# APRESENTAÇÃO DA DISCIPLINA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

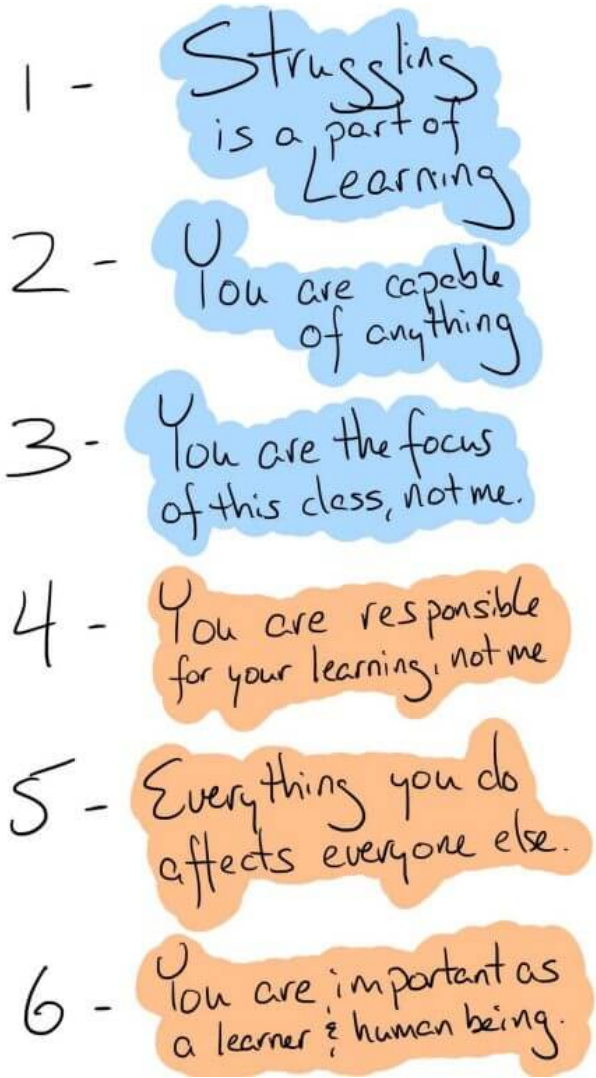
# Agenda

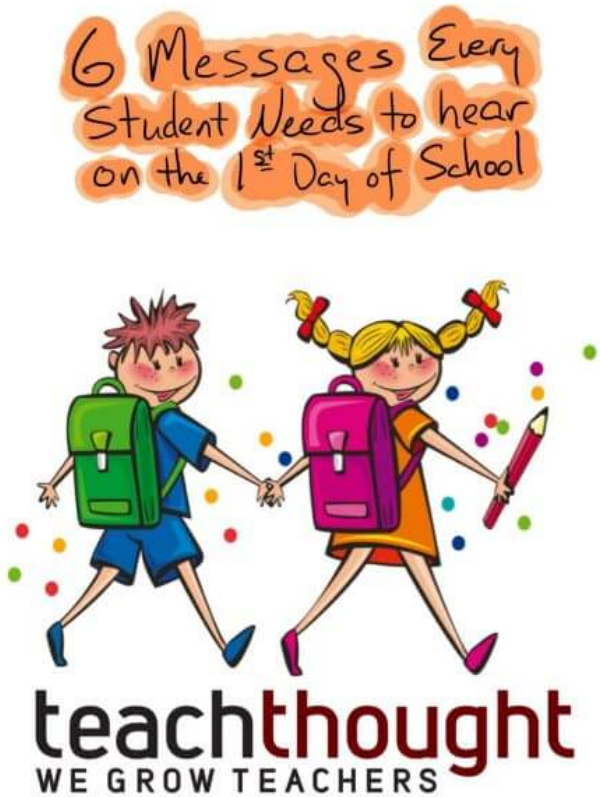
# Sobre mim

- Graduação em CC (UFPE: 2006)
- Mestrado em CC (UFPE: 2010)
- Doutorado em CC (UFPE: 2016)
  - Bremen (DE) e York (UK)
- UFPE: professor (2017 – atual)
- Interesses de pesquisa
  - **Métodos formais**
    (teoria e prática)

# Agenda

# Em primeiro lugar!




[1]

[1] https://www.teachthought.com/pedagogy/6-messages-every-student-hear-first-day-school/

