Projeto e Análise de Algoritmos

Profa. Juliana Kaizer Vizzotto

Roteiro

- Introdução
- Exemplo: ordenação

Introdução

Análise de Algoritmos

- Estudo teórico da *performance* e utilização de recursos em programas de computadores.
- O que é mais importante que *performance*?
- Por que estudar algoritmos e *performance*?

Introdução

- Na prática é fundamental que um programa produza a solução com uso do tempo e de memória razoável.
- O fato de uma algoritmo resolver (teoricamente) um problema não significa que seja aceitável na prática.
- Os recursos de espaço e de tempo requeridos têm grande importância em casos práticos.
- As vezes o algoritmo mais imediato está longe de ser razoável em termos de eficiência.
- Um exemplo é o caso da solução de sistemas de equações lineares: método de Cramer $\times$ Método de Gauss.

Introdução

Notas

- 1 picosegundo: 10^{-12} segundos
- 1 nanosegundo: 10^{-9} segundos
- 1 microsegundo μs: 10^{-6} segundos
- 1 milisegundo ms: 10^{-3} segundos

Introdução

Crammer × Gauss

n	Cramer	Gauss
2	22 μs	50 μs
3	102 μs	159 μs
4	456 μs	353 μs
5	2,35 ms	666 mus
10	1,19 mim	4,59 ms
20	15225	38,63 ms
40	5 x 10^{33}	0,315s

Introdução

Notas

- **Problemas com Tempo linear**: tempo de execução é proporcional ao tamanho da entrada
- Uma máquina, num certo período máximo de tempo tolerável, resolve problemas de tamanho máximo x_1.
- Em um computador 10 vezes mais rápido, o mesmo algoritmo, no mesmo tempo, resolverá um problema de tamanho 10 vezes maior, isto $10x_1$.
- **Problemas com tempo quadrático**: tempo proporcional a n^2 para uma entrada de tamanho n.

Introdução

Exemplo: tempo quadrático

- Considere o tamanho máximo de um problema resolvível em um tempo t na máquina mais lenta é x_3, i.e., $k(x_3)^2 = t$
- Agora suponha a máquina dez vezes mais rápida, i.e., 10t.
- O tamanho do problema resolvido agora sera y=?

$$ky^2 = 10t \;\; ky^2 = 10k(x_3)^2$$
$$y^2 = 10(x_3)^2$$
$$y = \sqrt{10}x_3$$

- Portanto, agora y é aproximadamente $3,16x_3$.

Introdução

Problemas com tempo exponencial

- Leva o tempo 2^n para uma entrada de tamanho n.
- Se x_5 é o tamanho máximo de um problema resolvível num tempo t na máquina mais lenta e y na máquina mais rápida.

$$2^{x_5} = te2^y = 10t$$
$$2^y = 10.2^{x_5}$$
$$y = log_2$$

- O avanço tecnológico da máquina foi bem mais aproveitado pelo primeiro algoritmo.
- Um algoritmo de tempo exponencial, como no exemplo acima, praticamente não tira proveito da rapidez da segunda máquina.

Exemplo: Ordenação

Roteiro

- Vamos analisar um problema específico: ordenação por inserção
- Definiremos um pseudocódigo
- Analisaremos eu tempo de execução: tempo aumenta com o número de itens a serem ordenados
- Projeto de Algoritmos: divisão e conquista
- Algoritmo de ordenação por intercalação
- Análise do tempo de execução de ordenação por intercalação

Exemplo: Ordenação

Ordenação por inserção

- **Entrada**: uma sequência de n números $\langle a_1, a_2, ..., a_n \rangle$
- **Saída**: uma permutação (reordenação) $\langle a'_1, a'_2, ..., a'_n \rangle$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq ... \leq a'_n$
- Chamamos os números que queremos ordenar de chaves
- Ordenação por inserção: eficiente para ordenar um número pequeno de elementos
- Algoritmo funciona de maneira similar como as pessoas ordenam as cartas em um jogo de pôquer.

Exemplo: Ordenação

Ordenação por inserção: *Insertion-sort*

- **Entrada**: array $A[1...n]$ contendo uma sequência de tamanho n que deve ser ordenada
- Os números de entrada são **ordenados no local**: os números são reorganizados dentro do array A.
- O array de entrada A conterá a sequência de saída ordenada quando o *Insertion-sort* terminar

Exemplo: Ordenação

Ordenação por inserção: *insertion-sort*

```
Insertion-sort(A)
  for j <- 2 to comprimento[A]
    do chave <- A[j]
     > inserir A[j] na sequencia ordenada A[1 .. j-1]
      i <- j-1
      while i > 0 e A[i] > chave
        do A[i+1] <- A[i]
           i <- i -1
      A[i+1] < chave
```

