(\log n)$.

(f) $f(n) = 2^n + n^3 + n(\log_2 n)^2$.

Solution:

- Para $n_0 = 11$ e $c = 2$, $f(n) = O(2^n)$.
- Para $n_0 = 1$ e $c = 1/2$, $f(n) = \Omega(2^n)$.

Agora utilizando um programa de planilhas como, por exemplo, Microsoft Excel[1] ou Google Sheets[2], apresente gráficos para cada uma das funções anteriores para os limites assintóticos superior (O-grande) e inferior (Ω-grande) para valores de $n \in [0, 10^2]$.

ATENÇÃO: verifique se as constantes c e n_0 encontradas estão corretas, ou seja:

- A linha do gráfico referente ao limite superior mostra que a inequação para O-grande é verdadeira para todo $n \geq n_0$?
- Ou, a linha do gráfico que representa o limite inferior mostra que a inequação para Ω-grande é verdadeira pra todo $n \geq n_0$?

Solution:
Os gráficos para as funções das letras (a) até (f) juntamente com as constantes definidas para os limites superior (O-grande) e inferior (Ω-grande) estão disponíveis no Moodle juntamente com o gabarito dos exercícios no arquivo intitulado

"`Solução da Questão 4 - Gráficos dos limites e definição de constantes.ods`"

[1] https://www.microsoft.com/pt-br/microsoft-365/excel
[2] https://www.google.com/sheets/about/

5. Qual das seguintes afirmações sobre o crescimento assintótico das funções não é verdadeira:
   (a) $2n^2 + 3n + 1 = O(n^2)$.
   (b) $\log n^2 = O(\log n)$.
   (c) Se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, então $f(n) = O(h(n))$.
   (d) Se $f(n) = O(g(n))$, então $g(n) = O(f(n))$
   (e) $2^{n+1} = O(2^n)$.
   (f) $2^{2n} = O(2^n)$.
   (g) $f(n) = O(u(n))$ e $g(n) = O(v(n))$ então $f(n) + g(n) = O(u(n) + v(n))$.
   (h) $f(n) = O(u(n))$ e $g(n) = O(v(n))$ então $f(n) - g(n) = O(u(n) - v(n))$.

Solution:

(a) Verdadeira.

(b) Verdadeira, pois $\log n^2 = 2 \times \log n = O(\log n)$.

(c) Verdadeira.

(d) Falsa, é verdadeira somente se $f(n) = g(n)$.

(e) Verdadeira, pois, $2^{n+1} = 2^n + 2^1$, portanto, fazendo $c = 2$, torna a expressão verdadeira.

(f) Falsa, pois $2^{2n} = 4^n$ e não existem constantes inteiras c e n_0 que faça a expressão $4^n = O(2^n)$ verdadeira.

(g) Verdadeira, vide operações com a notação assintótica $O(n)$.

(h) Falsa, pois não é permitida a operação diferença para notação assintótica $O(n)$.

6. As afirmações apresentas na Tabela 2 mostram algumas propriedades da notação O-grande para as funções $f \equiv f(n)$, $h \equiv h(n)$ e $g \equiv g(n)$. Determine se cada afirmação é `VERDADEIRA` ou `FALSA` e, neste caso, quando a afirmação for falsa, faça a correção da fórmula.

Tabela 2: Afirmações da notação O-grande para as funções f e g.

Afirmação	É `VERDADEIRA` ou `FALSA`?	Se a afirmação é `FALSA`, então escreva a fórmula correta
Regra das somas: $O(f+g) = O(f) + O(g)$	`FALSO`	$O(f+g) = max\{O(f), O(g)\}$
Regra dos produtos: $O(f \times g) = O(f) \times O(g)$	`VERDADEIRO`	
Transitividade: se $g = O(f)$ e $h = O(f)$, então $g = O(h)$	`FALSO`	se $g = O(f)$ e $f = O(h)$, então $g = O(h)$
$5n + 8n^2 + 100n^3 = O(n^4)$	`VERDADEIRO`	
$5n + 8n^2 + 100n^3 = O(n^2 \log n)$	`FALSO`	$5n + 8n^2 + 100n^3 = O(n^3)$

Assuma que $\log n! = \log 1 \times 2 \times 3 \times \ldots \times n = \log 1 + \log 2 + \log 3 + \ldots + \log n \approx n \log n$.