

MCTA003-17 - Análise de Algoritmos

Apresentação

Maycon Sambinelli

m.sambinelli@ufabc.edu.br

<https://professor.ufabc.edu.br/~m.sambinelli/>

2023.Q2

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC



- 1 Análise de Algoritmos
- 2 Algoritmo para a Sequência de Fibonacci
- 3 "Algoritmo" para o problema da 3 coloração de grafos
- 4 Sobre o Curso
- 5 Exercícios de Revisão

Análise de Algoritmos

Definição

Um *algoritmo* é uma sequência finita de passos descritos de forma não ambígua que corretamente resolve um problema.

Um algoritmo:

- recebe um conjunto de dados **válidos** como entrada e devolve um conjunto de dados como saída.
- Para toda entrada possível ele produz uma saída que seja solução do problema para aquela entrada.

Permite:

1. Verificar a correção de um algoritmo;
2. Prever o desempenho do algoritmo
 - 2.1 Comparar algoritmos sem a necessidade de implementá-los e rodar testes de benchmark
3. Mostrar que um problema é tão difícil quanto um outro problema (reduções)

1. Algoritmo para a sequência de Fibonacci
 - Verificar a Correção
 - Prever o desempenho
 - Comparar algoritmos
2. "Algoritmo" para o problema da 3 coloração de grafos
 - Mostrar que um problema é tão difícil quanto outro

Algoritmo para a Sequência de Fibonacci

A *sequência de Fibonacci* é a sequência infinita de números:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

A *sequência de Fibonacci* é a sequência infinita de números:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Definição

A *sequência de Fibonacci* é a sequência de números F_1, F_2, F_3, \dots , onde

$$F_n = \begin{cases} 1 & \text{se } n = 1; \\ 1 & \text{se } n = 2; \\ F_{n-1} + F_{n-2} & \text{se } n > 2. \end{cases}$$

A *sequência de Fibonacci* é a sequência infinita de números:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Definição

A *sequência de Fibonacci* é a sequência de números F_1, F_2, F_3, \dots , onde

$$F_n = \begin{cases} 1 & \text{se } n = 1; \\ 1 & \text{se } n = 2; \\ F_{n-1} + F_{n-2} & \text{se } n > 2. \end{cases}$$

Problema: NÚMERO DE FIBONACCI

Entrada: um inteiro $n \geq 1$

Saída: F_n

- 1: **Função** $\text{FIB1}(n)$
- 2: **Se** $n \leq 2$ **então**
- 3: **Devolve** 1
- 4: **Devolve** $\text{FIB1}(n - 1) + \text{FIB1}(n - 2)$

- 1: **Função** $\text{FIB1}(n)$
- 2: **Se** $n \leq 2$ **então**
- 3: **Devolve** 1
- 4: **Devolve** $\text{FIB1}(n - 1) + \text{FIB1}(n - 2)$

Perguntas:

1. Esse algoritmo resolve o problema?

- 1: **Função** $\text{FIB1}(n)$
- 2: **Se** $n \leq 2$ **então**
- 3: **Devolve** 1
- 4: **Devolve** $\text{FIB1}(n - 1) + \text{FIB1}(n - 2)$

Perguntas:

1. Esse algoritmo resolve o problema?
2. Quanto tempo ele leva?

- 1: **Função** $\text{FIB1}(n)$
- 2: **Se** $n \leq 2$ **então**
- 3: **Devolve** 1
- 4: **Devolve** $\text{FIB1}(n - 1) + \text{FIB1}(n - 2)$

número de instruções do Algoritmo $1 \leq 2^{n+1} - n - 2$

```
1 fib:
2     push    rbp
3     mov     rbp, rsp
4     push    rbx
5     sub     rsp, 24
6     mov     DWORD PTR [rbp-20], edi
7     cmp     DWORD PTR [rbp-20], 2
8     jg     .L2
9     mov     eax, 1
10    jmp    .L3
11 .L2:
12    mov     eax, DWORD PTR [rbp-20]
13    sub     eax, 1
14    mov     edi, eax
15    call   fib
16    mov     ebx, eax
17    mov     eax, DWORD PTR [rbp-20]
18    sub     eax, 2
19    mov     edi, eax
20    call   fib
21    add     eax, ebx
22 .L3:
23    mov     rbx, QWORD PTR [rbp-8]
24    leave
25    ret
```

O valor de F_{3000} tem 627 dígitos, um valor muito alto para ser representado em um inteiro ou double.

