TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Apresentação

Instituto Superior Técnico

2022/2023

Corpo Docente

Aulas Teóricas

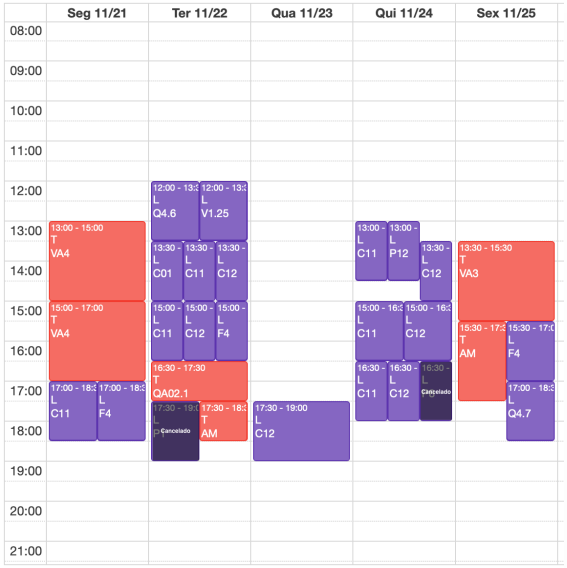
- Arlindo Oliveira
- Luís Guerra e Silva (responsável)

Aulas Práticas

- Jan Cederquist
- Luís Guerra e Silva
- Luís Russo
- Francisco Lisboa
- Eduardo Claudino

Os horários de dúvidas dos docentes estão disponíveis na página da UC em **Horários de Dúvidas**

Horário



Programa

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Algoritmos em Grafos [CLRS, Cap. 21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional

Método de Avaliação

Componente Teórica

- 6 avaliações MAP15 - **30%**
- 1 exame (1h) - **40%**

Componente Projetos

- 2 projetos de programação - **30%**

Mais detalhes, incluindo informações sobre época especial e trabalhadores-estudantes, encontram-se na página da UC em **Métodos de Avaliação**.

Método de Avaliação

Aulas Práticas / Avaliação MAP15

- Cada turno prático tem 2 aulas práticas / semana
- Avaliação MAP15:
  - 6 avaliações
  - Duração: 15 minutos
  - Ficha a resolver no início da 1ª aula prática de cada turno
  - Na própria folha do enunciado
  - Sobre os exercícios das aulas práticas da semana anterior
- 7 semanas de aulas:
  - Novembro: 1 semana (1 avaliação) - nesta primeira semana não há
  - Dezembro: 3 semanas (3 avaliações)
  - Janeiro: 2 semanas (2 avaliações)

Método de Avaliação

Projetos

- Cada projeto compreende a entrega do código e de um relatório
- A realizar por grupos de 1 ou 2 alunos
- Não têm nota mínima

1º Projeto

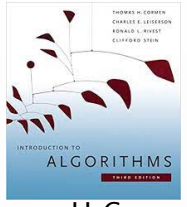
- Enunciado: 25 de novembro de 2022
- Entrega: 13 de dezembro de 2022 (18:00)

2º Projeto

- Enunciado: 16 de dezembro de 2022
- Entrega: 6 de janeiro de 2023 (18:00)

Mais detalhes encontram-se na página da UC em **Projetos**.

Bibliografia



Livro Principal

- **Introduction to Algorithms**, 3rd Edition, Thomas H Cormen, Charles E Leiserson, Ronald L Rivest e Clifford Stein, 2009, MIT Press

Outro Material

- **Algorithms**, S. Dasgupta, C. Papadimitriou e U. Vazirani, 2006, McGraw-Hill
- Slides + apontamentos das aulas
- Colectânea de exercícios

Objectivos

Objectivos

- Estudo e análise de Algoritmos
  - Complementar *Introdução aos Algoritmos e Estruturas de Dados*
  - Técnicas para análise e síntese de algoritmos
- Introdução à Complexidade Computacional

Não são objectivos

- Ensinar linguagens de programação
- Optimização de código

TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Notação assimptótica. Recorrências. Teorema Mestre.

CLRS Cap.1-4

Instituto Superior Técnico

2022/2023

Programa

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Algoritmos em Grafos [CLRS, Cap. 21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional

Algoritmo

Algoritmo

- Procedimento computacional bem definido que
  - aceita uma dada entrada
  - produz uma dada saída

Ferramenta para resolver um problema computacional bem definido:

- Ordenação de sequências de valores
- Caminhos mais curtos em grafos dirigidos
- etc.

Exemplo: Ordenação

Entrada: sequência de valores $A[1..n]$
Objectivo: ordenar valores em A de forma crescente
Saída: sequência de valores ordenados $A[1..n]$

InsertionSort(A)

```
for j = 2 to length[A] do
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key do
        A[i+1] = A[i]
        i = i - 1
    end while
    A[i+1] = key
end for
```

Exemplo: 7, 8, 5, 2, 4, 6, 3

Análise de Algoritmos

Q: Como aferir a complexidade de um algoritmo?
Q: Como comparar dois algoritmos diferentes?

- Notação assimptótica

Q: Que modelo computacional utilizar?

- Modelo RAM (Random Access Machine)
- Execução sequencial de instruções
- Apenas 1 processador
- Outros modelos computacionais relacionados polinomialmente com modelo RAM

Análise de Algoritmos

Medidas de Complexidade

- Tempo (execução) e espaço necessário

Notas

- Tanto o tempo como o espaço dependem do tamanho da entrada
- Entrada depende do problema que o algoritmo pretende resolver

Exemplos

- No InsertionSort uma medida razoável é o número de elementos a ordenar
- Num grafo as medidas utilizadas são o número de vértices/arestas

Tempo de Execução

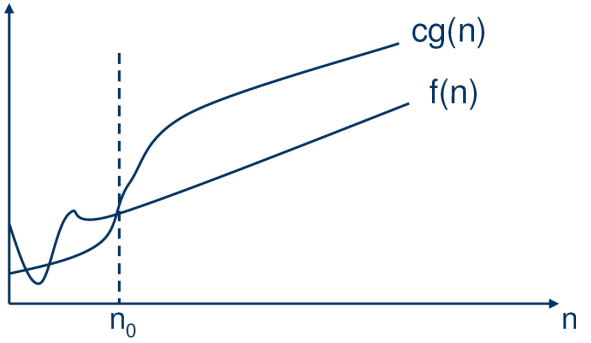
n	$f(n) = n$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = n!$
10	0.01 μs	0.10 μs	1.00 μs	3.63 ms
20	0.02 μs	0.40 μs	1.00 ms	77.1 anos
30	0.03 μs	0.90 μs	1.00 s	$8.4 * 10^{15}$ anos
40	0.04 μs	1.60 μs	18.3 min	
50	0.05 μs	2.50 μs	13 dias	
100	0.10 μs	10 μs	$4 * 10^{13}$ anos	
1000	1.00 μs	1 ms		

Objectivo
Caracterizar tempos de execução dos algoritmos para tamanhos arbitrários das entradas

- Permite estabelecer taxas de crescimento dos tempo de execução dos algoritmos em função dos tamanhos das entradas
- Constantes multiplicativas e aditivas tornam-se irrelevantes
  - Tempo de execução de cada instrução não é essencial para o comportamento assimptótico de um algoritmo

Notação O

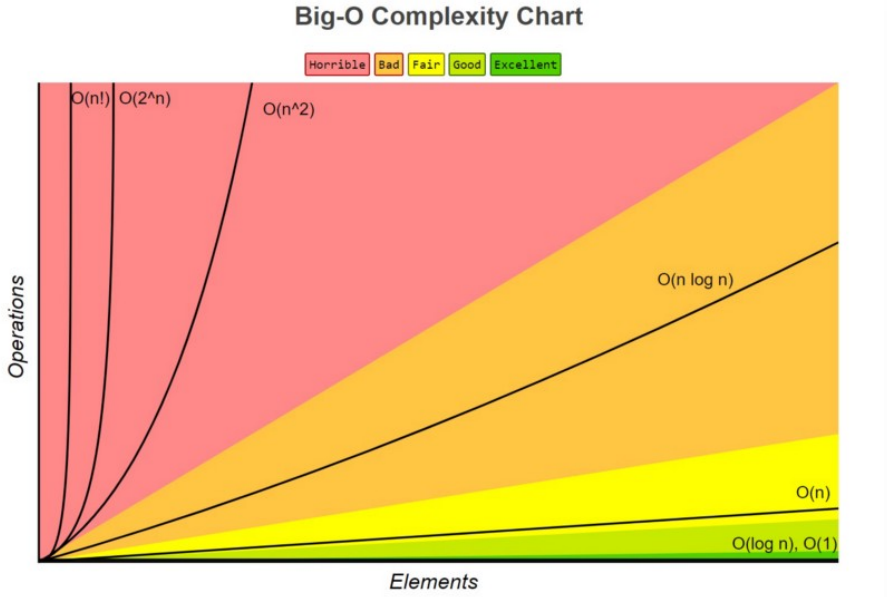
- Limite Assimptótico Superior
- $O(g(n)) = \{f(n)$: existem constantes positivas c e n_0, tal que $0 \leq f(n) \leq cg(n)$, para $n \geq n_0\}$
- $f(n) = O(g(n))$, significa $f(n) \in O(g(n))$



Notação O - (algumas) propriedades

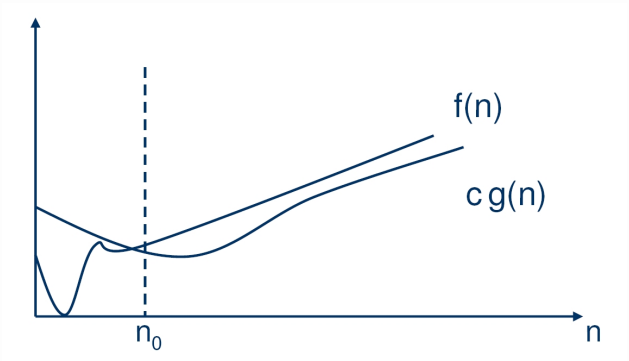
- Reflexividade: f é $O(f)$
- Constantes: Se f é $O(g)$ e $c > 0$, então cf é $O(g)$
- Somas: Se f_1 é $O(g_1)$ e f_2 é $O(g_2)$, então $f_1 + f_2$ é $O(max(g_1, g_2))$
- Transitividade. Se f é $O(g)$ e g é $O(h)$, então f é $O(h)$

Exemplo
$f(n) = 2n^4 + 7n^3 + 3n^2 + n + 8$ é $O(n^4)$



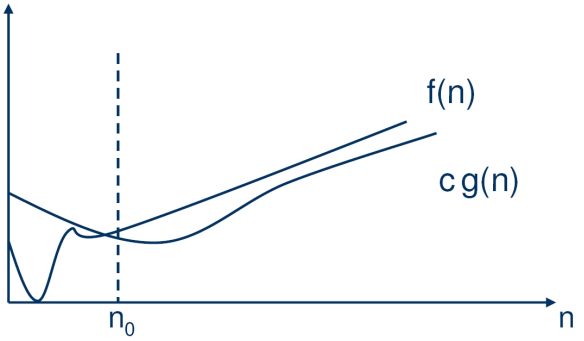
Notação Ω

- Limite Assimptótico Inferior
- $\Omega(g(n)) = \{f(n)$: existem constantes positivas c e n_0, tal que $0 \leq cg(n) \leq f(n)$, para $n \geq n_0\}$
- $f(n) = \Omega(g(n))$, significa $f(n) \in \Omega(g(n))$



