INF 112: Programação II

Aula 06

- **Introdução à análise de complexidade – parte 3**

  Analisando alguns exemplos

Fábio R. Cerqueira, UFV, DPI, frcerqueira@gmail.com

Sequência de passos para analisar um algoritmo

- Escolha o parâmetro *n* que indicará o tamanho da entrada;
- Identifique a operação básica do algoritmo;
- Checar se o número de vezes que a operação básica é executada pode variar para diferentes entradas de mesmo tamanho. Se sim, então ivestigar *melhor caso*, *caso médio* e *pior caso*, separadamente. Nós aqui só investigaremos o <u>*pior caso*</u>;
- Determine a função $C(n)$ que expressa o número de vezes que a operação básica é executada em função do tamanho da entrada. Note que $C(n)$ (no caso de algoritmos iterativos) será dada por algum somatório que deverá ser simplificado, quando possível, por regras conhecidas;
- Indique a classe de eficiência do algoritmo.

Exemplo 1: Maior elemento

ALGORITHM *MaxElement*$(A[0..n-1])$
//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

- **<u>Tamanho da entrada</u>: A medida mais natural para o tamanho da entrada aqui é o número de elementos no arranjo, i.e., *n*.**

- **<u>Operação básica</u>: A comparação.**

- **Note que o número de comparações será o mesmo para todos os arranjos de tamanho *n* (portanto, não há a necessidade de analisar pior caso, melhor caso e caso médio).**

Exemplo 1: Maior elemento

```
ALGORITHM MaxElement(A[0..n − 1])
    //Determines the value of the largest element in a given array
    //Input: An array A[0..n − 1] of real numbers
    //Output: The value of the largest element in A
    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

- **Denotemos por *C*(*n*) o número de vezes que a comparação será executada;**
- **O algoritmo faz uma comparação para cada iteração do laço, que é repetido para cada valor da variável *i* no intervalo de 1 a *n*-1;**
- $C(n) = \Sigma_{1 \leq i \leq n-1} 1 = n - 1$

Exemplo 1: Maior elemento

- **Encontrando a classe de eficiência:**

  Neste caso, é muito fácil a aplicação da definição da notação O. Se tomarmos $c = 1$ e $n_0 = 0$, fica claro que:

  $$n\text{-}1 \leq 1n,\ n \geq 0.$$

  Portanto, $n\text{-}1 \in O(n)$.

Exemplo 2: Problema da unicidade de elementos

ALGORITHM *UniqueElements*$(A[0..n-1])$

```
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n − 1]
//Output: Returns "true" if all the elements in A are distinct
//         and "false" otherwise
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

- **<u>Tamanho da entrada</u>: O número *n* de elementos no arranjo;**
- **<u>Operação básica</u>: A comparação entre dois elementos;**
- **Há dois cenários de pior caso:**
  - **Arranjos que não possuem qualquer par de elementos iguais;**
  - **Arranjos em que os dois últimos elementos são os únicos que formam par de elementos iguais.**

Exemplo 2: Problema da unicidade de elementos

- Num caso ou noutro, uma comparação é realizada para cada repetição do laço mais interno;
- As iterações do laço mais interno são repetidas para cada valor do laço mais externo. Desta forma, obtem-se:

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

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

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \in O(n^2) \text{ (prove).}$$

We also could have computed the sum $\sum_{i=0}^{n-2}(n-1-i)$ faster as follows:

$$\sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2}.$$

- Note que este resultado era previsível. No pior caso, o algoritmo precisa comparar todos os $n(n-1)/2$ (combinação dos n elementos 2 a 2) possíveis pares.

Example 3: Multiplicação de matriz

```
ALGORITHM MatrixMultiplication(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
    //Multiplies two n-by-n matrices by the definition-based algorithm
    //Input: Two n-by-n matrices A and B
    //Output: Matrix C = AB
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0.0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

- **Tamanho da entrada: Ordem *n* da matriz.**
- **Operação básica: Vamos escolher a multiplicação.**
- **Aqui também não haveria necessidade de analisar os três tipos de cenário.**

Example 3: Multiplicação de matriz

The number of multiplications made for every pair of specific values of variables i and j is

$$\sum_{k=0}^{n-1} 1,$$

and the total number of multiplications $M(n)$ is expressed by the following triple sum:

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1.$$

$$= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

- **A classe de eficiência do algoritmo é obviamente $O(n^3)$.**

Note que

- Se quisermos agora estimar o tempo de execução do algoritmo para uma máquina particular, podemos fazê-lo utilizando o produto:

  $$T(n) \approx c_{\mathrm{m}} M(n) = c_{\mathrm{m}} n^3,$$

  onde c_{m} é o tempo gasto para realizar uma multiplicação na máquina particular.

- Obviamente que obteríamos uma estimativa mais precisa se levássemos em conta o tempo gasto para as adições também:

  $$T(n) \approx c_{\mathrm{m}} M(n) + c_a A(n) = c_{\mathrm{m}} n^3 + c_a n^3 = (c_m + c_a) n^3,$$

  onde c_a é o tempo para realizar uma adição. Note que as estimativas, portanto, só se distinguem pelas suas constantes <u>não pela taxa de crescimento</u>.

Exemplo 4: Contando os dígitos binários

ALGORITHM *Binary*(n)
  //Input: A positive decimal integer n
  //Output: The number of binary digits in n's binary representation
  $count \leftarrow 1$
  while $n > 1$ **do**
    $count \leftarrow count + 1$
    $n \leftarrow \lfloor n/2 \rfloor$
  return $count$

- **Vamos pegar a divisão realizada no corpo do *while* como sendo a operação básica.**

Exemplo 4: Contando os dígitos binários

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
$count \leftarrow 1$
while $n > 1$ **do**
    $count \leftarrow count + 1$
    $n \leftarrow \lfloor n/2 \rfloor$
return $count$

- **O que chama a atenção neste exemplo é o fato de que a variável de controle do laço toma apenas alguns poucos valores entre seus limites inferior e superior.**
- **Já que o valor de n cai mais ou menos à metade a cada repetição do laço, o número de vezes que o laço é executado é aproximadamente $\log_2 n$.**
- **A classe de eficiência do algoritmo é, portanto, $O(\log n)$.**

Exercícios

- **Os exercícios a seguir foram retirados do livro:**

  Introduction to the Design & Analysis of Algorithms

  Autor: Anany Levitin

- **Os enunciados foram mantidos em inglês**

Exercícios

Consider the following algorithm.

Algorithm *Mystery*(n)
//Input: A nonnegative integer n
$S \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow S + i * i$
return S

a. What does this algorithm compute?

b. What is its basic operation?

c. How many times is the basic operation executed?

d. What is the efficiency class of this algorithm?

e. Suggest an improvement or a better algorithm altogether and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Exercícios

Consider the following algorithm.

Algorithm *Secret*$(A[0..n-1])$
//Input: An array $A[0..n-1]$ of n real numbers
$minval \leftarrow A[0]$; $maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] < minval$
    $minval \leftarrow A[i]$
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval - minval$

Answer the questions a–e of Problem 1 about this algorithm.

Exercícios

Consider the following algorithm.

Algorithm *Enigma*$(A[0..n-1, 0..n-1])$
//Input: A matrix $A[0..n-1, 0..n-1]$ of real numbers
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i,j] \neq A[j,i]$
            return false
return true

Answer the questions a–e of Problem 1 about this algorithm.

Exercícios

Improve the implementation of the matrix multiplication algorithm (see Example 3) by reducing the number of additions made by the algorithm. What effect will this change have on the algorithm's efficiency?