Análise de algoritmos

Conceitos básicos

Notas

Conteúdo

Notas

Problema de ordenação

Entrada
Uma sequência de n números $\langle a_1, a_2, \ldots, a_n \rangle$

Saída
Uma permutação (reordenação) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ da sequência de entrada tal que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Como resolver o problema de ordenação?

Notas

Ordenação por inserção

- Semelhante a maneira como muitas pessoas ordenam as cartas de um baralho
- Abordagem **incremental**: a cada iteração a porção do vetor ordenada é incrementada
- Eficiente para ordenar um número pequeno de elementos

Notas

Ordenação por inserção

```
insertion-sort(A)
1 for j = 2 to A.length
2     chave = A[j]
3     # insere A[j] na sequência ordenada A[1..j-1]
4     i = j - 1
5     while i > 0 and A[i] > chave
6         A[i + 1] = A[i]
7         i = i - 1
8     A[i + 1] = chave
```

Como saber se a ordenação por inserção funciona corretamente?

Notas

Prova de corretude

- Usamos invariantes de laço para entender por que um algoritmo é correto.

Inicialização Ela é verdadeira antes da primeira iteração do laço.
Manutenção Se for verdadeira antes de uma iteração do laço, ela permanecerá verdadeira antes da próxima iteração.
Término Quando o laço termina, a invariante nos fornece uma propriedade útil para mostrar que o algoritmo é correto.

Invariante do laço principal do `insertion-sort`

No começo de cada iteração do laço for das linhas 1 a 8, o subarranjo $A[1..j-1]$ consiste nos elementos contidos originalmente em $A[1..j-1]$, mas em sequência ordenada

Veja os detalhes nas páginas 13 e 14.

Notas

Ordenação por inserção

```
insertion-sort(A)
1 for j = 2 to A.length
2     chave = A[j]
3     # insere A[j] na sequência ordenada A[1..j-1]
4     i = j - 1
5     while i > 0 and A[i] > chave
6         A[i + 1] = A[i]
7         i = i - 1
8     A[i + 1] = chave
```

Como saber se a ordenação por inserção é eficiente?

Notas

Análise do tempo de execução

Analisar um algoritmo

Significa **prever** os recursos que ele necessitara durante a sua execução.

- É preciso um modelo da tecnologia de implementação que será utilizada
  - Máquina de Turing? Muito diferente dos computadores reais
  - RAM (*random-acess machine* - máquina de acesso aleatório)
- Características da RAM
  - As instruções são executadas uma após a outra
  - Instruções aritméticas, de movimentação de dados e de controle
  - Cada uma das instruções demora um tempo constante
  - Não considera a hierarquia de memória

Notas

Análise do tempo de execução

- O recurso que queremos prever para o `insertion-sort` é o tempo
- O **tempo de execução** de um algoritmo em uma determinada entrada é o número de operações primitivas executadas
- O tempo de execução depende da quantidade de itens na entrada (**tamanho da entrada**)
- Vamos assumir que cada linha executada consome um período constante de tempo

Notas

Análise do tempo de execução

`insertion-sort(A)`	custo	vezes
`1 for j = 2 to A.length`	c_1	n
`2  chave = A[j]`	c_2	$n-1$
`3  # insere A[j] ...`	0	$n-1$
`4  i = j - 1`	c_4	$n-1$
`5  while i > 0 and A[i] > chave`	c_5	$\sum_{j=2}^{n} t_j$
`6  A[i + 1] = A[i]`	c_6	$\sum_{j=2}^{n}(t_j-1)$
`7  i = i - 1`	c_7	$\sum_{j=2}^{n}(t_j-1)$
`8  A[i + 1] = chave`	c_8	$n-1$

- t_j é o número de vezes que o teste do laço `while` da linha é executado para o valor j
- Para calcular $T(n)$, o tempo de execução de `insertion-sort`, somamos os produtos das colunas custo e vezes, obtendo

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

Notas

Análise do tempo de execução

- Mesmo para entradas do mesmo tamanho, o tempo de execução pode ser diferente (depende de t_j)
- Como deve ser a entrada para o algoritmo executar com o menor número de operações?
- Como deve ser a entrada para o algoritmo executar com o maior número de operações?

Notas

Análise do tempo de execução

- Melhor caso
  - Ocorre quando o arranjo já está ordenado
  - O teste `A[i] > chave` na linha 5 falha na primeira comparação quando $i = j - 1$
  - Portanto, $t_j = 1$, para $j = 2, 3, \ldots, n$, e o tempo de execução no melhor caso é

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

  - Esse tempo pode ser expresso como $T(n) = an + b$, para constantes a e b
  - É uma **função linear** de n.
  - $T(n) = \Theta(n)$

Notas

Análise do tempo de execução

- Pior caso
  - Ocorre quando o arranjo está em ordem inversa
  - Cada elemento $A[j]$ deve sem comparado com os elementos do subarranjo ordenado $A[1..j-1]$
  - Portanto, $t_j = j$, para $j = 2, 3, \ldots, n$. Como

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

e

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

o tempo de execução no pior caso é

Notas

Análise do tempo de execução

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} + c_8\right) n - (c_1 + c_4 + c_5 + c_8)$$

- Esse tempo pode ser expresso como $T(n) = an^2 + bn + c$, para constantes a, b e c
- É uma **função quadrática** de n.
- $T(n) = \Theta(n^2)$

Notas

Análise do tempo de execução

- Porque preferimos o pior caso?
  - É o limite superior sobre o tempo de execução para qualquer entrada
  - Para alguns algoritmos, ocorre com bastante frequência
  - Muitas vezes, o caso médio é quase tão ruim quanto o pior caso

Notas

Ordenação por intercalação

Veja o capítulo 2 do livro Cormen. Páginas 21 a 28.

Notas

Prova de corretude

Veja o capítulo 2 do livro Cormen. Páginas 21 a 28.

Notas

Análise do tempo de execução

Veja o capítulo 2 do livro Cormen. Páginas 21 a 28.

Notas

Exercícios

- Leitura do capítulo 2 e do apêndice A
- Exercícios 2.1-1 a 2.1-4
- Exercícios 2.2-1 a 2.2-4
- Exercícios 2.3-1 a 2.3-7

Notas

Referências

- Thomas H. Cormen et al. Introdução a Algoritmos. 2ª edição em português. Capítulo 2.

Notas