Introdução à complexidade de algoritmos em C

INF0286 | INF0447 – Algoritmos e Estruturas de Dados I

Prof. Me. Raphael Guedes

raphaelguedes@ufg.br

2024

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Algoritmos

- Como resolver um problema no computador?
  - Precisamos descrevê-lo de uma forma clara e precisa.

- Precisamos escrever o algoritmo.
  - Um algoritmo é uma sequência simples e objetiva de instruções.
  - Cada instrução é uma informação que indica ao computador uma ação básica a ser executada.

INF
INSTITUTO DE INFORMÁTICA

Algoritmos

- Vários algoritmos para um mesmo problema.
  - Os algoritmos se diferenciam uns dos outros pela maneira como eles utilizam os recursos do computador.

- Os algoritmos dependem:
  - do tempo que demora para ser executado;
  - da quantidade de memória do computador.

Análise de Algoritmos

- Análise de Algoritmos é a área de pesquisa cujo foco são os algoritmos.
    - Visa responder à pergunta: podemos fazer um algoritmo mais eficiente?
    - Algoritmos diferentes, mas capazes de resolver o mesmo problema, não necessariamente o fazem com a mesma eficiência.

- Exemplo: ordenação de números.

INF
INSTITUTO DE INFORMÁTICA

Análise de Algoritmos

- As diferenças de eficiência podem ser:
  - irrelevantes para um pequeno número de elementos processados;
  - crescer proporcionalmente com o número de elementos processados.
- Dependendo do tamanho dos dados e da eficiência, um programa poderia executar:
  - instantaneamente;
  - de um dia para o outro;
  - por séculos.
- Para comparar a eficiência dos algoritmos foi criada uma medida chamada **complexidade computacional**.
  - Indica o custo de aplicação de um algoritmo.
  - **custo = memória + tempo**
    - memória: quanto de espaço o algoritmo consumirá.
    - tempo: a duração de sua execução.

Análise de Algoritmos

- Importante
  - O custo pode estar associado a outros recursos computacionais, além da memória.
    - Exemplo: tráfego de rede.
  - No entanto, para a maioria dos problemas, o custo está relacionado ao tempo de execução em função do tamanho da entrada a ser processada.

- Para determinar se um algoritmo é o mais eficiente, podemos utilizar duas abordagens:
  - **análise empírica:** comparação entre os programas.
  - **análise matemática:** estudo das propriedades do algoritmo.

INF
INSTITUTO DE INFORMÁTICA

Análise Empírica

Análise empírica

INF
INSTITUTO DE INFORMÁTICA

- Avalia o custo (ou complexidade) de um algoritmo a partir da avaliação da execução dele, quando implementado.
- Análise pela execução de seu programa correspondente.

Análise empírica

- Vantagens:
  - permite avaliar o desempenho em uma determinada configuração de computador/linguagem;
  - considera custos não aparentes;
    - por exemplo, o custo da alocação de memória.
  - permite comparar computadores;
  - permite comparar linguagens.

Análise empírica

- Desvantagens (dificuldades)
  - necessidade de implementar o algoritmo;
    - Depende da habilidade do programador.
  - resultado pode ser mascarado:
    - hardware: computador utilizado;
    - software: eventos ocorridos no momento de avaliação.
  - depende da natureza dos dados:
    - dados reais;
    - dados aleatórios: desempenho médio;
    - dados perversos: desempenho no pior caso.

Contar Instruções dos Algoritmos

- Considere o trecho de um algoritmo que procura o maior valor presente em um vetor **vet_a** contendo **n** elementos e o armazena na variável **menor**.

```
int menor = vet_a[0];
for(int i = 0; i < n; i++){
    if(vet_a[i] >= menor) {
        menor = vet_a[i];
    }
}
```

Contar Instruções dos Algoritmos

- Quantas instruções simples ele executa?
- Instrução simples é uma instrução que pode ser executada diretamente pelo CPU, ou algo muito perto disso. Exemplos:
  - atribuição de um valor a uma variável;
  - acesso ao valor de um determinado elemento do vetor;
  - comparação de dois valores;
  - incremento de um valor;
  - operações aritméticas básicas, como adição e multiplicação.
- Todas as instruções simples possuem o mesmo custo (custo de 01 instrução). Comandos de seleção, possuem custo 0, pois não conta como instrução.