# Agenda

# Por que estudar isso?

Algoritmos e estruturas de dados

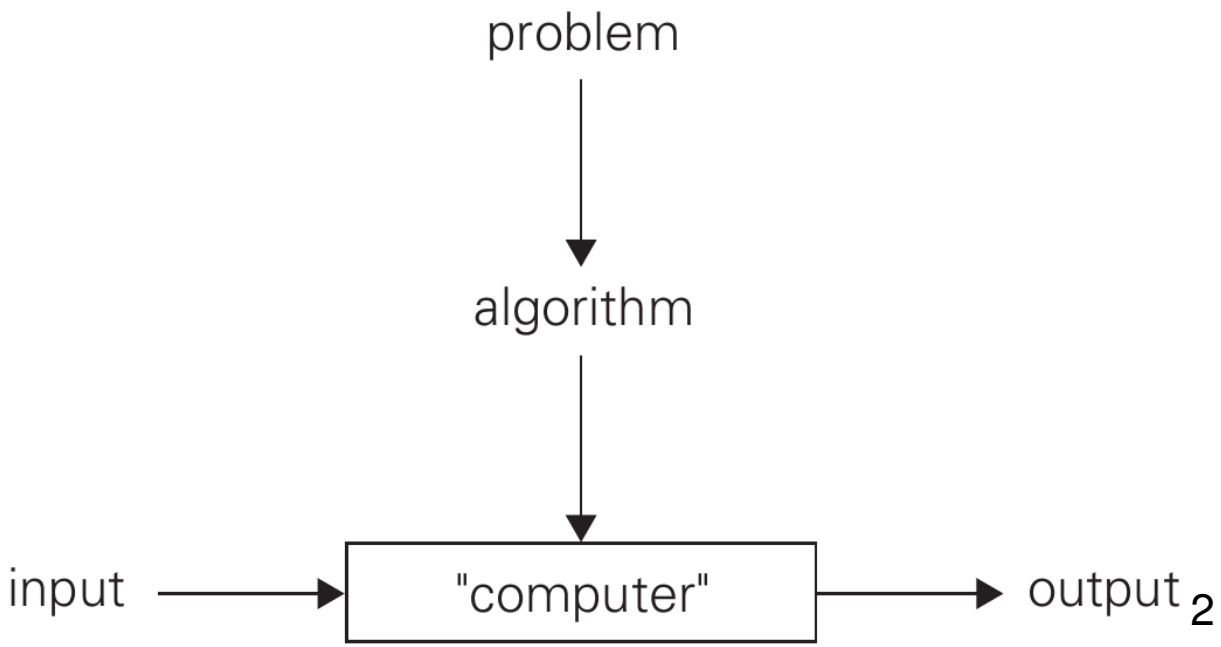
- Perspectiva prática
- Perspectiva teórica
- Pensamento analítico

Não só respostas a problemas, mas procedimentos de obter respostas.

Actually, a person does not *really* understand something until after teaching it to a *computer*, i.e., expressing it as an algorithm... [Donald Knuth]

# O que é um algoritmo?

Uma sequência não-ambígua de instruções para resolver um problema; ou seja, de obter uma saída desejada para qualquer entrada legítima em uma quantidade finita de tempo.




[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

# O que é um algoritmo?

Exact instructions challenge


[3]

Vídeo original:
https://www.youtube.com/watch?v=cDA3_5982h8

[3] Fonte: https://www.youtube.com/watch?v=pdhqwbUWf4U

# Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
2. É preciso especificar as entradas legítimas
3. Um algoritmo pode ser representado de diferentes maneiras
4. Podem existir vários algoritmos para um mesmo problema
5. Analisar os prós/contras dos algoritmos para um mesmo problema

# Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
2. É preciso especificar as entradas legítimas
3. Um algoritmo pode ser representado de diferentes maneiras
4. Podem existir vários algoritmos para um mesmo problema
5. Analisar os prós/contras dos algoritmos para um mesmo problema

Como calcular o gcd de inteiros não-negativos e não-ambos-zero?

- gcd($m$, $n$) = o maior inteiro que divide de forma inteira (sem resto) tanto $m$ como $n$

# gcd – opção 1

Algoritmo de Euclides

- **Passo 1**: Se $n = 0$, retorne o valor de $m$ e pare; caso contrário, vá para o Passo 2.
- **Passo 2**: Divida $m$ por $n$ e atribua o valor do resto da divisão a $r$. Vá para o Passo 3.
- **Passo 3**: Atribua o valor de $n$ a $m$, e o valor de $r$ a $n$. Em seguida, volte para o Passo 1.

Exemplo: gcd(60,24) = ?

# gcd – opção 1

Algoritmo de Euclides

- **Passo 1**: Se $n = 0$, retorne o valor de $m$ e pare; caso contrário, vá para o Passo 2.
- **Passo 2**: Divida $m$ por $n$ e atribua o valor do resto da divisão a $r$. Vá para o Passo 3.
- **Passo 3**: Atribua o valor de $n$ a $m$, e o valor de $r$ a $n$. Em seguida, volte para o Passo 1.

Exemplo: gcd(60,24) = ?

Por que funciona?
R: $\gcd(m, n) = \gcd(n, m \bmod n)$

# gcd – opção 1 (uma representação diferente)

**Algoritmo 1:** gcd(m,n)

**input** : inteiros não-negativos e não-ambos-zero
**output** : maior divisor comum de $m$ e $n$

```
1  while n ≠ 0 do
2  |    r ← m mod n;
3  |    m ← n;
4  |    n ← r;
5  return m;
```

# gcd – opção 2

Algoritmo baseado na verificação consecutiva de inteiros

- **Passo 1**: Atribua o valor de $\min(m, n)$ a $t$. Vá para o Passo 2.
- **Passo 2**: Divida $m$ por $t$. Se o resto da divisão for 0, vá para o Passo 3; caso contrário, vá para o Passo 4.
- **Passo 3**: Divida $n$ por $t$. Se o resto da divisão for 0, retorne o valor de $t$ e pare; caso contrário, vá para o Passo 4.
- **Passo 4**: Decremente o valor de $t$ em 1. Vá para o Passo 2.

