Estruturas de Dados e Algoritmos

Introdução e Motivação

Dalton Serey
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande

Pra início de conversa...

- O que é um algoritmo?
- Como podemos descrever algoritmos?
- Como devemos avaliar algoritmos?
- O que é análise de algoritmos?

Pra início de conversa...

- O que é um algoritmo?
- Como podemos descrever algoritmos?
- Como devemos avaliar algoritmos?
- O que é análise de algoritmos?

O objetivo desta aula é responder a estas quatro perguntas...

O que é um algoritmo?

É um procedimento a partir do qual calculamos valores, ditos *de saída*, a partir de valores, ditos *de entrada.*

O que é um algoritmo?

É uma sequência finita de instruções computáveis que transformam um conjunto de valores em uma dada situação inicial em uma situação final que satisfaz condições especificas.

Exemplos de algoritmos

- O método para resolver **equações quadráticas**...
- O clássico **algoritmo de Euclides**...
- O **crivo de Eratóstenes**...
- Os métodos ensinados às crianças para **somar**, **subtrair**, **multiplicar** e **dividir inteiros**...
- etc, etc, etc...

Qual a diferença entre programas e algoritmos?

- Um **algoritmo** é uma **idéia**.
- Um **programa** é um **texto** que descreve um sistema computacional.

Qual a diferença entre programas e algoritmos?

- **Programas** são escritos em notações projetadas para que *computadores* a processem:
  - Pascal, C, C++, Java, PHP, etc...
- **Algoritmos** são escritos para que *humanos* os entendam...
  - como ensinam crianças a somar dois números inteiros? alguém escreve algum programa?

Como podemos descrever algoritmos?

- Em princípio, **toda linguagem** serve...
  - português, inglês, chinês, Pascal, Java, assembly, ...
- Mas, há **problemas** que queremos evitar...
  - ambiguidade
  - prolixidade
  - falta de estrutura
  - inadequação para o uso final: programação

Como podemos descrever algoritmos?

- Conclusão: precisamos de algo mais formal!
- Algo que permita:
  - hierarquizar e organizar a descrição do algoritmo
  - descrever aspectos controle de fluxo:
  - reduzir a ambiguidade
  - ser conciso, mas sem ser criptográfico

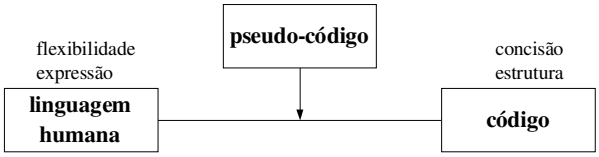
Como podemos descrever algoritmos?

- Conclusão: precisamos de algo mais formal!
- Algo que permita:
  - hierarquizar e organizar a descrição do algoritmo
  - descrever aspectos controle de fluxo:
  - reduzir a ambiguidade
  - ser conciso, mas sem ser criptográfico
- Que tal **linguagens de programação**?

Como podemos descrever algoritmos?

- O problema é que linguagens de programação...
  - são feitas para leitores “burros” (computadores)
  - têm exigências sintáticas muito “chatas”
  - não permitem elipses, mesmo que sejam óbvias
  - obrigam a tratar condições de erro (robustez)
  - obrigam a pensar em detalhes irrelevantes
- Queremos o poder de **abstrair** com inteligência!

Solução: pseudo-código



- A idéia é combinar
  - a *estrutura* e a *concisão* (de linguagem em código)
  - com *flexibilidade* e *expressão* (da linguagem humana)

Usando pseudo-código...

- ... resolva o seguinte problema:

  Especifique um programa que calcule as médias acumuladas de um vetor V contendo *n* inteiros. As médias devem ser armazenadas em um vetor M com *n* reais.

- 5 minutos para resolver...

Exemplo de pseudo-código

```
Real[1..n] MediaAcumulada(Inteiro V[1..n])
1: for i := 1 to n
2:    soma := 0
3:    for j := 1 to i
4:       soma += V[j]
5:    M[i] := soma / i
6: return M
```

E agora? Como avaliamos nosso algoritmo?

Como devemos avaliar um algoritmo?

3 critérios para avaliar algoritmos:

- **corretude**
- **simplicidade**
- **eficiência**

Como devemos avaliar um algoritmo?

corretude

- um algoritmo é correto se para toda entrada especificada a saída correta é produzida

Como devemos avaliar um algoritmo?

simplicidade

- um algoritmo é simples se puder ser facilmente entendido, implementado e mantido;

Como devemos avaliar um algoritmo?

eficiência

- a eficiência de um algoritmo é definida como a inversa da quantidade de **recursos** que requer para seu funcionamento
- que recursos são esses?

Retomemos nosso exemplo...

```
Real[1..n] MediaAcumulada(Inteiro V[1..n])
1: for i := 1 to n
2:    soma := 0
3:    for j := 1 to i
4:       soma += V[j]
5:    M[i] := soma / i
6: return M
```

- Responda:
  - é correto? é simples? é eficiente?

Corretude

```
Real[1..n] MediaAcumulada(Inteiro V[1..n])
1: for i := 1 to n
2:    soma := 0
3:    for j := 1 to i
4:       soma += V[j]
5:    M[i] := soma / i
6: return M
```

- Como podemos **provar** que o algoritmo é correto?

"*Testes servem apenas para provar que um algoritmo tem erros, nunca para provar que está correto.*" (Dijkstra)

Corretude

- A idéia é garantir que:
  - em qualquer possível execução, cada bloco faz exatamente o que esperamos que faça
- Para isso, devemos
  - identificar *o que deve ser feito* por cada bloco
  - identificar *o estado* antes do bloco executar
  - avaliar o efeito do bloco sobre *o estado*
  - caracterizar *o estado* após a execução do bloco

Corretude

- Provar a corretude de algoritmos não é fácil!
- Contudo, é boa prática para a concepção de algoritmos ter em mente os elementos necessários para uma prova...

Invariantes de laço

Método para provar a corretude de blocos de laço:

1. **Declarar a invariante**
   - propriedade relevante para provar a corretude
2. **Verificar o caso base**
   - deve valer antes de executar o laço...
   - e para o primeiro valor da variável de controle
3. **Verificar que a propriedade se mantém a cada passo**
4. **Aplicar a invariante ao caso final**
   - ou seja, para o valor final da variável de controle

Corretude de MediaAcumulada

```
Real[1..n] MediaAcumulada(Inteiro V[1..n])
1: for i := 1 to n
2:    soma := 0
3:    for j := 1 to i
4:       soma += V[j]
5:    M[i] := soma / i
6: return M
```

- Vamos assumir que para qualquer `i`,
  - as linhas 2-4 calculam a soma de `V[1..i]`
- Logo, temos a seguinte abstração

Corretude: invariante de laço

```
  Real[1..n] MediaAcumulada(Inteiro V[1..n])
  1: for i := 1 to n
2-4:    soma := calcula a soma de V[1..i]
  5:    M[i] := soma / i
  6: return M
```

- Se `soma` é a soma de `i` valores, então
- `M[i]`, após executar a linha 5, é necessariamente a média desses valores
- Logo, temos outra abstração...

Corretude: invariante de laço

```
  Real[1..n] MediaAcumulada(Inteiro V[1..n])
  1: for i := 1 to n
2-5:    M[i] := média dos valores de V[1..i]
  6: return M
```

- Agora ficou fácil provar o método inteiro...
- Qual invariante de laço é útil aqui?

Corretude: invariante de laço

```
  Real[1..n] MediaAcumulada(Inteiro V[1..n])
  1: for i := 1 to n
2-5:    M[i] := média dos valores de V[1..i]
  6: return M
```

- Declaração da invariante:

  I(`i`) = "`M[1..i-1]` armazena as médias acumuladas de `V[1..i-1]`"

Corretude: invariante de laço

```
  Real[1..n] MediaAcumulada(Inteiro V[1..n])
  1: for i := 1 to n
2-5:    M[i] := média dos valores de V[1..i]
  6: return M
```

- A base:
  - para `i=1`, a condição *I*(1) é trivialmente verdade...
  - `M[1..0]` e `V[1..0]` são vazios;
  - ou seja, temos base para a "indução"...

