Complexidade de Algoritmos

Análise e Correção de Algoritmos

Algoritmo Correto

Usamos para provar que um algoritmo é correto o conhecido Laço Invariante ou Loop Invariante – do inglês *Loop Invariant.*

Laço em programação é um bloco de instruções que é executado repetidamente um número finito de vezes.

Laço invariante é uma característica (ou condição) que ocorre em um laço que é verdadeira imediatamente antes (e depois) de cada iteração do laço.

Laço Invariante

Vamos verificar no algoritmo Insertion-Sort um exemplo de laço invariante.

O algoritmo recebe como entrada um Arranjo A com n chaves, $A=\{a_1,a_2,...,a_n\}$, e faz a ordenação dessas chaves.

O método de ordenação por inserção divide o Arranjo A inicial em duas partes, uma ordenada e a outra desordenada e então insere os elementos da parte desordenada na parte ordenada comparando tal elemento a partir da última posição da porção ordenada em direção a primeira posição procurando a posição correta do elemento.

Insertion-Sort

algoritmo Insertion-Sort(A)

1. para j = 2 até n faça
2. chave = A[j]
3. i = j – 1
4. // inserir A[j] no subarranjo ordenado A[1...j-1]
5. enquanto i > 0 e A[i] > chave faça
6. A[i+1]=A[i];
7. i = i -1
8. A[i+1] = chave

Exemplo Laço Invariante

J – indica a chave atual sendo ordenada

A[1 … j-1] - é o subarranjo ordenado de A

A[j+1 … n] – é o subarranjo desordenado de A

Observando o laço mais externo do algoritmo (para) temos com invariante nele a característica de que o subarranjo A[1 … j-1] estar sempre ordenado antes de qualquer iteração do laço, mesmo antes da primeira iteração.

Laço Invariante e Algoritmo Correto

Usamos o laço invariante para mostrar que um algoritmo é correto, para tal é preciso provar três detalhes sobre um loop invariante.

1. **Inicialização**: O invariante deve ser verdadeiro antes da primeira iteração do laço.

2. **Manutenção**: O invariante, se é verdadeiro antes de uma iteração i, permanecerá verdadeiro antes da iteração i+1.

3. **Término**: Ao fim da execução do loop o invariante fornece uma propriedade que mostra que o algoritmo é correto.

Insertion-Sort está correto ???

Loop Invariante :arranjo A[1 ..j-1] ordenado

Inicialização: quando j=2, o subarranjo A[1..j-1] é composto por apenas um elemento, A[1] , que obviamente está ordenado.

Manutenção: informalmente: o corpo do laço exterior para funciona deslocando A[j-1], A[j-2], A[j-3] …. , uma posição para a direita até encontrar a posição correta para o elemento em A[j] (linhas de 4 a 7), e nesse ponto insere o valor em A[j] (linha 8).

Formalmente é necessário estabelecer um invariante para o laço interno enquanto, o que faremos mais a frente.

Término: o laço externo para encerra quando j = n+1, substituindo j no arranjo do loop invariante temos que A[1 … n] é o subarranjo ordenado, porém A[1 .. n] é o arranjo inteiro e deste modo o arranjo inteiro está ordenado, o que mostra que o algoritmo é correto.

Análise do Algoritmo

- Analisar um algoritmo consiste em prever os recursos que o algoritmo necessitará:
  - Memória
  - Largura de banda de comunicação
  - Hardware de computador,
  - Entre outras.
- Mas o item principal é o tempo de computação

Análise de Algoritmo

- Usaremos o Modelo de máquina RAM de computação para analisar os algoritmos:
  - As instruções são executadas sequencialmente, sem execução concorrente.
  - As instruções são:
    - Aritméticas
    - Movimentação de dados,
    - De controle
  - Cada uma dessas demora um período constante para ser executada.
  - Os dados são do tipo inteiro ou ponto-flutuante
  - Não considera hierarquia de memória.

Análise do Insertion-Sort

- O tempo de execução depende diretamente do tamanho da entrada e o custo que cada instrução consome.
- Vamos definir uma expressão matemática em função do tamanho da entrada para o Insertion-Sort.

Insertion-Sort

algoritmo Insertion-Sort(A)

	Custo	Vezes
1. <u>para</u> j = 2 <u>até</u> n <u>faça</u>	c1	n
2. chave = A[j]	c2	n-1
3. i = j – 1	c3	n-1
4. // inserir A[j] no subarranjo ordenado A[1...j-1]	0	n-1
5. <u>enquanto</u> i > 0 <u>e</u> A[i] > chave <u>faça</u>	c5	$\Sigma^{n}_{j=2}\ t_j$
6. A[i+1]=A[i];	c6	$\Sigma^{n}_{j=2}\ (t_j - 1)$
7. i = i -1	c7	$\Sigma^{n}_{j=2}\ (t_j - 1)$
8. A[i+1] = chave	c8	n – 1

Onde t_j é o número de vezes que o teste do loop <u>enquanto</u> na linha 5 é executado para esse j

Insertion-Sort

- Pior caso

$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5.\Sigma^{n}_{j=2}\ t_j+c_6.\Sigma^{n}_{j=2}(t_j-1)+\ c_{7.}\Sigma^{n}_{j=2}(t_j-1)+\ c_8(n-1)$

Uma vez que o pior caso é a entrada ordenada inversamente isso faz $t_j = j$ e assim:

$\Sigma^{n}_{j=2}\ j = \ (n.(n-1)/2)\ -1$

$\Sigma^{n}_{j=2}(j-1) = (n.(n-1)/2)$

Temos que:

$T(n) = c_1n+c_2(n-1)+c_4(n-1)+c_5.((n.(n-1)/2)-1)+c_6.(n.(n-1)/2)+\ c_{7.}(n.(n-1)/2)+\ c_8(n-1)$

$T(n) = ((c_5+c_6+c_7)/2)\ \mathbf{n^2} + (c_1+c_2+c_4+(c_5+c_6+c_7)/2)+c_8)\ \mathbf{n} – (c_1+c_2+c_4+c_8)$

$T(n) = a\mathbf{n^2} + b\mathbf{n} – c$

Insertion-Sort

- Melhor Caso

$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5(n-1)+ c_8(n-1)$

$T(n) = (c_1+c_2+c_4+c_5+c_8)\mathbf{n} – (c_2+c_4+c_5+c_8)$

$T(n) = d\mathbf{n} + e$

Insertion-Sort

- Caso Médio

  No caso médio $t_j = j/2$