Exemplo: $\gcd(60, 24) = ?$

# gcd – opção 2

Algoritmo baseado na verificação consecutiva de inteiros

- **Passo 1**: Atribua o valor de $\min(m, n)$ a $t$. Vá para o Passo 2.
- **Passo 2**: Divida $m$ por $t$. Se o resto da divisão for 0, vá para o Passo 3; caso contrário, vá para o Passo 4.
- **Passo 3**: Divida $n$ por $t$. Se o resto da divisão for 0, retorne o valor de $t$ e pare; caso contrário, vá para o Passo 4.
- **Passo 4**: Decremente o valor de $t$ em 1. Vá para o Passo 2.

Exemplo: $\gcd(60, 24) = ?$

O que acontece se $n$ ou $m$ for zero?

# gcd – opção 3

Algoritmo visto no ensino médio

- **Passo 1**: Encontre os fatores primos de $m$. Vá para o Passo 2.
- **Passo 2**: Encontre os fatores primos de $n$. Vá para o Passo 3.
- **Passo 3**: Identifique todos os fatores primos comuns nas expansões realizadas nos passos 1 e 2. Se $p$ é um fator primo comum ocorrendo $p_m$ vezes em $m$ e $p_n$ vezes em $n$, este deve ser repetido $\min(p_m, p_n)$ vezes. Vá para o Passo 4.
- **Passo 4**: Calcule o produto de todos os fatores primos comuns selecionados no Passo 3. O valor resultante é o maior divisor comum de $m$ e $n$.

Exemplo: $\gcd(60, 24) = ?$

# gcd – opção 3

Algoritmo visto no ensino médio

- **Passo 1**: Encontre os fatores primos de $m$. Vá para o Passo 2.
- **Passo 2**: Encontre os fatores primos de $n$. Vá para o Passo 3.
- **Passo 3**: Identifique todos os fatores primos comuns nas expansões realizadas nos passos 1 e 2. Se $p$ é um fator primo comum ocorrendo $p_m$ vezes em $m$ e $p_n$ vezes em $n$, este deve ser repetido $\min(p_m, p_n)$ vezes. Vá para o Passo 4.
- **Passo 4**: Calcule o produto de todos os fatores primos comuns selecionados no Passo 3. O valor resultante é o maior divisor comum de $m$ e $n$.

Exemplo: $\gcd(60, 24) = ?$

O que acontece se $m$ ou $n$ for 1?
Como encontrar fatores primos?
Como encontrar elementos comuns em listas?

# gcd – opção 3 | Fatores primos = crivo de Eratóstenes

**Algoritmo 2:** sieve(n)

**input** : inteiro positivo maior do que 1
**output** : array $L$ de todos os números primos menores ou iguais a $n$

```
1  for p ← 2 to n do A[p] ← p;
2  for p ← 2 to ⌊√n⌋ do
3      if A[p] ≠ 0 then
4          j ← p ∗ p;
5          while j ≤ n do
6              A[j] ← 0;
7              j ← j + p;
8  i ← 0;
9  for p ← 2 to n do
10     if A[p] ≠ 0 then
11         L[i] ← A[p];
12         i ← i + 1;
```

# gcd

Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
   - Como encontrar fatores primos e elementos comuns em listas?

# gcd

Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
   - Como encontrar fatores primos e elementos comuns em listas?
2. É preciso especificar as entradas legítimas
   - Entrada zero (opção 2) ou um (opção 3)

# gcd

Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
   - Como encontrar fatores primos e elementos comuns em listas?
2. É preciso especificar as entradas legítimas
   - Entrada zero (opção 2) ou um (opção 3)
3. Um algoritmo pode ser representado de diferentes maneiras
   - Por exemplo: especificação textual *vs.* pseudocódigo

# gcd

Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
   - Como encontrar fatores primos e elementos comuns em listas?
2. É preciso especificar as entradas legítimas
   - Entrada zero (opção 2) ou um (opção 3)
3. Um algoritmo pode ser representado de diferentes maneiras
   - Por exemplo: especificação textual *vs.* pseudocódigo
4. Podem existir vários algoritmos para um mesmo problema
   - Opções 1, 2 e 3

# gcd

Aspectos importantes

1. O caráter não-ambíguo não pode ser comprometido
   - Como encontrar fatores primos e elementos comuns em listas?
2. É preciso especificar as entradas legítimas
   - Entrada zero (opção 2) ou um (opção 3)
3. Um algoritmo pode ser representado de diferentes maneiras
   - Por exemplo: especificação textual *vs.* pseudocódigo
4. Podem existir vários algoritmos para um mesmo problema
   - Opções 1, 2 e 3
5. Analisar os prós/contras dos algoritmos para um problema
   - Opção 1: simples e rápido
   - Opção 2: mais lento, menos entradas legítimas
   - Opção 3: mais complexo e lento, menos entradas legítimas

# Agenda

# IF672: visão geral

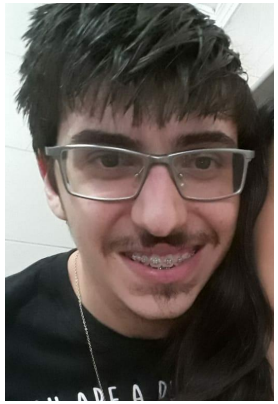
- Informações importantes
  - Professor: Gustavo Carvalho
    - E-mail: **ghpc@cin.ufpe.br**
    - Sala C126
  - **Site**: https://sites.google.com/a/cin.ufpe.br/if672/
    - Logado @cin.ufpe.br
  - **Lista**: https://groups.google.com/d/forum/if672_ghpc
    - Email: if672_ghpc@googlegroups.com
    - Importante: solicitar inclusão na lista!
- Plano de ensino
- Plano de aulas
- Plano pedagógico

# IF672: visão geral

- Informações importantes
  - Professor: Gustavo Carvalho
    - E-mail: **ghpc@cin.ufpe.br**
    - Sala C126
  - **Site**: https://sites.google.com/a/cin.ufpe.br/if672/
    - Logado @cin.ufpe.br
  - **Lista**: https://groups.google.com/d/forum/if672_ghpc
    - Email: if672_ghpc@googlegroups.com
    - Importante: solicitar inclusão na lista!
- Plano de ensino
- Plano de aulas
- Plano pedagógico | importante: **cópias**!