Exemplo: Ordenação

Loops invariantes e a correção do *insertion-sort*

- O índice j indica a "carta atual" sendo inserida na mão.
- No início de cada interação do loop `for`, indexado por j, o subarranjo que consiste nos elementos $A[1..j-1]$ constitui a mão atualmente ordenada e os elementos $A[j+1...n]$ correspondem à pilha de cartas ainda na mesa.
- Propriedades de $A[1..j-1]$: **loop invariante**
  - No começo de cada interação do loop `for`, o subarray $A[1...j-1]$ consiste nos elementos contidos originalmente em $A[1...j-1]$, mas em sequência ordenada.
- Usamos **loops invariantes** para nos ajudar a entender por que um algoritmo é correto. Devemos mostrar três detalhes sobre um loop invariante:

Exemplo: Ordenação

Loops invariantes e a correção do *insertion-sort*

- **Inicialização**: Ele é verdadeiro antes da primeira iteração do loop
- **Manutenção**: Se for verdadeiro antes de uma iteração do loop, ele permanecerá verdadeiro antes da próxima.
- **Término**: Quando o loop termina, o invariante nos fornece uma propriedade útil que ajuda a mostrar que o algoritmo é correto

Exemplo: Ordenação

Correção do *insertion-sort*

- **Inicialização**: na primeira iteração temos $j = 2$. Então o subarray $A[1...j-1]$ consiste apenas no único elemento $A[1]$. Esse array é ordenado trivialmente!
- **Manutenção**: informalmente, o corpo do loop `for` funciona deslocando $A[j-1]$, $A[j-2]$, $A[j-3]$ e dai por diante, uma posição à direita até ser encontrada a posição adequada para $A[j]$, e nesse ponto o valor de $A[j]$ é inserido.
- **Término**: o loop `for` termina quando j excede n, isto é, quando $j = n+1$, o que significa que o subarray de $A[1...n]$, o qual é o array inteiro está ordenado!

Exemplo: Ordenação

Exercícios

1. Ilustre a operação de *Insertion-sort* no array $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.
2. Considere o seguinte problema de pesquisa:
   - **Entrada**: uma sequência de n números $A = \langle a_1, a_2, ..., a_n \rangle$ e um valor v
   - **Saída**: um índice i tal que $v = A[i]$ ou o valor especial NIL, se v não aparecer em A.
3. Escreva o pseudocódigo para a pesquisa linear que faça a varredura da sequência, procurando por v. Usando um loop invariante prove que eu algoritmo é correto. Certifique-se que seu loop invariante satisfaz as três propriedades necessárias.

Exemplo: Ordenação

Análise de Algoritmos

- Analisar um algoritmo significa prever os recursos (memória, hardware de computador e **tempo de computação**) de que o algoritmo necessitará
- A análise proporciona identificar um algoritmo mais eficiente
- Para a análise: inicialmente devemos ter um modelo de tecnologia de implementação que será usada.
- Por exemplo: modelo de computação genérico com um único processador, RAM.

Exemplo: Ordenação

Análise do *Insertion-sort*

- O tempo despendido pelo procedimento *Insertion-sort* depende da entrada: a ordenação de mil números demora mais que a ordenação de 3 números.
- Além disso, o *Insertion-sort* pode demorar períodos diferentes para ordenar duas sequências de entrada do mesmo tamanho.
- Em geral, o tempo de duração de um algoritmo cresce com o tamanho da entrada

Exemplo: Ordenação

Análise do *Insertion-sort*

- **Tamanho da entrada**:
  - a medida mais natural é o número de itens de entrada (e.g, o tamanho n do array para ordenação)
  - para outros diversos problemas, como a multiplicação de dois inteiros, a melhor medida de tamanho é o número total de bits necessários para representar a entrada em notação binária comum.
- **Tempo de execução**: é o número de operações primitivas ou etapas executadas. Definimos a noção de etapa (ou passo) de forma que ela seja independente de máquina utilizada. Vamos adotar a seguinte visão:

Exemplo: Ordenação

Análise do *Insertion-sort*

- **Tempo de execução**:
  - Um período constante de tempo é exigido para calcular cada linha do pseudocódigo
  - Uma única linha pode demorar um período diferente de outra linha. Vamos considerar que cada execução da i-ésima linha leva um período constante de tempo c_i, onde c_i é uma constante.
- Análise do *Insertion-sort*: definimos o "custo" de tempo de cada instrução e o número de vezes que cada instrução é executada.
- Para cada $j = 2, 3..., n$, seja t_j o número de vezes que o teste do loop `while` é executado para o valor de j.

Exemplo: Ordenação

Análise do *Insertion-sort*

```
Insertion-sort(A)                     custo   vezes
for j<- 2 to comprimento[A]           c1      n
  do chave <- A[j]                    c2      n-1
  i <- j - 1                          c4      n-1
  while i > 0 e A[i] > chave          c5      S (j=2 n) tj
    do A[i+1] <- A[i]                 c6      S (j=2 n) (tj -1)
       i <- i-1                       c7      S (j=2 n) (tj -1)
  A[i+1] <- chave                     c8      n-1
```