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

UFABC

Outline

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

UFABC

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.

UFABC

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)

Um breve tour

UFABC

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

Algoritmo para a sequência de Fibonacci

UFABC

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

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Algoritmo para a sequência de Fibonacci

UFABC

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

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Definição

A *sequência de Fibonacci* é a sequência de números $F_1, F_2, F_3, \ldots$, 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:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Definição

A *sequência de Fibonacci* é a sequência de números $F_1, F_2, F_3, \ldots$, 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

Algoritmo 1

UFABC

```
1: Função FIB1(n)
2:     Se n ≤ 2 então
3:         Devolve 1
4:     Devolve FIB1(n − 1) + FIB1(n − 2)
```

Algoritmo 1

UFABC

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

Perguntas:

1. Esse algoritmo resolve o problema?

UFABC

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

Perguntas:

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

Esse algoritmo é bom?

UFABC

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

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

Algoritmo 1: Assembly

```
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
```

UFABC

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

$F_{3000} = 410615886307971260333568378719267105220125108637369252408885430926905584$
$274113403731330491660850044560830036835706942274588569362145476502674373045446852160486606292497360503469773453733196887405847255290082049086907512622059054542195889758031109222670849274793859539133318371244795543147611073276240066737934085191731810993201706776838934766764778739502174470268627820918553842225858306408301661862900358266857238210235802504351951472997919676524004784236376453347268364152648346245840573214241419937917242918602639810097866942392015404620153818671425739835074851396421139982713640679581178458198658692285968043243656709796000$

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

Desktop (AMD Ryzen 9 5900X)

UFABC

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

Frontier

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/

Esse algoritmo é bom?

UFABC

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

Esse algoritmo é bom?

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

- E agora, José?!

Algoritmo 2

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** $novo$

Algoritmo 2

UFABC

```
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?

Algoritmo 2

UFABC

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?

Esse algoritmo é bom?

UFABC

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

Esse algoritmo é bom?

UFABC

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!

UFABC

1: **Função** FIB3(n)

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

Perguntas:

1. Esse algoritmo resolve o problema?

UFABC

1: **Função** 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?

Esse algoritmo é bom?

UFABC

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 o melhor algoritmo?

UFABC

Qual algoritmo implementar?

- Algoritmo 1
- Algoritmo 2
- Algoritmo 3

Abordagem em Análise de Algoritmos

UFABC

- 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.

$$T_1(n) = 2^{n+1} - n - 2 = \mathcal{O}(2^n)$$
$$T_2(n) = n^2 + 5n + 3 = \mathcal{O}(n^2)$$
$$T_3(n) = n^{1,5} + n + \lg(n) + 3 = \mathcal{O}(n^{1,5})$$

UFABC

- 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.

$$T_1(n) = 2^{n+1} - n - 2 = \mathcal{O}(2^n)$$
$$T_2(n) = n^2 + 5n + 3 = \mathcal{O}(n^2)$$
$$T_3(n) = n^{1,5} + n + \lg(n) + 3 = \mathcal{O}(n^{1,5})$$

$$n^{1,5} << n^2 << 2^n$$

”Algoritmo” para o problema da 3 coloração de grafos

Problema de 3-satisfatibilidade booliana

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 de 3-satisfatibilidade booliana

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.

Problema de 3-satisfatibilidade booliana

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

UFABC

Problema: 3-coloração de grafos (3-coloração)

Entrada: uma grafo G

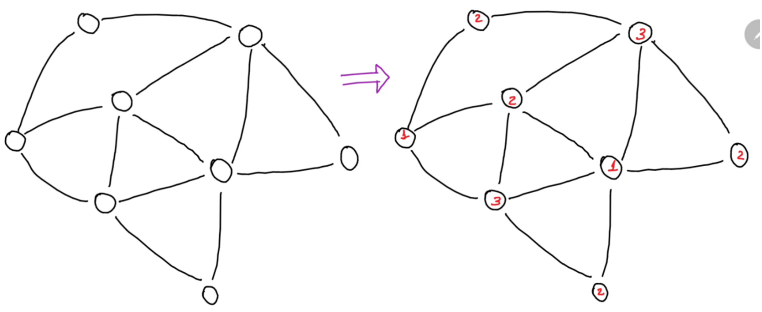
Saída: Verdadeiro, se existe uma coloração $c\colon V(G) \to \{1, 2, 3\}$ tal que se $uv \in E(G)$, então $c(u) \neq c(v)$; e Falso, caso contrário.

UFABC

Problema: 3-coloração de grafos (3-coloração)

Entrada: uma grafo G

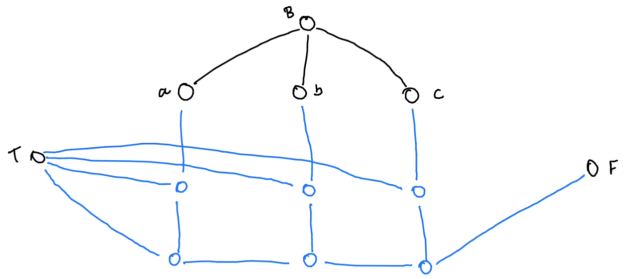
Saída: Verdadeiro, se existe uma coloração $c\colon V(G) \to \{1, 2, 3\}$ tal que se $uv \in E(G)$, então $c(u) \neq c(v)$; e Falso, caso contrário.



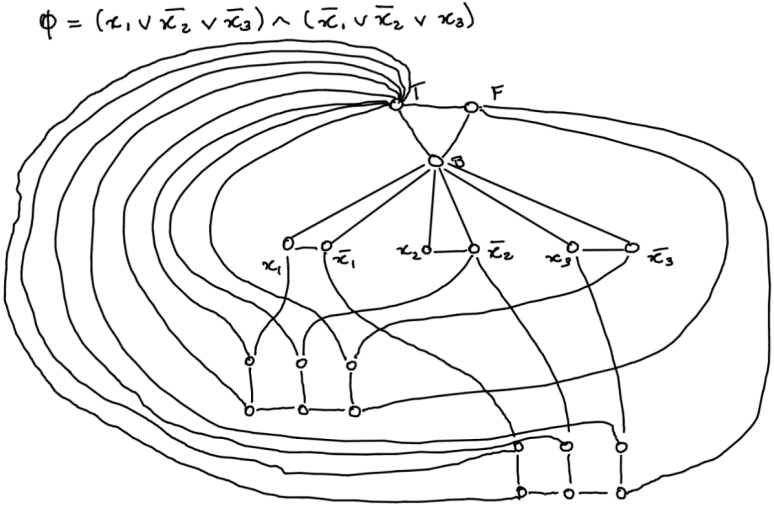
3-coloração é tão difícil quanto o 3-SAT

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 clausula $(a \vee b \vee c)$, conecte o seguinte *gadget* de 6 vértices e 13 arestas



Exemplo de $G(\Phi)$



3-coloração é tão difícil quanto o 3-SAT

- 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

3-coloração é tão difícil quanto o 3-SAT

- 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

3-coloração é tão difícil quanto o 3-SAT

- 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

3-coloração é tão difícil quanto o 3-SAT

- 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

UFABC

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

UFABC

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

UFABC

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

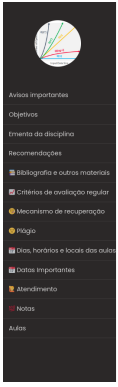
Abordagem em Análise de Algoritmos

UFABC

- 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

Informações



MCTA003-17 - Análise de Algoritmos

Segundo Quadrimestre de 2022

Turma(s): Noturno

Professor: Maycon Sambinelli

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

Caixa de Sugestões

Avisos importantes

- 20/05 - 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 corretude de algoritmos;
- Caracterizar técnicas gerais de desenvolvimento de algoritmos que permitam ao aluno melhor projetá-los conforme sua natureza. 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/2024Q2-AA/index.html

Neste Curso

UFABC

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

Recomendações

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 logarítmos e exponenciação)

Sobre as Aulas

- Terças-feiras: 16h - 18h (sala S-305-1);
- Quintas-feiras: 14h - 16h (sala S-305-1);

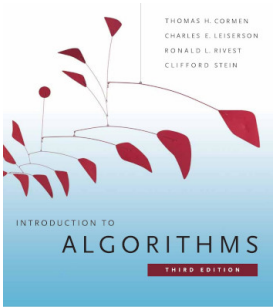
Atendimento

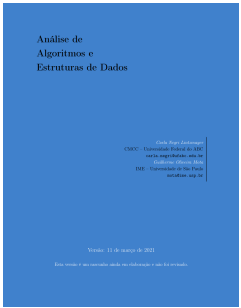
UFABC

Professor:

- Quintas-feiras: 16h-18h (Maycon Sambinelli)

Bibliografia





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

$$MF = 0.5P1 + 0.5P2,$$

onde $P1$ e $P2$ são avaliações

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

$$CF = \begin{cases} \text{A}, & \text{se } MF \in [8.5; 10.0] \\ \text{B}, & \text{se } MF \in [7.0; 8.5) \\ \text{C}, & \text{se } MF \in [6.0; 7.0) \\ \text{F}, & \text{se } MF < 6.0 \end{cases}$$

Datas Importantes

UFABC

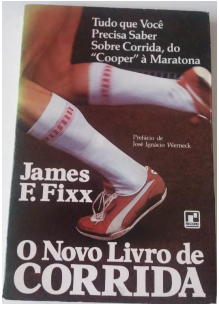
Avaliações:

- Avaliação 1: ???
- Avaliação 2: ???
- Avaliação substitutiva: ???

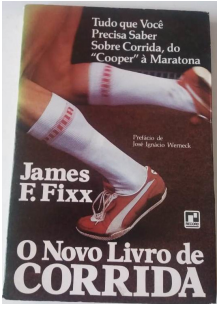
Avaliação substitutiva está disponível apenas para os casos assegurados pelo regulamento da UFABC.

UFABC

- Extremamente importante



- Extremamente importante



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

Exercícios de Revisão

Exercício 1

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

Exercício 2

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

Exercício 3

UFABC

Avalie a expressão

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

Exercício 4

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 CATIORO(S, i, j)
2:     Se i ≥ j então
3:         Devolve sim
4:     Se S[i] == S[j] então
5:         Devolve CATIORO(S, i + 1, j − 1)
6:     Devolve não
```