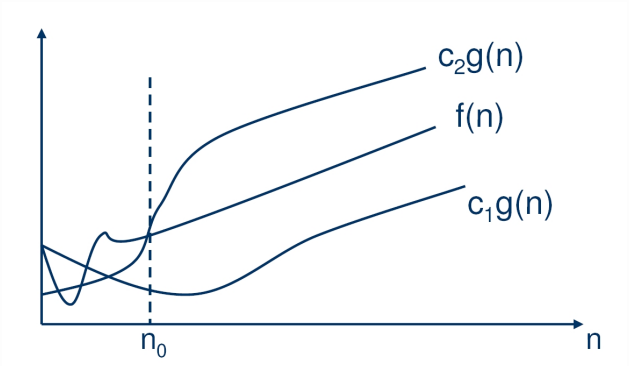
Exemplo

- $f(n) = 3n^2 + n + 8$
- $f(n)$ é simultaneamente $\Omega(n^2)$ e $\Omega(n)$
- $f(n)$ não é $\Omega(n^3)$



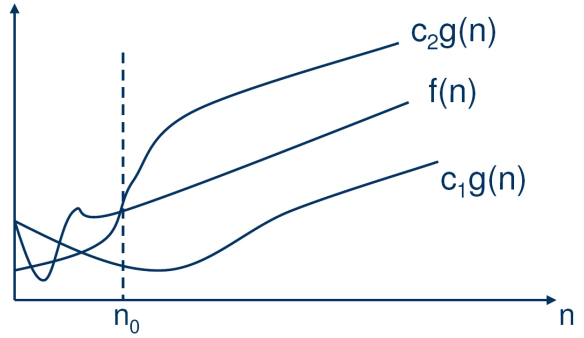
Notação Θ

- Limite Assimptótico Apertado
- $\Theta(g(n)) = \{f(n)$: existem constantes positivas c_1, c_2 e n_0, tal que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, para $n \geq n_0\}$
- $f(n) = \Theta(g(n))$, significa $f(n) \in \Theta(g(n))$



Exemplo

- $f(n) = 3n^2 + n + 8$
- $f(n)$ é $\Theta(n^2)$
- $f(n)$ não é $\Theta(n)$ nem $\Theta(n^3)$



Notação Adicional

Notação adicional

- Floor e Ceiling: $\lfloor \quad \rfloor \quad \lceil \quad \rceil$
- Polinónios
  - Um polinómio cresce com o maior grau que o compõe
- Exponenciais
  - Uma exponencial (base > 1) cresce mais depressa do que um polinómio
- Logaritmos

Recorrências

Recorrências

- Utilizadas para exprimir e calcular do tempo de execução de **procedimentos recursivos**

Resolução de Recorrências

- Por **Substituição**
  - Sugestão de solução
  - Provar formalmente a solução utilizando indução
- Por **Iteração**
  - Expandir recorrência com o intuito de a expressar em termos apenas de n e das condições iniciais
- Por **Teorema Mestre**

Recorrências

Recorrências subtract-and-conquer

$$T(n) \leq \begin{cases} c, & n \leq 1 \\ aT(n-b) + f(n), & n > 1 \end{cases}, a > 0, b > 0$$