Contar Instruções dos Algoritmos

- Quantas instruções simples ele executa?

```
int menor = vet_a[0];                 - custa 1 instrução
for(int i = 0; i < n; i++){           - custa 2 + 2n instruções
    if(vet_a[i] >= menor) {
        menor = vet_a[i];
    }
}
```

- Ao <u>inicializar</u> o **for** uma operação de atribuição e uma comparação são executadas. Posteriormente, uma instrução de comparação e incremento de i serão executadas **n vezes**.
- Até o momento, o custo do algoritmo é f(n) = 2n + 3.
    - As instruções no corpo (interior) do laço nem sempre são executadas

Contar Instruções dos Algoritmos

- Quantas instruções simples ele executa?

```
int menor = vet_a[0];                   - custa 1 instrução
for(int i = 0; i < n; i++){             - custa 2 + 2n instruções
    if(vet_a[i] >= menor) {             - custa 1 instrução
        menor = vet_a[i];               - pode custar 1 instrução
    }
}
```

- No if temos mais uma instrução: a que acessa o valor do vetor e o atribui a outra variável (menor = vet_a[i]).
- Essas instruções podem ou não ser executadas. Depende do resultado da comparação feita pelo **if**. Isso complica o cálculo do custo do algoritmo.

Contar Instruções dos Algoritmos

- Antes, bastava saber o tamanho do vetor **vet_a**, para definir a função de custo ***f(n)***.
  - Agora, temos que considerar também o conteúdo do vetor.
  - Tome como exemplo os dois vetores abaixo:
    - `int vet01[4] = {1, 2, 3, 4}`
    - `int vet02[4] = {4, 3, 2, 1}`
  - O vetor 01 irá executar o if mais vezes (n vezes), o vetor dois não causará execução do if do algoritmo em discussão.
- É necessário analisar o pior caso. Aquele no qual o maior número de instruções é executado.

Contar Instruções dos Algoritmos

INF
INSTITUTO DE INFORMÁTICA

- Como fica o custo no pior caso?

```
int menor = vet_a[0];                    - custa 1 instrução
for(int i = 0; i < n; i++){              - custa 2 + 2n instruções
    if(vet_a[i] >= menor) {              - custa 1 instrução
        menor = vet_a[i];                - pode custar 1 instrução
    }
}
```

- As operações no interior do for são executadas 2n vezes, portanto:
  - $f(n) = 3 + 2n + 2n$
  - $f(n) = 4n + 3$

Contar Instruções dos Algoritmos

- Quantas instruções simples ele executa?

```
int menor = mat_a[0][0];                    - custa 1 instrução
for(int i = 0; i < n; i++){                 - custa 2 + 2n instruções
    for(j = 0; j < n; j++){                 - custa n * (2 + 2n)
        if(menor < mat_a[i][j])
            menor = mat_a[i][j];            - custa n * 2n
    }
}
```

- $f(n) = 3 + 2n + n(2 + 2n + 2n)$
- $f(n) = 3 + 2n + 2n + 2n^2 + 2n^2$
- $\mathbf{f(n) = 4n^2 + 4n + 3}$

Análise Matemática

Análise Matemática

INF
INSTITUTO DE INFORMÁTICA

- Permite um estudo formal de um algoritmo ao nível ideia.
- Recorre a um computador idealizado e simplificações que buscam considerar somente os custos dominantes do algoritmo.

- Detalhes de baixo nível são ignorados, como:
  - linguagem de programação utilizada;
  - hardware no qual o algoritmo é executado;
  - conjunto de instruções da CPU.

- Permite entender como um algoritmo se comporta à medida que o conjunto de dados de entrada cresce.
- Expressa a relação entre o conjunto de dados de entrada e a quantidade de tempo necessária para processar esses dados.

Comportamento Assintótico

- Será que todos os termos da função `f(n) = 4n + 3` são necessários para saber o seu custo?
- Nem todos os termos são necessários.
  - Certos termos na função podem ser descartados (constantes).
  - Manter apenas os que dizem o que acontece quando o tamanho dos dados da entrada (n) cresce muito.
- Assim, exclui-se o termo 3, resultando em `f(n) = 4n.`
- Remove-se também a constante multiplicada por n (4n).
- Resultado: `f(n) = n.`

- Constantes que multiplicam o termo **n** da função também devem ser descartadas, pois:
  - representam particularidades de cada linguagem e compilador;
  - queremos analisar apenas a ideia por trás do algoritmo, sem influências da linguagem.