Corretude: invariante de laço

```
  Real[1..n] MediaAcumulada(Inteiro V[1..n])
  1: for i := 1 to n
2-5:    M[i] := média dos valores de V[1..i]
  6: return M
```

- A manutenção (*k*-ésima execução do laço):
  - antes do laço devemos assumir que *I*(*k*) é verdade
  - ou seja, que M[1..*k*-1] está ok!
  - durante o laço, calculamos M[*k*] corretamente
  - ao atingirmos o ponto da invariante, M[1..*k*] estará ok
  - fazendo *I*(*k*+1) ser verdadeira...

Corretude: invariante de laço

```
Real[1..n] MediaAcumulada(Inteiro V[1..n])
1: for i := 1 to n
2:   M[i] := soma valores de V[1..i] / i
6: return M
```

- Após a última execução do laço, i = n+1
  - após a última execução do laço, `i=n+1`
  - logo, pela invariante, temos que I(n+1) é verdade:
  - M[1..n] contém as médias acumuladas de V[1..n]
- Portanto, o algoritmo está correto!

Corretude

Não existe algoritmo meio correto...

Simplicidade

- Agora que sabemos que o algoritmo é correto,
- Podemos responder se ele é **simples**?
- Responda:
  - você teria dificuldades em implementá-lo?
  - você teria dificuldades em encontrar erros em uma implementação de outra pessoa?
- Nosso algoritmo é **simples**!
- De fato, o algoritmo é óbvio... (?!)

Eficiência

É intuitivo que a eficiência deve admitir uma **escala de valores**. Portanto, não se trata apenas de verificar se um algoritmo é ou não eficiente... mas, sim, de dizer **o quão eficiente** ele é.

Eficiência

Dissemos que um algoritmo é uma idéia e não uma implementação. Logo, como podemos avaliar os recursos que ele consome?

Eficiência

- Na prática, interessa **comparar algoritmos** que resolvam o **mesmo problema**.
- Logo, o resultado da análise deve refletir apenas o que é intrínseco de cada algoritmo...
- Deve abstrair tudo o que for da implementação ou do ambiente usado para execução.
- Como resolver esse problema?

Eficiência

- Método experimental
  - várias implementações completas
  - um grande número de execuções *controladas*
  - medição cautelosa das variáveis de interesse
  - análise (estatística) dos resultados
- Método analítico
  - idéia: construir um modelo matemático do algoritmo
  - comparar algoritmos com base nos modelos

De volta a nosso exemplo...

- Como podemos avaliar nosso algoritmo?
- Serve rodar o programa e ver o tempo de execução para diferentes entradas?
- Quanto espaço de memória ele precisa?
- Exercício: responda às perguntas acima.

O que é análise de algoritmos?

Para o profissional de computação, analisar um algoritmo significa determinar sua eficiência em termos da quantidade de recursos necessários para sua execução.

- Está implícito que o algoritmo é correto!

E como devemos fazer análise de algoritmos?

- Isso já é assunto da próxima aula...

Resumo

- **Um algoritmo é** uma **sequência finita de instruções** que permite computar valores de saída a partir de valores de entrada.
- **Para descrever algoritmos** usamos **pseudo-código**, uma forma de linguagem que enfatiza **flexibilidade** e **expressão** sem abrir mão de **estrutura** e **concisão**.
- **Avaliamos algoritmos** com base em três critérios fundamentais: **corretude**, **simplicidade** e **eficiência**.
- **Analisar um algoritmo** é avaliar sua **eficiência**, ou seja, determinar a quantidade de recursos que requer para operar.

Exercícios

1. Prove a corretude do algoritmo de ordenação conhecido como *bubblesort*. Escreva um programa em Java que implemente o *bubblesort*.
2. Escreva um programa de teste para o *bubblesort* que submeta vetores com dados aleatórios e temporize o tempo total de ordenação.
3. Execute o programa várias vezes para vetores de tamanhos diferentes. Anote os valores e construa um gráfico com os valores obtidos (sugestão: use o *gnuplot*).
   - abcissas: tamanho do vetor
   - ordenada: média dos tempos de execução observados