CI165 — Introdução

André Vignatti

31 de julho de 2014

Antes de mais nada...

- Os slides de 6 aulas (**introdução, insertion sort, mergesort, quicksort, recorrências e limitantes de ordenação**) foram originalmente feitos pelos Profs. Cid C. de Souza e Cândida N. da Silva.
  - Nessas 6 aulas, desde então, várias partes foram modificadas, melhoradas ou colocadas ao gosto particular de cada docente. Em particular pelos Profs. Orlando Lee e Pedro J. de Rezende. Algumas contribuições menores foram incluídas por Guilherme Pimentel e Flávio K. Miyazawa. Uma maior alteração desses feita posteriormente por André Vignatti.
  - A todos os envolvidos nesses slides, agradecemos pela gentileza de disponibilizá-los para o curso CI165 - Análise de Algoritmos do DINF-UFPR.
- O restante do material de CI165 foi originalmente feito por André Vignatti.
- Os erros/imprecisões que se encontram devem ser comunicados a `{vignatti}@inf.ufpr.br`

Introdução

O que veremos nesta disciplina?

- "Corretude" de algoritmos
- Estimar quantidade de recursos (tempo, memória) de um algoritmo = análise de complexidade
- Algumas técnicas de projeto de algoritmos: divisão-e-conquista, algoritmos aleatorizados, reduções, etc
- A dificuldade intrínseca de vários problemas: problemas bem resolvidos e difíceis

O que é um algoritmo?

Informalmente, um algoritmo é um passo-a-passo bem definido que:

- recebe valores de entrada (instâncias) e
- produz valores de saída.

Equivalentemente, um algoritmo é uma solução de um problema computacional. Um problema computacional define a relação precisa que deve existir entre a entrada e a saída do algoritmo.

Exemplos de problemas: Fatoração

Problema: determinar os **fatores primos** de um dado número.

Exemplo:

Entrada: 9411461

Saída: 9411461.

Exemplo:

Entrada: 8411461

Saída: 1913 e 4397.

Corretude de algoritmos

- Um algoritmo determinístico (para um certo problema) está correto se, para toda instância do problema, ele pára e devolve uma resposta correta.

- Algoritmos probabilísticos podem errar, mas neste caso, queremos que a probabilidade de errar seja muuuuito pequena.

Exemplos de problemas: ordenação

Definição: um vetor $A[1 \ldots n]$ é crescente se $A[1] \leq \ldots \leq A[n]$.

Problema: rearranjar um vetor $A[1 \ldots n]$ de modo que fique crescente.

Entrada:

1										n
33	55	33	44	33	22	11	99	22	55	77

Saída:

1										n
11	22	22	33	33	33	44	55	55	77	99

Algoritmos e tecnologia

Exemplo: **ordenação de um vetor de n elementos**

- Suponha que os computadores A e B executam $1G$ e $10M$ instruções por segundo, respectivamente. Ou seja, A é **100 vezes mais rápido** que B.
- Algoritmo 1: implementado em A por um excelente programador em linguagem de máquina (ultra-rápida). Executa $2n^2$ instruções.
- Algoritmo 2: implementado na máquina B por um programador mediano em linguagem de alto nível dispondo de um compilador "meia-boca". Executa $50n \log n$ instruções.

Algoritmos e tecnologia

- O que acontece quando ordenamos um vetor de um milhão de elementos? **Qual algoritmo é mais rápido?**
- Algoritmo 1 na máquina *A*:
$$\frac{2.(10^6)^2 \text{ instruções}}{10^9 \text{ instruções/segundo}} \approx 2000 \text{ segundos}$$
- Algoritmo 2 na máquina *B*:
$$\frac{50.(10^6 \log 10^6) \text{ instruções}}{10^7 \text{ instruções/segundo}} \approx 100 \text{ segundos}$$
- Ou seja, *B* foi **VINTE VEZES** mais rápido do que *A*!
- Se o vetor tiver 10 milhões de elementos, esta razão será de 2.3 dias para 20 minutos!

Algoritmos e tecnologia – Conclusões

- Usar um algoritmo adequado traz ganhos extraordinários de desempenho.
- Pode ser até mais importante que o projeto de *hardware*.
- A melhoria talvez não seria obtida só com o avanço da tecnologia.
- Melhorias nos algoritmos produzem avanços em outras componentes básicas das aplicações (pense nos compiladores, buscadores na internet, etc).

A importância dos algoritmos para a computação

- Quais aplicações se beneficiam de algoritmos “eficientes”?
  - rede mundial de computadores
  - comércio eletrônico
  - planejamento da produção de indústrias
  - logística de distribuição
  - simulações físicas e químicas
  - projetos de genoma de seres vivos
  - ...

Dificuldade intrínseca de problemas

- Há problemas onde **não se conhece** algoritmos eficientes para resolvê-los. Eles são chamados problemas $\mathcal{NP}$-completos. Curiosamente, **não foi provado** que tais algoritmos não existem!
- Esses problemas tem a característica notável de que se um deles admitir um algoritmo "eficiente" então todos admitem algoritmos "eficientes".

Dificuldade intrínseca de problemas

Exemplos:

- calcular as rotas dos caminhões de entrega de uma distribuidora de bebidas no Paraná, minimizando a distância percorrida. (vehicle routing)
- calcular o número mínimo de *containers* para transportar um conjunto de caixas com produtos. (bin packing 3D)
- calcular a localização e o número mínimo de antenas de celulares para garantir a cobertura de uma certa região geográfica. (facility location)
- e muito mais...

É importante saber indentificar quando estamos lidando com um problema $\mathcal{NP}$-completo!

Descrição de algoritmos

Podemos descrever um algoritmo de várias maneiras:

- linguagem de alto nível: `C`, `Pascal`, `Java` etc
- linguagem de baixo nível/linguagem de máquina
- em português
- em um pseudo-código de alto nível, como no livro do CLRS

Usaremos essencialmente as duas últimas alternativas

Medindo Computação

- Como medir a eficiência de um algoritmo?
- Não estamos interessados em quem programou, em que linguagem foi escrito e nem qual a máquina foi usada!
- Queremos um critério **uniforme** e **independente de tecnologia** para comparar algoritmos.

Modelo Computacional

Solução: definir um modelo computacional formal.

- O modelo computacional estabelece:
  - **recursos** disponíveis,
  - **instruções básicas**
  - **custo** (= tempo) das instruções.

- Nesse modelo, podemos estimar através de uma análise matemática o tempo que um algoritmo gasta em função do tamanho da entrada
  (= análise de complexidade).

- A análise de complexidade depende sempre do modelo computacional adotado.

Modelo Computacional

Existem vários modelos computacionais, os mais usados são:

- Máquina de Turing
  - É um clássico (since 1936!)
  - Muito baixo nível
  - Muito difícil de descrever algoritmos
  - Muito formal
  - Muito chata! : $-/$
- Máquina RAM
  - Mais alto nível
  - Mais fácil de descrever algoritmos
  - Não perde em formalismo para a Máquina de Turing

Teorema

Os modelos de Máquina de Turing e Máquina RAM são computacionalmente equivalentes.

Máquinas RAM

Salvo mencionado o contrário, usaremos o Modelo Abstrato RAM (Random Access Machine):

- simula máquinas convencionais (de verdade),
- possui um único processador que executa instruções seqüencialmente,
- tipos básicos são números inteiros e reais,
- há um limite no tamanho de cada *palavra de memória*: se a entrada tem "tamanho" n, então cada inteiro/real é representado por $c \log n$ bits onde $c \geq 1$ é uma constante.

  Isto é razoável?

Máquinas RAM

- executa operações aritméticas (soma, subtração, multiplicação, divisão, piso, teto), comparações, movimentação de dados de tipo básico e fluxo de controle (teste *if/else*, chamada e retorno de rotinas) em **tempo constante**,
- Certas operações caem em uma **zona cinza**, por exemplo, exponenciação,
- veja maiores detalhes do modelo RAM no CLRS.

Tamanho da entrada

A complexidade de tempo de um algoritmo é o número de instruções básicas que ele executa em função do tamanho da entrada.

Problema: Fatoração

Entrada: inteiro n

Tamanho: número de bits de $n \approx \lg n = \log_2 n$

Problema: Ordenação

Entrada: vetor $A[1 \ldots n]$

Tamanho: cn, onde c é uma constante

Medida de complexidade e eficiência de algoritmos

- Adota-se uma “atitude pessimista”: análise de pior caso.
- Estuda o comportamento para entradas de tamanho GRANDE: análise assintótica.

Um algoritmo é dito eficiente se a complexidade de tempo é uma função **polinomial**

Por exemplo: n, $3n - 7$, $4n^2$, $143n^2 - 4n + 2$, n^5.

Mas por que polinômios?

Desvantagens do método de análise proposto

- Em uma aplicação real, é possível que “o pior caso” ocorra raramente.
- Não fornece nenhuma informação sobre o comportamento do algoritmo no caso médio.
- A análise de complexidade de algoritmos no caso médio é bastante difícil, principalmente, porque muitas vezes não é claro o que é o “caso médio”.