# IF672: monitores

## Monitores



Gabriel Pessoa

gap@cin.ufpe.br

José Cruz

jvsc@cin.ufpe.br

Pedro Coutinho

pnc@cin.ufpe.br

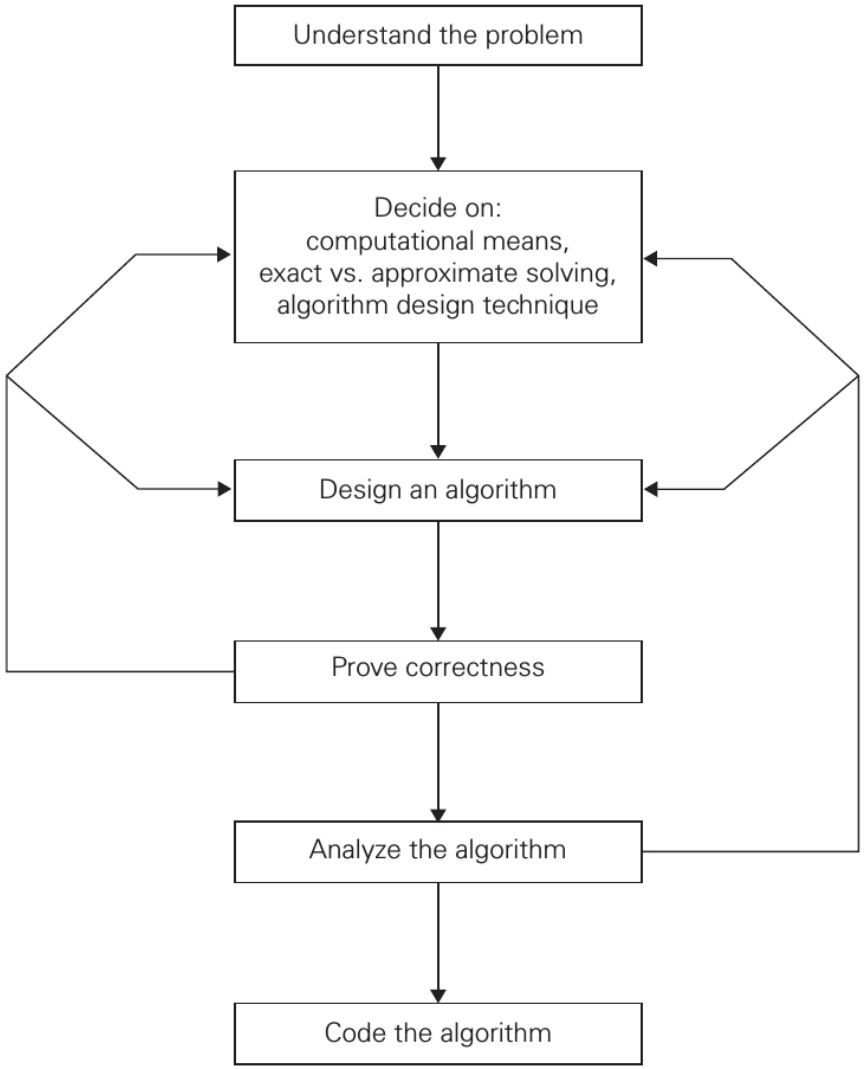
Rodrigo Lemos

rfrl@cin.ufpe.br

Aulas de monitoria: **sextas-feiras (12h – 13h)**

# Agenda

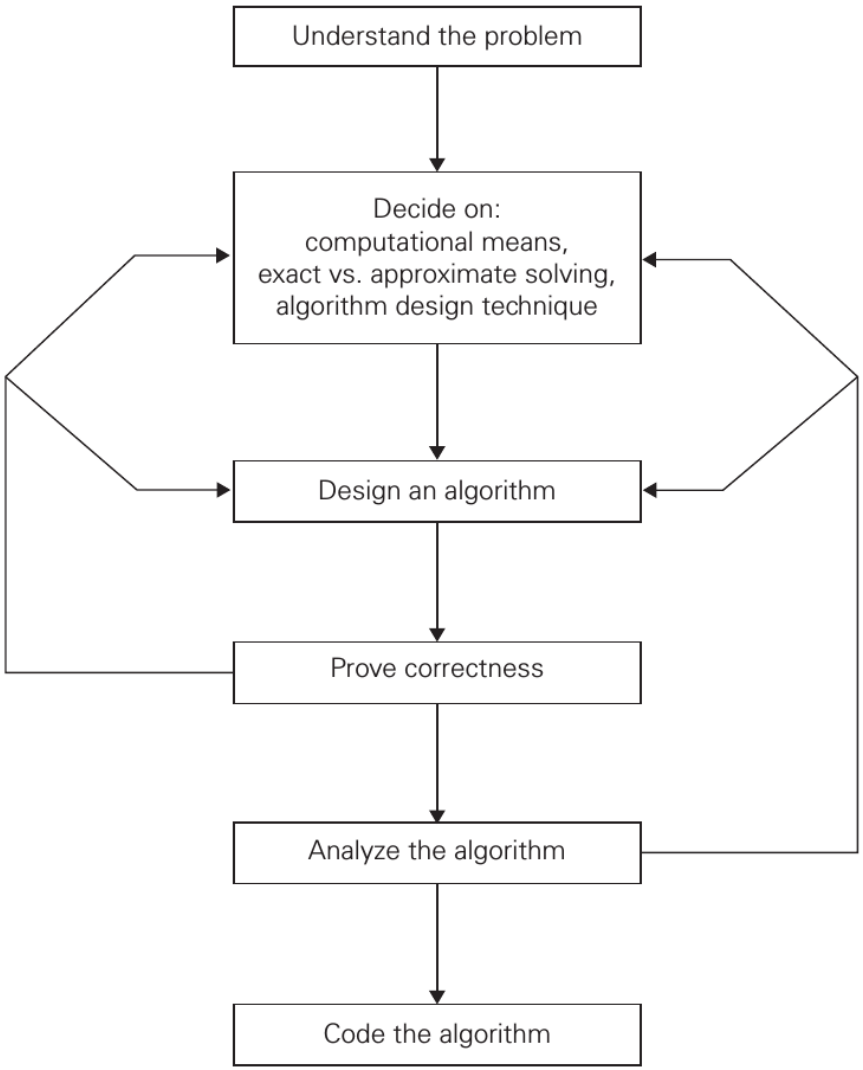
# Projetando e analisando algoritmos




4

[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