Comportamento Assintótico

- Ao descartarmos todos os termos constantes e mantermos apenas o de maior crescimento, obtemos o **comportamento assintótico** do algoritmo.
- Chamamos de comportamento assintótico o comportamento de uma função **f(n)** quando **n** tende ao infinito.
- O termo de maior expoente domina o comportamento da função.

Comportamento Assintótico

- Suprimem-se os termos menos importantes da função e considera-se apenas o termo de maior grau.
    - Isso descreve a complexidade usando somente o seu custo dominante.
    - Se a função não possui nenhum termo multiplicado por n, seu comportamento assintótico é constante (1).

Função custo	Comportamento assintótico
$f(n) = 105$	$f(n) = 1$
$f(n) = 15n + 2$	$f(n) = n$
$f(n) = n^2 + 5n + 2$	$f(n) = n^2$
$f(n) = 5n^3 + 200n^2 + 112$	$f(n) = n^3$

Comportamento Assintótico

- De modo geral, obtém-se a função de custo de um programa simples apenas contando os comandos de laços aninhados.
    - Não possui laço (exceto se houver recursão): **$f(n) = 1$.**
    - Um comando de laço indo de 1 a n: **$f(n) = n$.**
    - Dois comandos de laço aninhados: **$f(n) = n^2$.**
    - E assim por diante.

INF
INSTITUTO DE INFORMÁTICA

Tipos de notação assintótica

- Existem várias formas de análise assintótica:
  - Notação grande-ômega, Ω;
  - Notação grande-O, O;
  - Notação grande-theta, Θ;
  - Notação pequeno-o, o;
  - Notação pequeno-omega, ω.

Notação O-grande (Big-O)

- Existem várias formas de análise assintótica (ver adiante)
- A mais conhecida e utilizada é a notação grande-O (O)

- Evidencia o custo do algoritmo no **pior caso possível** para todas as entradas de tamanho **n**.
- Analisa o limite superior de entrada
- Permite dizer que o comportamento do algoritmo não pode nunca ultrapassar um certo limite

- A ordenação de dados é um problema interessante para entendermos como funciona a notação Grande-O (O), por se tratar de um problema comum em sistemas reais e por possuir uma grande variedade de algoritmos para resolvê-lo.

Notação O-grande (Big-O)

- O que $O(n^2)$ significa para um algoritmo?
- A notação $O(n^2)$ informa que o custo do algoritmo não é, assintoticamente, pior do que n^2.
    - O algoritmo nunca vai ser mais lento do que um determinado limite.
    - Ou seja, o custo do algoritmo original é no máximo tão ruim quanto n^2.
        - Pode ser melhor, mas nunca pior.
        - Limite superior para a complexidade real do algoritmo.

Notação O-grande (Big-O)

- A notação O descreve o **limite assintótico superior** de um algoritmo.
- Notação utilizada para analisar o **pior caso** do algoritmo.

- A notação O(n^2) informa que o custo do algoritmo é, assintoticamente, menor ou igual a n^2.
  - O custo do algoritmo original é no máximo tão ruim quanto n^2.

- Matematicamente, a notação O é definida como:
  - uma função custo **f(n)** é **O(g(n))** se existem duas constantes positivas **c** e **m** tais que, para $n \geq m$, temos $f(n) \leq cg(n)$.
- Em outras palavras, para todos os valores de **n** à direita do valor **m**, o resultado da função custo **f(n)** é sempre menor ou igual ao valor da função usada na notação O, **g(n)**, multiplicada por uma constante **c**.