$F_{3000} = 410615886307971260333568378719267105220125108637369252408885430926905584$
2741134037313304916608500445608300368357069422745885693621454765026743730454468
5216048660629249736050346977345373319688740584725529008204908690751262205905454
2195889758031109222670849274793859539133318371244795543147611073276240066737934
0851917318109932017067768389347667647787395021744702686278209185538422258583064
0830166186290035826685723821023580250435195147299791967652400478423637645334726
8364152648346245840573214241419937917242918602639810097866942392015404620153818
6714257398350748513964211399827136406795811784581986586922859680432436567097960
00

- Nossos números precisam ser representados em uma estrutura de **bigint/arrayNumber**.
 - Esse fato foi levado em conta durante a estimativa do número de instruções.

Nome	AMD Ryzen 9 5900X
Fabricante	AMD
Cores	12
Threads	24
Clock	até 4,8 GHz
Preço do processador	R\$ 2600,00
Instruções por Segundo	125816×10^6

Supercomputador mais potente em operação atualmente (Jun 2022)¹

Nome	Frontier
Fabricante	HPE
Núcleos	8.730.112 (CPU: 591.872, GPU: 8.138.240)
Processador	AMD Optimized 3rd Generation EPYC 64C 2GHz
GPU	AMD Instinct MI250X
Sistema Operacional	Linux (HPE Clay OS)
Localização	Oak Ridge National Laboratory - EUA
Propósito	Pesquisa Científica
Custo	US\$600 milhões
Instruções por Segundo	10^{18}

¹<https://www.top500.org/lists/top500/2022/06/>

	$n = 30$	$n = 60$	$n = 80$	$n = 100$
Desktop	17s	≈ 212 dias	≈ 609 milênios	≈ 638979729 milênios
Frontier	2.19×10^{-6}	$\approx 2s$	≈ 27 dias	≈ 80 milênios

	$n = 30$	$n = 60$	$n = 80$	$n = 100$
Desktop	17s	≈ 212 dias	≈ 609 milênios	≈ 638979729 milênios
Frontier	2.19×10^{-6}	$\approx 2s$	≈ 27 dias	≈ 80 milênios

- E agora, José?!

```
1: Função FIB2( $n$ )  
2:    $penultimo = 1$   
3:    $ultimo = 1$   
4:   Para  $i = 3$  até  $n$  faça  
5:      $novo = penultimo + ultimo$   
6:      $penultimo = ultimo$   
7:      $ultimo = novo$   
8:   Devolve  $novo$ 
```

```
1: Função FIB2( $n$ )
2:    $penultimo = 1$ 
3:    $ultimo = 1$ 
4:   Para  $i = 3$  até  $n$  faça
5:      $novo = penultimo + ultimo$ 
6:      $penultimo = ultimo$ 
7:      $ultimo = novo$ 
8:   Devolve  $novo$ 
```

Perguntas:

1. Esse algoritmo resolve o problema?

```
1: Função FIB2( $n$ )
2:    $penultimo = 1$ 
3:    $ultimo = 1$ 
4:   Para  $i = 3$  até  $n$  faça
5:      $novo = penultimo + ultimo$ 
6:      $penultimo = ultimo$ 
7:      $ultimo = novo$ 
8:   Devolve  $novo$ 
```

Perguntas:

1. Esse algoritmo resolve o problema?
2. Quanto tempo ele leva?

número de instruções do Algoritmo $2 \leq n^2 + 5n + 3$

	$n \leq 10^5$	$n = 10^8$	$n = 10^{10}$
Desktop	$< 0,08s$	≈ 22 horas	≈ 25 anos
Frontier	$< 10^{-0,8}s$	$\approx 0,01s$	$\approx 3,17$ séculos

número de instruções do Algoritmo $2 \leq n^2 + 5n + 3$

	$n \leq 10^5$	$n = 10^8$	$n = 10^{10}$
Desktop	$< 0,08s$	≈ 22 horas	≈ 25 anos
Frontier	$< 10^{-0,8}s$	$\approx 0,01s$	$\approx 3,17$ séculos

Podemos fazer melhor!