# Projetando e analisando algoritmos (1)

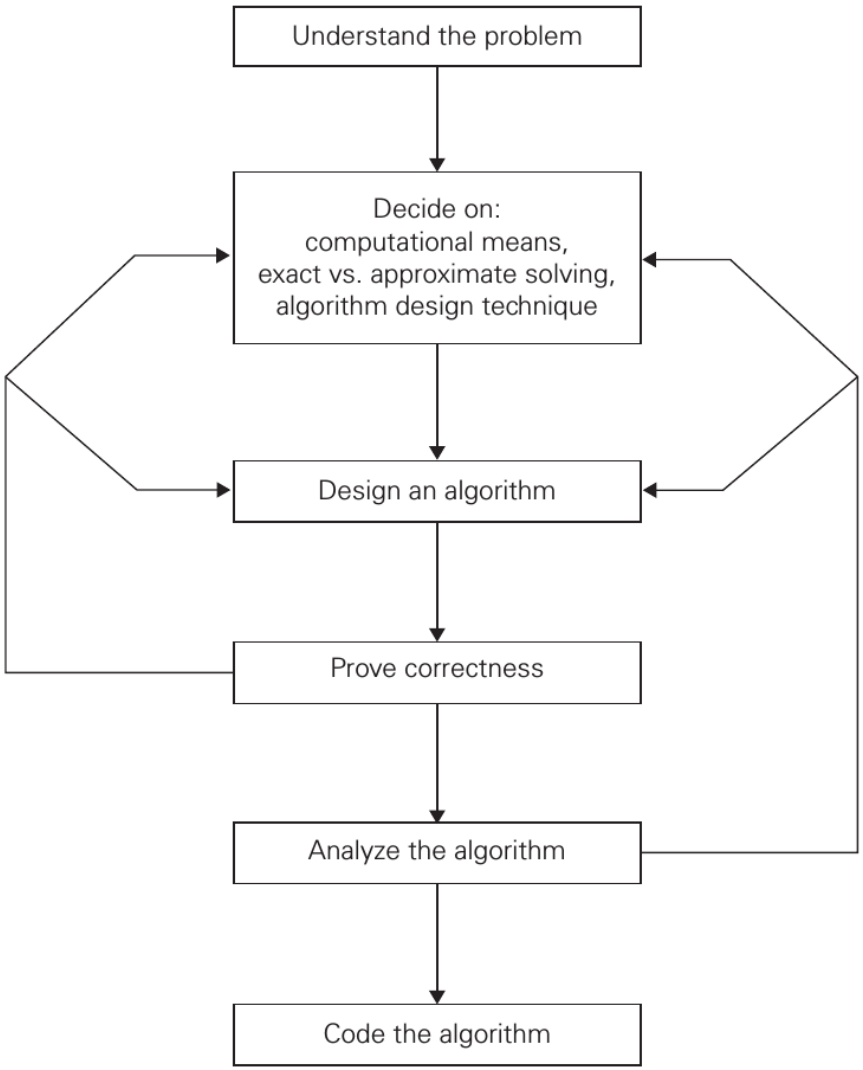



Entender o problema

- Fazer alguns exemplos na mão
- Pensar sobre casos especiais

Entrada para um algoritmo: instância

# Projetando e analisando algoritmos (2)




Avaliar capacidades computacionais

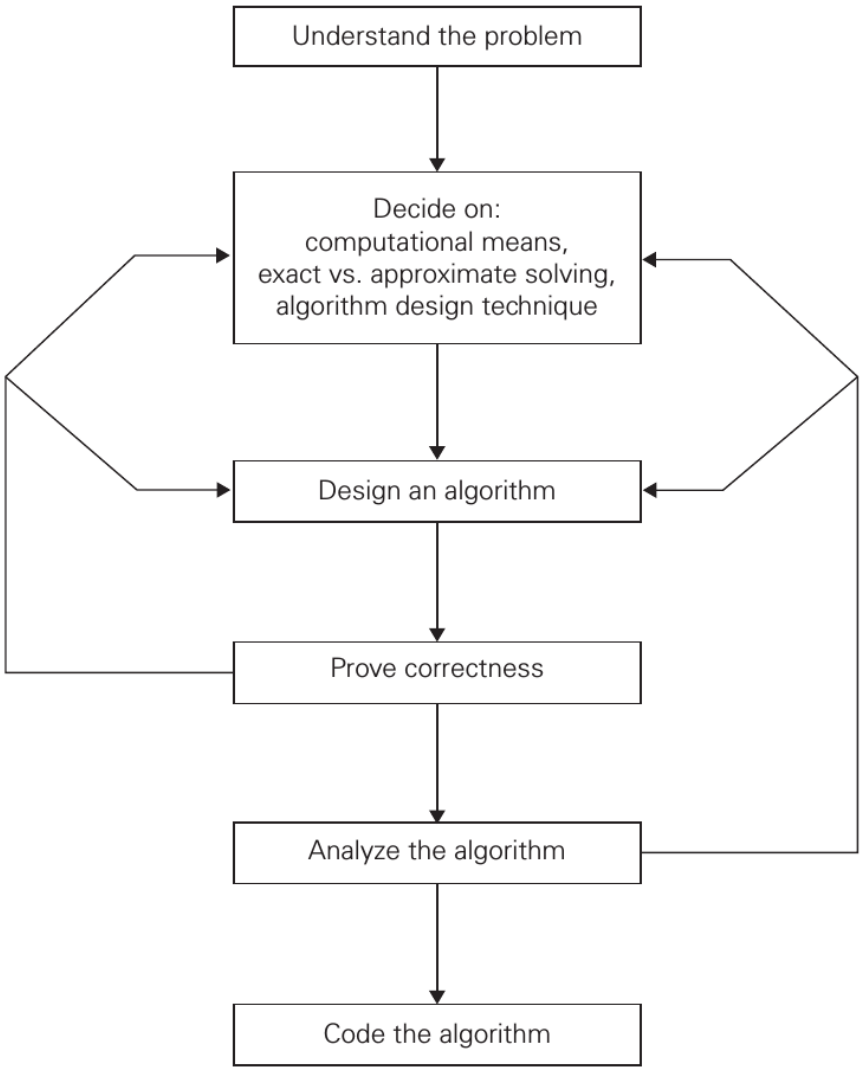
- Memória
- Tempo