Notação O-grande (Big-O)

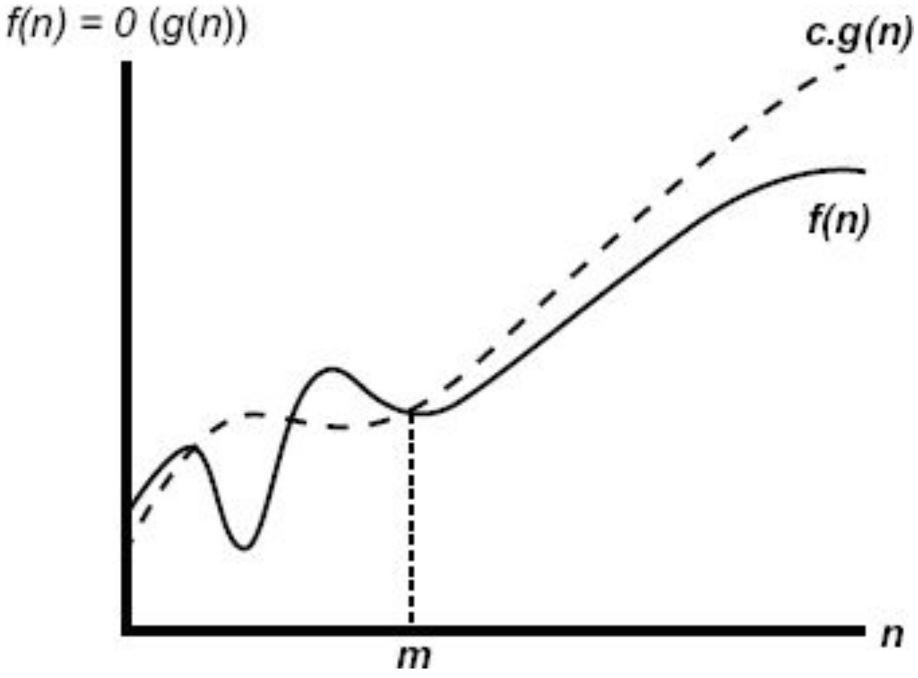

Figura 1. Descreve o comportamento da notação O.

Notação O-grande (Big-O)

INSTITUTO DE INFORMÁTICA

- A notação Grande-O, O, possui algumas operações, sendo a mais importante a regra da soma.
  - Permite a análise da complexidade de diferentes algoritmos em sequência
- Definição: se dois algoritmos são executados em sequência, a complexidade será dada pela complexidade do maior deles.
  - $O(\ f(n)\ ) + O(\ g(n)\ ) = O(\ \max(\ f(n),\ g(n)\ )\ )$

- Exemplos:
  - Dois algoritmos cujos tempos de execução são $O(n)$ e $O(n^2)$, a execução deles em sequência será $O(\max(n,n^2))$ que é $O(n^2)$.
  - Dois algoritmos cujos tempos de execução são $O(n)$ e $O(n \log n)$, a execução deles em sequência será $O(\max(n,n \log n))$ que é $O(n \log n)$.

Notação Ω (ômega grande)

INF
INSTITUTO DE INFORMÁTICA

- A notação Ω descreve o **limite assintótico inferior** de um algoritmo.
- Notação utilizada para analisar o melhor caso do algoritmo.

- A notação Ω (n^2) informa que o custo do algoritmo é, assintoticamente, maior ou igual a n^2.
  - O custo do algoritmo original é no mínimo tão ruim quanto n^2.

- Matematicamente, a notação Ω é definida como:
  - uma função custo **f(n)** é **Ω (g(n))** se existem duas constantes positivas **c** e **m** tais que, para **n ≥ m**, temos **f(n) ≥ cg(n)**.
- Em outras palavras, para todos os valores de **n** à direita do valor **m**, o resultado da função custo **f(n)** é sempre maior ou igual ao valor da função usada na notação Ω, **g(n)**, multiplicada por uma constante **c**.

Notação Ω (ômega grande)