1: **Função** $\text{FIB3}(n)$

2: **Devolve** $\left[\frac{\left(\frac{1 + \sqrt{5}}{2} \right)^n}{\sqrt{5}} \right]$

Perguntas:

1. Esse algoritmo resolve o problema?

1: **Função** $\text{FIB3}(n)$

2: **Devolve** $\left\lfloor \frac{\left(\frac{1 + \sqrt{5}}{2}\right)^n}{\sqrt{5}} \right\rfloor$

Perguntas:

1. Esse algoritmo resolve o problema?
2. Quanto tempo ele leva?

número de instruções do Algoritmo 3 $\leq 3 + \ln(n) + n^{1,5} + n$

	$n \leq 10^5$	$n = 10^8$	$n = 10^{10}$
Desktop	$< 0,00025s$	$\approx 8s$	$\approx 2,2$ horas
Frontier	$< 3 \times 10^{-11}s$	$\approx 1 \times 10^{-6}s$	$\approx 0,001s$

Qual algoritmo implementar?

- Algoritmo 1
- Algoritmo 2
- Algoritmo 3

- Não falamos de número de instrução, falamos em passos *simples* em uma máquina abstrata.
- Seja $T_i(n)$ o tempo de execução (número de passos) do Algoritmo i , para $i = 1, 2, 3$, para uma entrada de tamanho n .

$$T_1(n) = 2^{n+1} - n - 2 = \mathcal{O}(2^n)$$

$$T_2(n) = n^2 + 5n + 3 = \mathcal{O}(n^2)$$

$$T_3(n) = n^{1,5} + n + \lg(n) + 3 = \mathcal{O}(n^{1,5})$$

- Não falamos de número de instrução, falamos em passos *simples* em uma máquina abstrata.
- Seja $T_i(n)$ o tempo de execução (número de passos) do Algoritmo i , para $i = 1, 2, 3$, para uma entrada de tamanho n .

$$T_1(n) = 2^{n+1} - n - 2 = \mathcal{O}(2^n)$$

$$T_2(n) = n^2 + 5n + 3 = \mathcal{O}(n^2)$$

$$T_3(n) = n^{1,5} + n + \lg(n) + 3 = \mathcal{O}(n^{1,5})$$

$$n^{1,5} \ll n^2 \ll 2^n$$

"Algoritmo" para o problema da 3 coloração de grafos

Literal	uma variável booleana ou sua negação	x_i ou \bar{x}_i
Clausula	a disjunção de 3 literais distintos	$C_j = (x_1 \vee \bar{x}_2 \vee x_3)$
Forma Normal Conjuntiva	uma fórmula Φ que é a conjunção de clausulas	$C_1 \wedge C_2 \wedge C_3 \wedge C_4$

Literal	uma variável booleana ou sua negação	x_i ou \bar{x}_i
Clausula	a disjunção de 3 literais distintos	$C_j = (x_1 \vee \bar{x}_2 \vee x_3)$
Forma Normal Conjuntiva	uma fórmula Φ que é a conjunção de clausulas	$C_1 \wedge C_2 \wedge C_3 \wedge C_4$

Problema: 3-SATISFATIBILIDADE BOOLIANA (3-SAT)

Entrada: uma fórmula Φ consistindo de k clausulas sobre n variáveis

Saída: **Verdadeiro**, se existe uma atribuição de valores as variáveis de Φ tal que o valor de Φ seja verdadeiro; **Falso**, caso contrário.

Literal	uma variável booleana ou sua negação	x_i ou \bar{x}_i
Clausula	a disjunção de 3 literais distintos	$C_j = (x_1 \vee \bar{x}_2 \vee x_3)$
Forma Normal Conjuntiva	uma fórmula Φ que é a conjunção de clausulas	$C_1 \wedge C_2 \wedge C_3 \wedge C_4$

Problema: 3-SATISFATIBILIDADE BOOLIANA (3-SAT)

Entrada: uma fórmula Φ consistindo de k clausulas sobre n variáveis

Saída: **Verdadeiro**, se existe uma atribuição de valores as variáveis de Φ tal que o valor de Φ seja verdadeiro; **Falso**, caso contrário.

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$$

Literal	uma variável booleana ou sua negação	x_i ou \bar{x}_i
Clausula	a disjunção de 3 literais distintos	$C_j = (x_1 \vee \bar{x}_2 \vee x_3)$
Forma Normal Conjuntiva	uma fórmula Φ que é a conjunção de clausulas	$C_1 \wedge C_2 \wedge C_3 \wedge C_4$

Problema: 3-SATISFATIBILIDADE BOOLIANA (3-SAT)

Entrada: uma fórmula Φ consistindo de k clausulas sobre n variáveis

Saída: **Verdadeiro**, se existe uma atribuição de valores as variáveis de Φ tal que o valor de Φ seja verdadeiro; **Falso**, caso contrário.

$$\Phi = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$$

Solução:

x_1	x_2	x_3	x_4
V	V	F	T

Problema: 3-COLORAÇÃO DE GRAFOS (3-COLORAÇÃO)

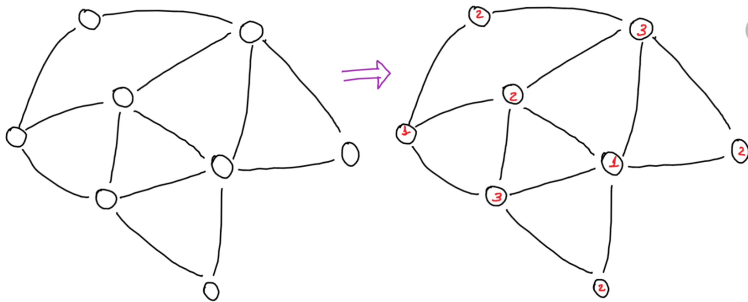
Entrada: uma grafo G

Saída: **Verdadeiro**, se existe uma coloração $c: V(G) \rightarrow \{1, 2, 3\}$ tal que se $uv \in E(G)$, então $c(u) \neq c(v)$; e **Falso**, caso contrário.

Problema: 3-COLORAÇÃO DE GRAFOS (3-COLORAÇÃO)

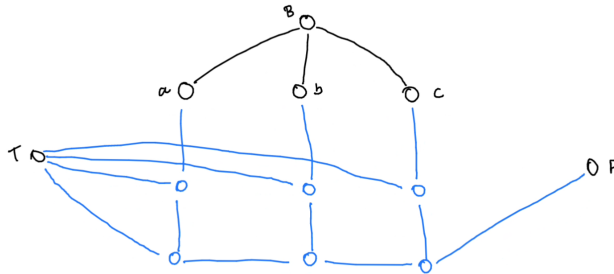
Entrada: um grafo G

Saída: **Verdadeiro**, se existe uma coloração $c: V(G) \rightarrow \{1, 2, 3\}$ tal que se $uv \in E(G)$, então $c(u) \neq c(v)$; e **Falso**, caso contrário.

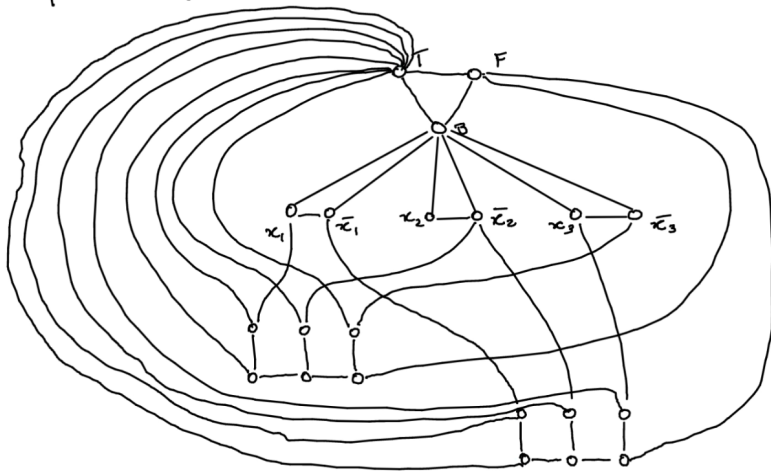


Dada uma fórmula Φ , vamos construir o grafo $G(\Phi)$

1. Crie os vértices v_i e \bar{v}_i para cada variável v_i de Φ
2. Crie três vértices adicionais chamados T, F, B , coloque todas as arestas entre esses três vértices, e ligue todos os vértices representando literais ao vértices B .
3. Conecte cada vértice literal ao vértice que representa a sua negação.
4. Para cada cláusula $(a \vee b \vee c)$, conecte o seguinte *gadget* de 6 vértices e 13 arestas



$$\Phi = (\kappa_1 \vee \bar{\kappa}_2 \vee \bar{\kappa}_3) \wedge (\bar{\kappa}_1 \vee \bar{\kappa}_2 \vee \kappa_3)$$



- Φ é satisfazível se e somente se $G(\Phi)$ é 3-colorível

Se tivermos um algoritmo eficiente para o problema da 3-coloração, então temos um algoritmo eficiente para o 3-SAT

Algoritmo para 3-SAT(Φ)

1. Construa $G(\Phi)$
2. Rode o algoritmo eficiente para o problema da 3-coloração sobre $G(\Phi)$
3. Retorne o mesmo valor lógico do resultado do passo anterior

- Φ é satisfazível se e somente se $G(\Phi)$ é 3-colorível
 - precisamos demonstrar!

Se tivermos um algoritmo eficiente para o problema da 3-coloração, então temos um algoritmo eficiente para o 3-SAT

Algoritmo para 3-SAT(Φ)

1. Construa $G(\Phi)$
2. Rode o algoritmo eficiente para o problema da 3-coloração sobre $G(\Phi)$
3. Retorne o mesmo valor lógico do resultado do passo anterior

- Φ é satisfazível se e somente se $G(\Phi)$ é 3-colorível
 - precisamos demonstrar!
- $G(\Phi)$ pode ser construído de forma eficiente

Se tivermos um algoritmo eficiente para o problema da 3-coloração, então temos um algoritmo eficiente para o 3-SAT

Algoritmo para 3-SAT(Φ)

1. Construa $G(\Phi)$
2. Rode o algoritmo eficiente para o problema da 3-coloração sobre $G(\Phi)$
3. Retorne o mesmo valor lógico do resultado do passo anterior

- Φ é satisfazível se e somente se $G(\Phi)$ é 3-colorível
 - precisamos demonstrar!
- $G(\Phi)$ pode ser construído de forma eficiente
 - precisamos analisar!

Se tivermos um algoritmo eficiente para o problema da 3-coloração, então temos um algoritmo eficiente para o 3-SAT

Algoritmo para 3-SAT(Φ)

1. Construa $G(\Phi)$
2. Rode o algoritmo eficiente para o problema da 3-coloração sobre $G(\Phi)$
3. Retorne o mesmo valor lógico do resultado do passo anterior


- Vamos aprender a dizer que um problema A é tão difícil quanto um B

- Vamos aprender a dizer que um problema A é tão difícil quanto um B
 - redução

- Vamos aprender a dizer que um problema A é tão difícil quanto um B
 - redução
- Vamos aprender a classificar problemas:

- Vamos aprender a dizer que um problema A é tão difícil quanto um B
 - redução
- Vamos aprender a classificar problemas:
 - P, NP, NP-difícil, NP-completo

Sobre o Curso



- Avisos importantes
- Objetivos
- Ementa da disciplina
- Recomendações
- Bibliografia e outros materiais
- Critérios de avaliação regular
- Mecanismo de recuperação
- Plágio
- Dias, horários e locais das aulas
- Dados importantes
- Atendimento
- Notas
- Aúdas

MCTA003-17 - Análise de Algoritmos

Segundo Quadrimestre de 2022

Turma(s) Nenhum

Professor Márcio Sambinelli

E-mail: m.sambinelli@ufabc.edu.br

📌 [Cotas de Sugestões](#)

Avisos importantes

- 2023 - Página no ar.

Objetivos

- Apresentar noções e conceitos de complexidade de computação;
- Apresentar métodos e conceitos que permitam ao aluno, de maneira confiável, avaliar a qualidade de um algoritmo. A essência destes métodos e conceitos estará focalizada no cálculo de complexidade e prova de correção de algoritmos;
- Caracterizar técnicas gerais de desenvolvimento de algoritmos que permitam ao aluno melhor projetá-los conforme suas necessidades. As técnicas gerais escolhidas a serem estudadas são Divisão e Conquista, Método Guloso e Programação Dinâmica;
- Apresentar noções básicas de Classes de Complexidade, em particular as classes P, NP e NP-Completo.

Ementa da disciplina

Conceitos básicos: recorrências, medidas de complexidade: melhor caso, caso médio e pior caso. Técnicas gerais de projeto de algoritmos: divisão e conquista, método guloso e programação dinâmica. Classes de complexidade: P, NP e NP-completude.

Recomendações

Disciplinas: Matemática Discreta, Algoritmos e Estruturas de Dados I

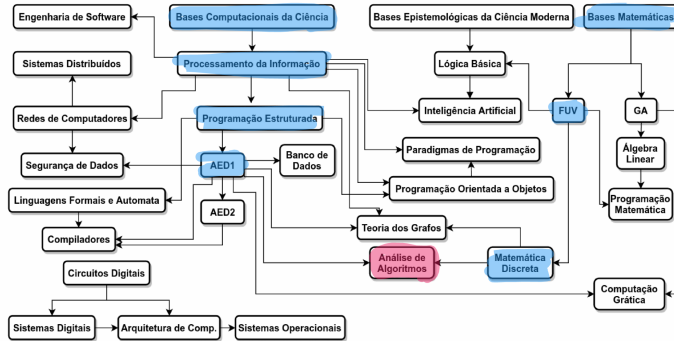


<http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q2-AA/index.html>

1. Técnicas de solução de problemas
 - divisão e conquista
 - gulosos
 - programação dinâmica
2. Provas de correção de algoritmos
 - demonstração (prova por indução, contradição e etc)
3. Análise do tempo de execução de um algoritmo
 - Vocabulário e técnicas de análise de algoritmos
 - Notação assintótica
 - Método da iteração, substituição, árvore de recorrência e Teorema Mestre
4. Complexidade
 - Redução entre problemas
 - Problemas P, NP, NP-difíceis e NP-completos

Disciplinas: Matemática Discreta, Algoritmos e Estruturas de Dados I

ANEXO G: FLUXOGRAMA DAS RECOMENDAÇÕES ENTRE DISCIPLINAS OBRIGATORIAS DO BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO



Para facilitar o acompanhamento do curso, é recomendado que você já possua:

- Conhecimentos de programação, com boas noções de algoritmos
- Familiaridade com estruturas de dados básicas (vetores, listas, pilhas, filas e árvores)
- Capacidade para reconhecer argumentos lógicos em uma prova matemática (**indução**)
- Familiaridade com álgebra matemática (conjuntos, somatórios, leis dos logaritmos e exponenciação)

- Quartas-feiras: 21h - 23h (sala S-212-0)
- Sextas-feiras: 19h - 21h (sala S-214-0)

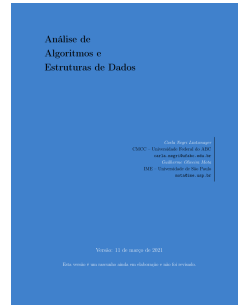
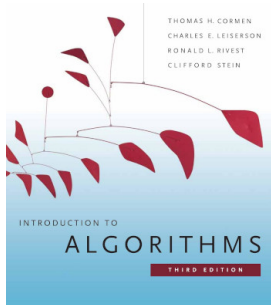
🔥🔥🔥 **Enquete (até 02/06):** <https://forms.gle/WCwVByrBeNcmYKnm8>

Professor:

- horário a definir

Monitores:

- três monitores
- horário a definir



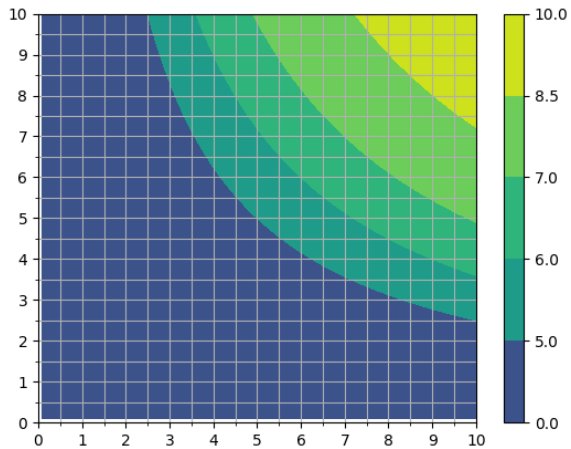
A média final antes da REC (**MF**) será calculada da seguinte forma:

$$MF = 0.85\sqrt{P1 \times P2} + 0.15L,$$

onde $P1$ e $P2$ são avaliações e L é a média aritmética das listas de exercícios

O seu conceito final antes da REC (**CF**) será:

$$CF = \begin{cases} \text{A,} & \text{se } MF \in [8.5; 10.0] \\ \text{B,} & \text{se } MF \in [7.0; 8.5] \\ \text{C,} & \text{se } MF \in [6.0; 7.0] \\ \text{D,} & \text{se } MF \in [5.0; 6.0] \\ \text{F,} & \text{se } MF < 5.0 \\ \text{O,} & \text{se o total de faltas exceder 25\% das aulas} \end{cases}$$



A1	A2	L	MF
10	10	10	10.
5	6	10	6.1556417
5	6	7	5.7056417
3	6	10	5.1062446
3	8	7.5	5.2891326
2	8.5	10	5.0046398

- Requisito: conceito final **D** ou **F**.
- Escopo: todos os temas vistos durante o quadrimestre.

A nota obtida na avaliação de recuperação (**NR**) será usada para obter a nota final com recuperação (**NFR**), que consiste na média:

$$NFR = \frac{MF + NR}{2}$$

O conceito final com recuperação (**CFR**) será calculado da seguinte maneira:

$$CFR = \begin{cases} C, & \text{se } CF = D \text{ e } NFR \geq 6; \\ D, & \text{se } CF = D \text{ e } NFR < 6; \\ D, & \text{se } CF = F \text{ e } NFR \geq 5; \\ F, & \text{caso contrário.} \end{cases}$$

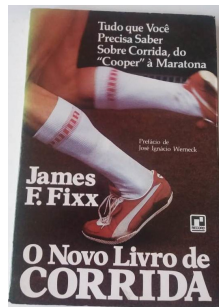
Avaliações:

- Avaliação 1: 07/07 (Sex)
- Avaliação 2: 18/08 (Sex)
- Avaliação substitutiva: 22/08 (Ter)

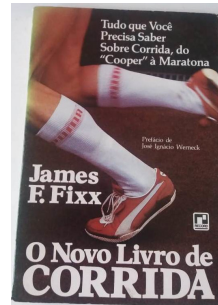
Recuperação:

- 23/09 (Sáb) às 10h – Sala a definir
Avaliação substitutiva está disponível apenas para casos excepcionais.

- Extremamente importante



- Extremamente importante



- Simulado da prova
- Podem ser entregues no dia das avaliações
 - Não valem nota, mas valem moral

Exercícios de Revisão

Prove que se $a < b$ e $c < d$, então $c - b < d - a$.

Avalie sem calculadora: $\log_4 \frac{u^2}{\sqrt{v}}$, sabendo que $\log_4 u = 3,2$ e $\log_4 v = 1,3$

Avalie a expressão

$$\sum_{j=0}^{\log_{4/3} n - 1} 2^j \sqrt{\left(\frac{3}{4}\right)^j n}$$

Prove que a soma dos n primeiros cubos perfeitos é $\left(\frac{n(n+1)}{2}\right)^2$ para todo $n \geq 1$.

Seja S um vetor. O que o algoritmo a seguir faz?

- 1: **Função** $\text{CATIORO}(S, i, j)$
- 2: **Se** $i \geq j$ **então**
- 3: **Devolve sim**
- 4: **Se** $S[i] == S[j]$ **então**
- 5: **Devolve** $\text{CATIORO}(S, i + 1, j - 1)$
- 6: **Devolve não**