Algoritmos sequênciais *vs.* paralelos

Algoritmos exatos *vs.* aproximação

- Ausência de solução universal
- Problemas complexos

# Projetando e analisando algoritmos (3)



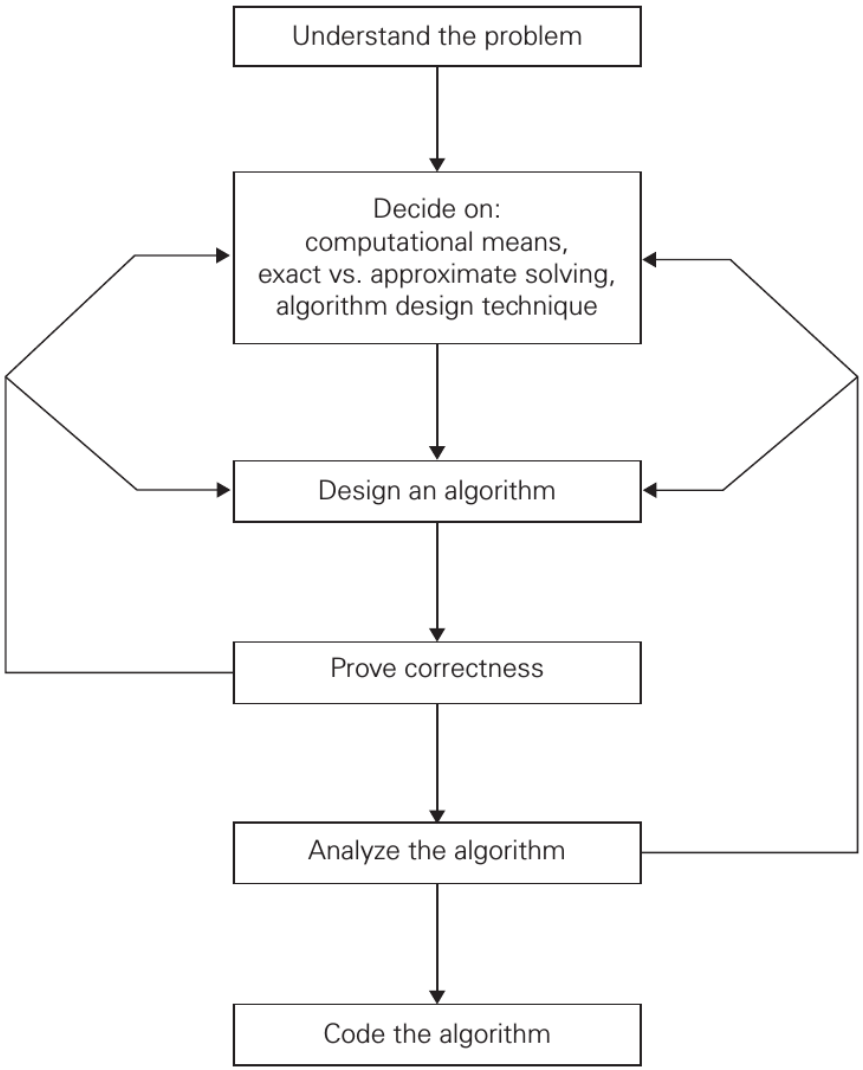

Projetar o algoritmo

- Estrutura de dados apropriada

Como especificar o algoritmo

- Linguagem natural
- Pseudocódigo
- Fluxogramas