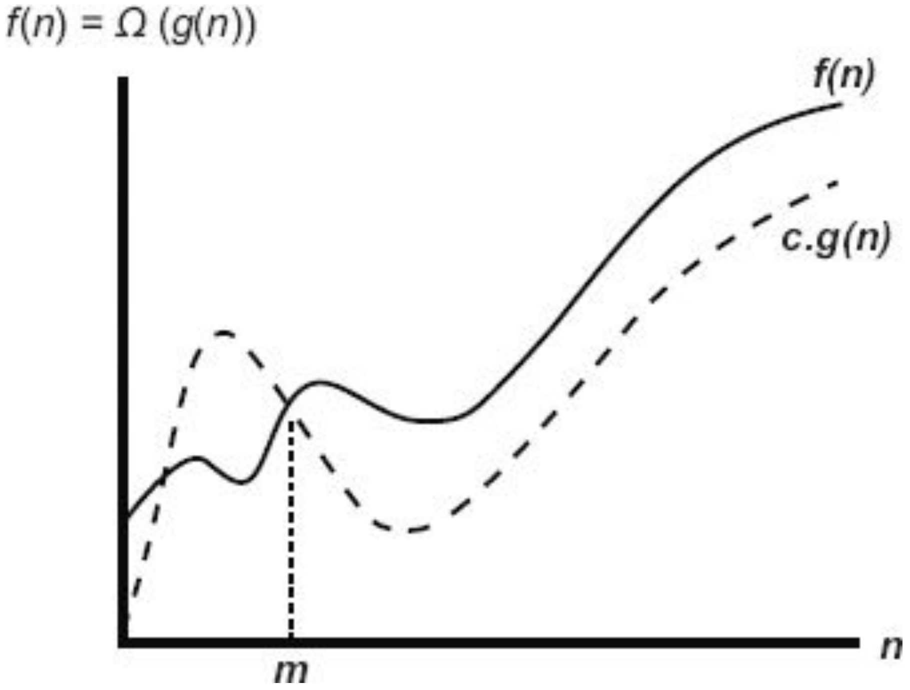

Figura 1. Descreve o comportamento da notação Ω.

Notação Θ (theta grande)

- A notação θ (lê-se, grande theta) descreve o limite **assintótico firme (ou estreito)** de um algoritmo.
- Notação utilizada para analisar os limites inferior e superior do algoritmo.

- A notação $\theta(n^2)$ informa que o custo do algoritmo é, assintoticamente, igual a n^2.
  - O custo do algoritmo original é n^2 dentro de um fator constante acima e abaixo.

- Matematicamente, a notação θ é definida como:
  - uma função custo **f(n)** é **θ(g(n))** se existem três constantes positivas $\mathbf{c_1}$, $\mathbf{c_2}$ e m tais que, para $\mathbf{n \geq m}$, temos $\mathbf{c_1 * g(n) \leq f(n) \leq c_2 * g(n)}$.
- Em outras palavras, para todos os valores de **n** à direita do valor **m**, o resultado da função custo **f(n)** é sempre igual ao valor da função usada na notação θ, **g(n)**, quando esta função é multiplicada por constantes $\mathbf{c_1}$ e $\mathbf{c_2}$.

INF
INSTITUTO DE INFORMÁTICA

Notação Θ (theta grande)

INF
INSTITUTO DE
INFORMÁTICA

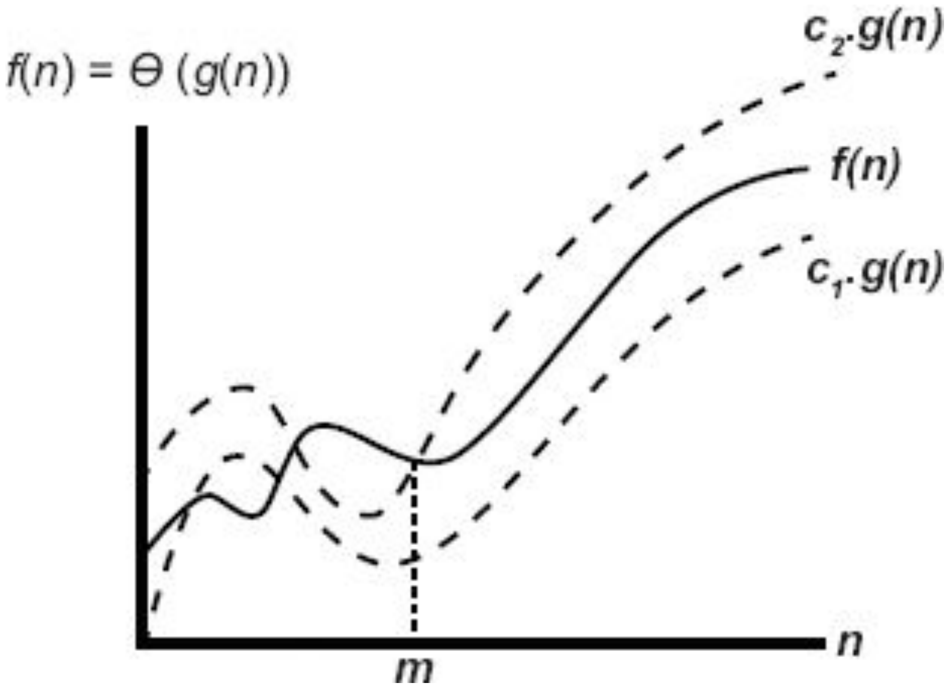

Figura 1. Descreve o comportamento da notação Θ.

Notação o (o-pequeno) e ω (ômega-pequeno)

- As notações **o** e **ω** são muito parecidas com as notações O e Ω, respectivamente.

- As notações O e Ω possuem uma relação de **menor ou igual e maior ou igual**.
- As notações **o** e **ω** possuem uma relação de **menor e maior**, somente.

- Não representam limites próximos da função **f(n)**, apenas estritamente superiores e inferiores.
    - A notação $o(n^2)$ informa que o custo do algoritmo é, assintoticamente, sempre menor do que n^2. Matematicamente, uma função custo $\mathbf{f(n)}$ é $\mathbf{o(g(n))}$ se existem duas constantes positivas $\mathbf{c}$ e $\mathbf{m}$ tais que, para $\mathbf{n \geq m}$, temos $\mathbf{f(n) < c\, g(n)}$.
    - A notação $\omega(n^2)$ informa que o custo do algoritmo é, assintoticamente, sempre maior do que n^2. Matematicamente, uma função custo $\mathbf{f(n)}$ é $\boldsymbol{\omega}\mathbf{(g(n))}$ se existem duas constantes positivas $\mathbf{c}$ e $\mathbf{m}$ tais que, para $\mathbf{n \geq m}$, temos $\mathbf{f(n) > c\, g(n)}$.

Classes de problemas

- O(1) **ordem constante:**
  - as instruções são executadas um número fixo de vezes;
  - não depende do tamanho dos dados de entrada.
- O(log n) **ordem logarítmica**:
  - típica de algoritmos que resolvem um problema transformando-o em problemas menores.
- O(n) ordem linear:
  - uma certa quantidade de operações é realizada sobre cada um dos elementos de entrada.
- O(n log n) **ordem log linear**:
  - típica de algoritmos que trabalham com particionamento dos dados;
  - os algoritmos resolvem um problema, transformando-o em problemas menores, resolvidos de forma independente e depois unidos.

Classes de problemas

- $O(n^2)$ **ordem quadrática**:
  - ocorre quando os dados são processados aos pares.
  - algoritmos desse tipo possuem um aninhamento de dois comandos de repetição.
- $O(n^3)$ **ordem cúbica**:
  - caracterizado pela presença de três estruturas de repetição aninhadas.
- $O(2^n)$ **ordem exponencial**:
  - ocorre quando se usa uma solução de força bruta;
  - não são úteis do ponto de vista prático.
- $O(N!)$: **ordem fatorial**:
  - ocorre quando se usa uma solução de força bruta;
  - não são úteis do ponto de vista prático.
  - possui comportamento muito pior que o exponencial.

Classes de problemas

- A relação entre as classes de complexidades é definida como:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(N!)$$

Classes de problemas: comparação tempo

f(n)	n = 10	n = 20	n = 30	n = 50	n = 100
n	1,0E-05 segundos	2,0E-05 segundos	4,0E-05 segundos	5,0E-05 segundos	6,0E-05 segundos
n log n	3,3E-05 segundos	8,6E-05 segundos	2,1E-04 segundos	2,8E-04 segundos	3,5E-04 segundos
n^2	1,0E-04 segundos	4,0E-04 segundos	1,6E-03 segundos	2,5E-03 segundos	3,6E-03 segundos
n^3	1,0E-03 segundos	8,0E-03 segundos	6,4E-02 segundos	0,13 segundos	0,22 segundos
2n	1,0E-03 segundos	1,0 segundo	2,8 dias	35,7 anos	365,6 séculos
3n	5,9E-02 segundos	58,1 minutos	3855,2 séculos	2,3E+08 séculos	1,3E+13 séculos

Referências

- BACKES, André Ricardo. **Algoritmos e Estruturas de Dados em C**. Rio de Janeiro: LTC, 2023.

INF

INSTITUTO DE INFORMÁTICA

Obrigado!

raphaelguedes@ufg.br
raphaelguedes@inf.ufg.br

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS