Introdução à complexidade de algorimtos

Universidade Federal do Estado do Rio de Janeiro
Bacharelado de Sistemas de Informação
Estruturas de Dados 2
Professora: Vânia Félix Dias

Introdução

- Um algoritmo é um processo sistemático para a resolução de um problema.

- Existem dois aspectos básico: a <u>correção</u> e a <u>análise</u>.

- O primeiro consiste em verificar a exatidão do método e a análise visa à obtenção de parâmetros através de uma prova matemática.

Introdução

- Como exemplo, seja uma sequência de elementos armazenada no vetor S[r], $1 \leq i \leq n$.

- Deseja-se inverter os elementos da sequência no vetor.

- Um algoritmo para resolver esse problema é simples: basta trocar de posição o primeiro com o último elemento, sucessivamente.

Introdução

Algoritmo para inverter uma sequência S[n]

Para i := 1 … piso(n/2) faça
    Temp := S[i]
    S[i] := S[n – i +1]
    S[n – i + 1] := Temp

Recursividade

- É aquele procedimento que contém, em sua descrição, uma ou mais chamadas a si mesmo.

- Um procedimento não recursivo é, pois, aquele em que todas as chamadas são externas.

- Entretanto, muitas vezes há desvantagens ao emprego prático da recursividade.

Recursividade

- Exemplo 1: fatorial (recursivo)

```
Função fat(i)
  F := se i ≤ 1 então 1 senão i * fat(i – 1)
  retornar F
```

- Exemplo 2: fatorial (não-recursivo)

```
  Fat[0] := 1
  Para j := 1 … n faça
      Fat[j] := j * fat [j - 1]
```

Complexidade de Algoritmos

- Característica importante – Tempo de execução
- É possível determiná-lo através de métodos empíricos, naturalmente.
- Em contrapartida, é possível obter uma ordem de grandeza do tempo de execução através de métodos analíticos.

Complexidade de Algoritmos

- Ao contrário do empírico, o analítico visa aferir o tempo de execução de forma independente do computador utilizado, da linguagem e compiladores empregados e das condições locais de processamento.
  - Somente o comportamento assintótico será avaliado;
  - Não serão consideradas constantes aditivas ou multiplicativas;

Complexidade de Algoritmos

- O processo de execução de um algoritmo pode ser dividido em etapas elementares, denominadas *passos*, cada qual consiste na execução de um número fixo de operações básicas cujos tempos de execução são considerados constantes.

- A operação básica de maior frequência de execução no algoritmo é denominada *operação dominante*.

Complexidade de Algoritmos

- Pelo exposto, é natural definir a expressão matemática de avaliação do tempo de execução de um algoritmo como sendo um <u>função</u> que fornece o <u>número de passos</u> efetuados pelo algoritmo <u>a partir de uma certa entrada</u>.

  <u>Tamanho da entrada</u> : n
  <u>Função de tempo</u> : $f(n)$

Complexidade de Algoritmos

- O algoritmo abaixo descreve a computação da matriz soma de duas matrizes.

  Algoritmo 1: Soma de Matrizes

```
Para i := 1 … n faça
    Para j := 1 … n faça
        C[i,j]  := A [i,j] + B[ i,j]
```

Complexidade de Algoritmos

- O algoritmo abaixo descreve a computação da matriz produto de duas matrizes.

  Algoritmo 2: Produto de matrizes

```
Para i := 1 … n faça
    Para j := 1 … n faça
        C[i,j] := 0
        Para k := 1 … n faça
            C[i,j] := C[i,j] + A[i,k] * B[k,j]
```

Complexidade de Algoritmos

- Ambos os algoritmos de soma e produto efetuam as mesmas operações, respectivamente, sempre que A, B forem matrizes dimensão n x n. A variável independente é o parâmetro n.

- Cada passo no algoritmo 1 corresponde à execução de uma soma enquanto o segundo, corresponde ao produto.

Complexidade de Algoritmos

- O número total de passos é igual ao número total de somas e produtos, respectivamente.

- Ou seja, o primeiro efetua n^2 e o segundo n^3.

Complexidade de Algoritmos

- A complexidade tem por objetivo avaliar a eficiência de tempo ou espaço.

- O termo complexidade será empregado com o significado de *complexidade de pior caso*.

Complexidade de Algoritmos

- A noção de complexidade de tempo é descrita a seguir.

- Seja A um algoritmo, $\{E_1,\ldots,E_m\}$ o conjunto de todas as entradas possíveis de A.

