



Técnicas de Programação III

Análise de Algoritmos

(Continuação)

Aula ministrada em: 28/08/2007

Prof. Mauro L. C. Silva



Objetivos da Aula

- Entender a Análise e a Complexidade de Algoritmos



Interpretação da Complexidade de Tempo

- Qual é o comportamento assintótico predominante de um algoritmo em função do tamanho do conjunto de dados a ser processado. Por exemplo:
 - se é linear,
 - polinomial (quadrático, cúbico, etc),
 - logarítmico ou
 - exponencial.



Análise Assintótica

- Para a análise do comportamento de algoritmos existe toda uma terminologia própria.
- Para o cálculo do comportamento de algoritmos foram desenvolvidas diferentes medidas de complexidade.
- A mais importante delas e que é usada na prática é chamada de Ordem de Complexidade ou Notação-O ou *Big-Oh*.



Análise Assintótica

- A Notação-O me fornece a Ordem de Complexidade ou a Taxa de Crescimento de uma função .
- Para isso, não consideramos os termos de ordem inferior da complexidade de um algoritmo, apenas o termo predominante.

Análise Assintótica

- Exemplo:

Um algoritmo tem complexidade $T(n) = 3n^2 + 100n$.

- Nesta função, o segundo termo tem um peso relativamente grande, mas a partir de $n = 11$, é o termo n^2 que "dá o tom" do crescimento da função: uma parábola.

- A constante 3 também tem uma influência irrelevante sobre a taxa de crescimento da função após um certo tempo.

- Por isso dizemos que este algoritmo é da ordem de n^2 ou que tem complexidade $O(n^2)$.



Cálculo da Complexidade de Tempo

- Um exemplo intuitivo:

inteiro somaCubos (inteiro n)

inteiro i, somaParcial;

início

```
1          somaParcial <- 0;
2          para i de 1 até n faça
3              somaParcial <- somaParcial + i * i * i;
4          fim para
5          retorne somaParcial;
fim
```



Análise Assintótica

- Análise:
- As declarações não tomam tempo nenhum.
- A linha 4 também não toma tempo nenhum.
- As linhas 1 e 5 contam uma unidade de tempo cada.
- A linha 3 conta 4 unidades de tempo (2 multiplicações, uma adição e uma atribuição) e é executada n vezes, contando com um total de $4n$ unidades de tempo.
- A linha 2 possui custos implícitos de inicializar o i , testar se é menor que n e incrementá-lo. Contamos 1 unidade para sua inicialização, $n + 1$ para todos os testes e n para todos os incrementos, o que perfaz $2n + 2$ unidades de tempo.
- O total perfaz $6n + 4$ unidades de tempo, o que indica que o algoritmo é $O(n)$, da Ordem de Complexidade n , ou seja, linear.



Regras para o Cálculo

- Laços Para-Faça e outros:
 - O tempo de execução de um laço é, no máximo, a soma dos tempos de execução de todas as instruções dentro do laço (incluindo todos os testes) multiplicado pelo número de iterações.
- Laços Aninhados:
 - Analise-os de dentro para fora.
 - O tempo total de execução de uma instrução dentro de um grupo de laços aninhados é o tempo de execução da instrução multiplicado pelo produto dos tamanhos de todos os laços.
 - Exemplo $O(n^2)$:
 - para i de 1 até n
 - para j de 1 até n
 - k <- k + 1;
 - fim para
 - fim para



Regras para o Cálculo

- Instruções Consecutivas:
 - Estes simplesmente somam, sendo os termos de ordem menor da soma ignorados.
 - Exemplo $O(n)+O(n^2) = O(n^2)$

```
para i de 1 até n
    a[i] <- 0;
fim para
para i de 1 até n
    para j de 1 até n
        a[i] <- a[j] + k + 1;
    fim para
fim para
```



Regras para o Cálculo

- IF/THEN/ELSE:

- Considerando-se o fragmento de código abaixo:

- se **cond** então

- expressão1

- senão

- expressão2

- fim se

- O tempo de execução de um comando IF/THEN/ELSE nunca é maior do que o tempo de execução do teste **cond** em si mais o tempo de execução da maior dentre as expressões **expressão1** e **expressão2**.

- Ou seja: se expressão1 é $O(n^3)$ e expressão2 é $O(n)$, então o teste é $O(n^3) + 1 = O(n^3)$.



```
UniaoInter(inteiro n)
inteiro conjA[n],conjB[n],conjU[n],conjI[n],i,j;
inicio
  para i de 1 até n faça
    conjA[i] <- valor;
  fim para
  para i de 1 até n faça
    conjB[i] <- valor;
  fim para
  para i de 1 até n faça
    conjU[i] <- conjA[i];
  fim para
  para i de 1 até n faça
    conjU[i+n] <- conjB[i];
  fim para
  para i de 1 até n faça
    para j de 1 até n faça
      se (conjA[i] = conjB[j]) então
        conjI[i] = conjA[i];
      fim se
    fim para
  fim para
fim
```



```
UniaoInter(inteiro n)
inteiro conjA[n],conjB[n],conjU[n],conjl[n],i,j,k;
inicio
  para i de 1 até n faça
    conjA[i] <- valor;
  fim para
  para i de 1 até n faça
    conjB[i] <- valor;
  fim para
  para i de 1 até n faça
    conjU[i] <- conjA[i];
    conjU[i+n] <- conjB[i];
  fim para
  k <- 1;
  para i de 1 até n faça
    para j de 1 até n faça
      se (conjA[i] = conjB[j]) então
        conjl[k] = conjA[i];
        k <- k + 1;
      fim se
    fim para
  fim para
fim
```