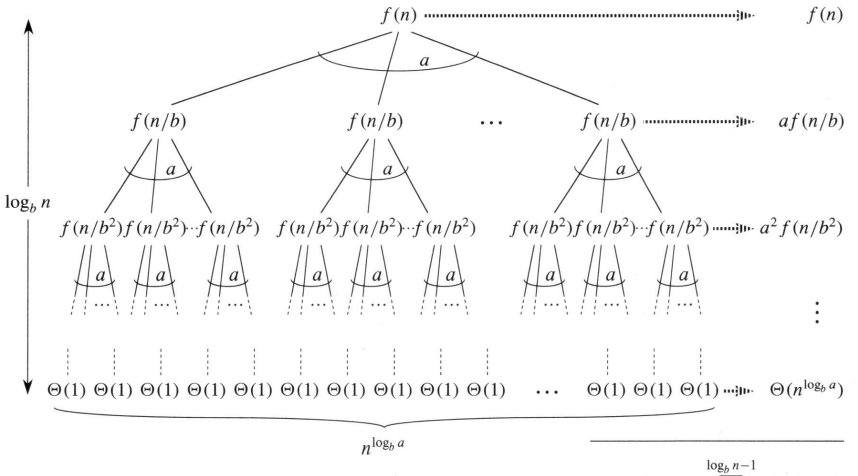
Exemplo - por iteração

$$\begin{aligned} T(n) &= T(n-2) + n^2 \\ &= T(n-4) + (n-2)^2 + n^2 \\ &= T(n-6) + (n-4)^2 + (n-2)^2 + n^2 \\ &= T(n-8) + (n-6)^2 + (n-4)^2 + (n-2)^2 + n^2 \\ &= \ldots \\ &= n^2 + (n-2)^2 + (n-4)^2 + \ldots + 2^2 + T(0) \\ &= T(0) + \textstyle\sum_{k=1}^{n/2} (2k)^2 \\ &= T(0) + 4\textstyle\sum_{k=1}^{n/2} k^2 \qquad \text{(série quadrados consecutivos)} \\ &= T(0) + 4\frac{\frac{n}{2}(\frac{n}{2}+1)(\frac{2n}{2}+1)}{6} \end{aligned}$$

$\Rightarrow T(n) \in \Theta(n^3)$

Recorrências

Recorrências divide-and-conquer



Teorema Mestre

Permite resolver recorrências da forma (divide-and-conquer)

$$T(n) = aT(n/b) + O(n^d)\ , \quad a \geq 1,\ b > 1,\ d \geq 0$$

Problema é dividido em a subproblemas, cada um com dimensão n/b

Teorema Mestre

Sejam $a \geq 1, b > 1, d \geq 0$ constantes, e seja $T(n)$ definido por $T(n) = aT(n/b) + O(n^d)$.

$T(n)$ é limitado assimptoticamente da seguinte forma:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^d) & \text{if } d > \log_b a \end{cases}$$

- Em casa um dos 3 casos estamos a comparar d com $\log_b a$
- Solução da recorrência é determinada pela maior das duas funções

Ver definição alternativa no livro do Cormen!

$T(n)$ é limitado assimptoticamente da seguinte forma:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^d) & \text{if } d > \log_b a \end{cases}$$

Exemplos

$T(n) = 9T(n/3) + n$

- $a = 9, b = 3, d = 1$
- $d = 1$ é $<$ que $\log_3 9$
- $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$ (caso 1 do Teorema Mestre)

$T(n)$ é limitado assimptoticamente da seguinte forma:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } d < \log_b a \\ \Theta(n^d \log n) & \text{if } d = \log_b a \\ \Theta(n^d) & \text{if } d > \log_b a \end{cases}$$

Exemplos

$T(n) = T(2n/3) + 1$

- $a = 1, b = \frac{3}{2}, d = 0$
- $d = 0$ é $=$ a $\log_{\frac{3}{2}} 1$
- $T(n) = \Theta(n^d \log n) = \Theta(\log n)$ (caso 2 do Teorema Mestre)

$T(n)$ é limitado assimptoticamente da seguinte forma:

$$T(n) = \begin{cases} \Theta(n^{log_b\, a}) & if\ d < log_b\, a \\ \Theta(n^d log\ n) & if\ d = log_b\, a \\ \Theta(n^d) & if\ d > log_b\, a \end{cases}$$

Exemplos

$T(n) = 3T(n/3) + n^2$

- $a = 3, b = 3, d = 2$
- $d = 2$ é $>$ que $log_3 3$
- $T(n) = \Theta(n^d) = \Theta(n^2)$ (caso 3 do Teorema Mestre)