Complexidade de Algoritmos

Denote por t_i o número de passos efetuados por A, quando a entrada for E_i. Definem-se:

Complexidade do pior caso = $\max_{E_i \in E} \{t_i\}$

Complexidade do melhor caso = $\min_{E_i \in E} \{t_i\}$

Complexidade do caso médio = $\Sigma_{1 \leq i \leq m} (p_i.t_i)$

Onde p_i é a probabilidade de ocorrência da entrada E_i

Notações

- Quando se considera o número de passos efetuados por um algoritmo, podem-se desprezar constantes aditivas ou multiplicativas.
- Além disso, como interesse é restrito a valores assintóticos, termos de menor grau também podem ser desprezados.
- Assim, um valor de número de passos igual a $n^2 + n$ será aproximado para n^2 .

Notação O

Sejam *f* e *h* funções reais positivas na variável inteira *n*.

Diz-se que *f* é *O(h)*, escrevendo-se *f=O(h)*, quando existir uma constante $c > 0$ e um valor inteiro n_0, tal que:

$$n > n_0 \Longrightarrow f(n) \leq c \cdot h(n)$$

Notação O

$$n > n_0 \Longrightarrow f(n) \leq c \,.\, h(n)$$

- Ou seja, a função *h* atua como um limite superior para valores assintóticos da função *f*.

- Por exemplo, no algoritmo 1 a complexidade é de $O(n^2)$ e no algoritmo 2 é $O(n^3)$.

Notação O

- É verdade que $2n^2 + 100n = O(n^2)$? Prove.

- É verdade que $10 + 4n = O(n^0) = O(1)$? Prove.

- Escreva a seguinte função em notação O:

  $4n^2 + 10 \log n + 500$.

Notação Ω

- Assim como a notação *O* é útil para descrever limites superiores assintóticos

- A notação Ω é empregada para limites inferiores assintóticos.

Notação Ω

Sejam *f* e *h* funções reais positivas na variável inteira *n*.

Diz-se que *f* é Ω*(h)*, escrevendo-se *f*=Ω*(h)*, quando existir uma constante $c > 0$ e um valor inteiro n_0, tal que:

$$n > n_0 \Longrightarrow f(n) \geq c \,.\, h(n)$$

Notação Ω

- Exemplos:

  $n^2 \in \Omega\ (n)$

  $n \in \Omega\ (\log n)$

  Se $f(n) = 7n^3 + 5$ e $g(n) = 2n$, então $g(n) \in \Omega\ f(n)$

Notação Θ

- Esta notação é para exprimir limites superiores justos.

- Sejam *f, g* funções reais positivas da variável inteira *n*. Diz-se que *f* é $\Theta(g)$, escrevendo-se $f = \Theta(g)$, quando ambas as condições forem verdadeiras:

$$f = O(g) \text{ e } g = O(f)$$

Notação Θ

- A notação Θ exprime o fato de que duas funções possuem a mesma ordem de grandeza assintótica.

Notação Θ

Certo ou errado?

Se f, g são funções tais que

$$f = \mathrm{O}(g) \text{ e } g = \Omega(f)$$

então $f = \Theta(g)$

Algoritmo Ótimo

- Intuitivamente, um algoritmo ótimo é aquele que apresenta a menor complexidade dentre todos os possíveis algoritmos existentes para resolver o mesmo problema.

- Assim como a notação O é conveniente para exprimir complexidade, a notação Ω é utilizada para limites inferiores.

Algoritmo Ótimo

- Existem limites inferiores naturais, como, por exemplo, o tamanho da entrada.

- De modo geral, o interesse é determinar a função que represente o maior limite inferior possível para um problema.

- Analogamente, para um certo algoritmo, o interesse é encontrar a função representativa da menor complexidade de pior caso.

Algoritmo Ótimo

- A determinação de complexidade justas é realizada, sem dificuldades, para uma grande quantidade de algoritmos conhecidos.

- O cálculo de limites inferiores, de modo geral, não é um problema simples. Esse cálculo se baseia no desenvolvimento de propriedades matemáticas do problema, independente dos algoritmos empregados.

Algoritmo Ótimo

- Considere o problema da ordenação de um conjunto de *n* elementos.

- Obviamente, o tamanho da entrada do problema é dado por *n*.

- O limite inferior trivial para a solução é, portanto, $\Omega(n)$.

Algoritmo Ótimo

- Contudo, há uma prova matemática de que $\Omega(n \log n)$ é um limite inferior.

- Por outro lado, existem algoritmos conhecidos de ordenação, cujas complexidades são $O(n \log n)$.

- Isso permite concluir que tais algoritmos são ótimos e que o limite $\Omega(n \log n)$ é o melhor possível.