# Projetando e analisando algoritmos (4)




Provar corretude

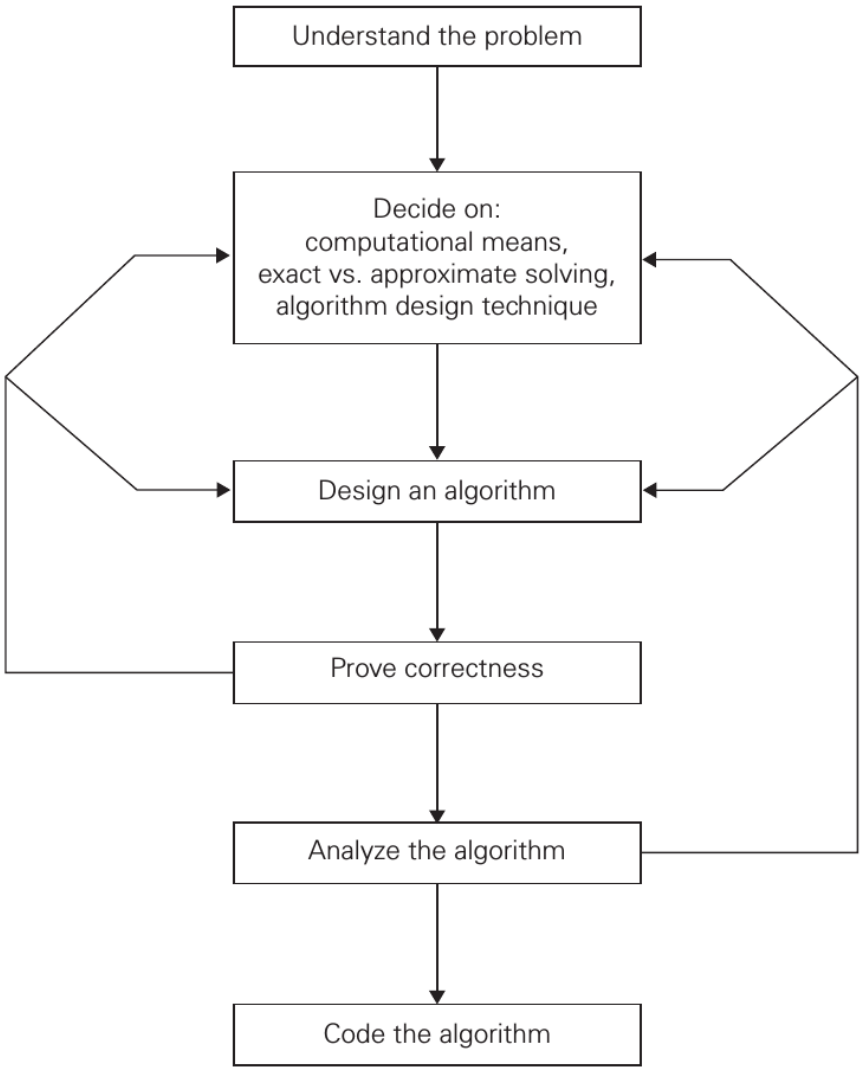
- Retorna uma solução para toda entrada legítima
- Especificação *vs.* implementação
- Técnica comum: indução

`Definition` is_a_sorting_algorithm
(*f*: **list nat** $\rightarrow$ **list nat**) :=
$\forall$ *al*, **Permutation** *al* (*f al*)
$\wedge$ **sorted** (*f al*).

E algoritmos de aproximação?

- Erro produzido menor do que limite estipulado

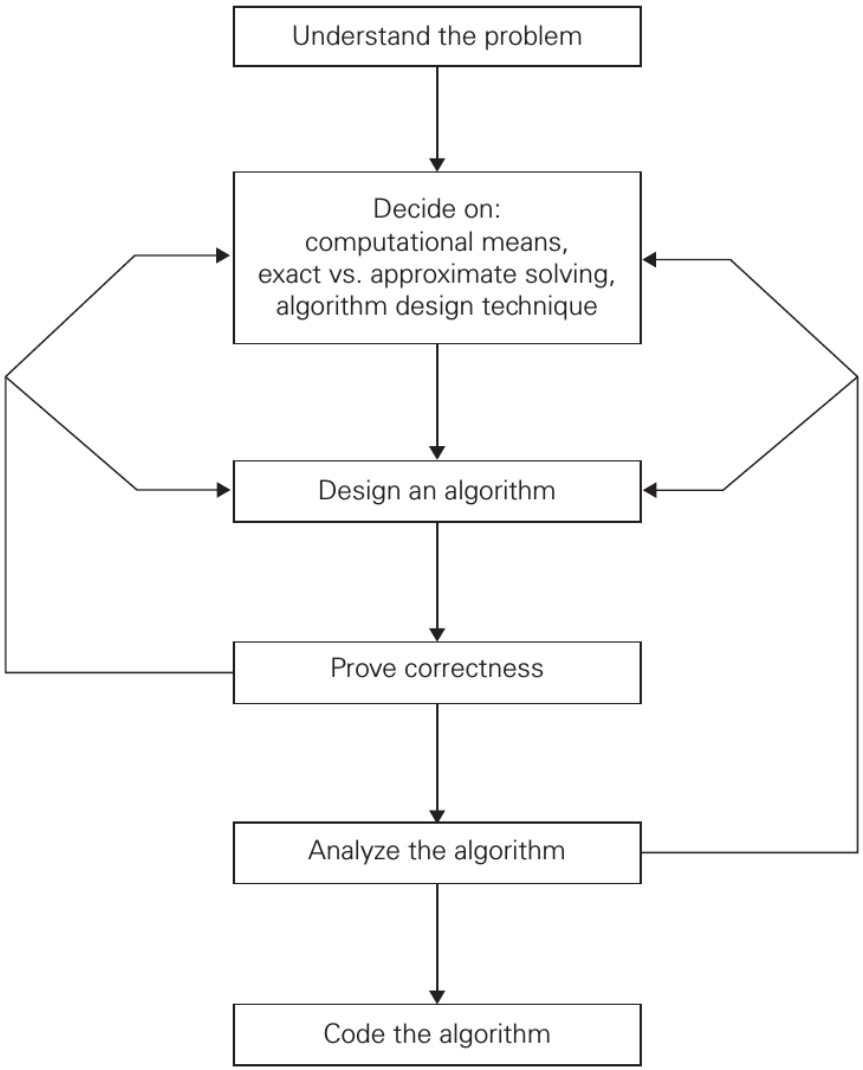
# Projetando e analisando algoritmos (5)




Analisar o algoritmo

- Eficiência (memória, tempo)
- Simplicidade
- Generalidade

# Projetando e analisando algoritmos (6)




Implementar o algoritmo

- Verificação: na prática, testes

Verificar suposições sobre entradas

# Agenda

# Principais estratégias para algoritmos

## Força bruta

- Construir a solução analisando todas as possibilidades

## Decrescer e conquistar

- Construir a solução a partir de instâncias menores (por uma constante)

## Dividir e conquistar

- Construir a solução a partir de instâncias menores

## Transformar e conquistar

- Transformar o problema antes de resolvê-lo

## Equilíbrio entre espaço e tempo

- Priorizar o tempo em detrimento do espaço

# Principais estratégias para algoritmos

## Programação dinâmica

- Construir a solução a partir de soluções intermediárias (previamente calculadas)

## Abordagens gulosas

- Construir a solução expandindo uma solução prévia

## Backtracking

- Retroceder quando um caminho não se mostrar promissor na busca da solução

## Branch-and-bound

- Parecido com backtracking, mas para problemas de otimização

## Abordagens de aproximação

- Quando não é possível ou viável achar uma solução exata

# Agenda

# Problemas importantes

Problemas de ordenação

- Ordenar não só números, mas registros (*key*)
- Estável | *in-place*

Problemas de busca

- Sequencial *vs.* binária
- Custo de adição e remoção de dados

Processamento de strings

- Strings de texto, strings binárias, sequências genéticas
- *String matching*

Problemas de grafos

- Busca em grafos
- Algoritmos de menor caminho
- Ordenação topológica

# Problemas importantes

Problemas combinatoriais

- Encontrar uma permutação, uma combinação ou um conjunto que satisfaz certas restrições

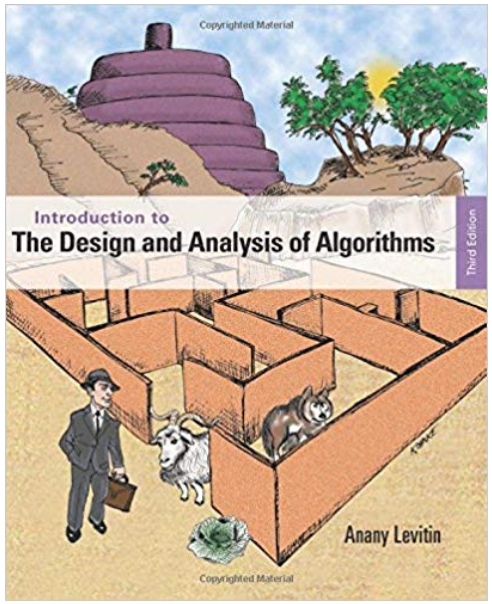
Problemas geométricos

- Problema *closest-pair*
- *Convex hull*

Problemas numéricos

- Tipicamente objetos matemáticos contínuos
- Muitos problemas somente com soluções aproximadas

# Bibliografia + leitura recomendada




**Capítulo 1 (pp. 1–25). Anany Levitin.**
*Introduction to the Design and Analysis of Algorithms*.
3a edição. Pearson. 2011.

# APRESENTAÇÃO DA DISCIPLINA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil