

# Introdução à Programação Competitiva



Foto de Clemens van Lay, no Unsplash (<https://unsplash.com/photos/a-football-field-with-the-words-start-written-on-it-5eThdzpVqyE>)

# O que será este curso?

- Introdução abrangente à programação competitiva
  - Não será tudo o que você precisa para ganhar o mundial
  - É o **primeiro e necessário passo** para estar entre os melhores programadores competitivos do mundo
- Treinamento **INTENSIVO** e **TORTURANTE** de:
  - Análise de problemas complexos
  - Estudo da teoria subjacente
  - Projeto de algoritmos
  - Implementação de algoritmos



## Pré-requisitos

- Já saber o básico de programação em **C++**, **C**, Python ou Java
- Conhecer o básico sobre **Estruturas de Dados** e o básico sobre o **Projeto de Algoritmos**
- Conhecer o básico sobre **Matemática Discreta**
- Conhecer o básico sobre **Análise de Complexidade de algoritmos**, no mínimo a notação Big-O.
- Em resumo: este curso não é para novatos na computação



# O que é Programação Competitiva?

## - Essência:

- **Projetar algoritmos computacionais eficientes para resolver problemas computacionais bem definidos**
- **Implementar corretamente e rapidamente os algoritmos**

## - Esporte:

- É um esporte mental de alta-performance, coletivo ou não
- Dado um conjunto de problemas bem-conhecidos na Ciência da Computação, resolva-os o mais rápido possível

É necessário:

- habilidade matemática
- pensamento computacional
- resolução de problemas



## Benefícios para os competidores

- Excelente modo de aprender os **conceitos, projeto e implementação** de algoritmos computacionais avançados
- Desenvolvimento do **pensamento computacional**
- Desenvolvimento de **resolução de problemas complexos**
- Aprender a "**se virar**", sozinho e em equipe
- Desenvolver capacidade de **debugging**
- **Big Techs** contratam muitos competidores de destaque
- Você se tornará um **cientista da computação melhor**, "top"



## Benefícios para os competidores

- Entenderá a busca "espiritual" por **elegância, parcimônia e performance**
- Aumentará sua capacidade de **projeto e codificação de algoritmos**
- Se tornará um **artesão** dos algoritmos e dominará esse **ofício**
- Cultura que desenvolve **habitos de excelência**
- Conhecimento profundo de **estruturas de dados** e algoritmos
- Salto de conhecimento
- Desenvolver **criatividade**



# Competições presenciais

## IOI: International Olympiad in Informatics

- Estudantes do ensino médio
- Dois dias de cinco horas
- Resolver 3 problemas difíceis por dia
- Divisão em times, mas a competição é individual

Desde o ano 2000 (exceto em 2003 e 2007), os vencedores são de universidades russas e asiáticas.

## ICPC: The International Collegiate Programming Contest

- Estudantes universitários
- Divisão em times de três estudantes, competem em grupo
- Só 1 computador por time
- Resolver de 10 a 15 problemas difíceis em cinco horas
- Competidores podem ver o ranking, exceto na última hora

## SBC: Maratona SBC de Programação

- Etapa classificatória brasileira para o ICPC



# Competições online

Diversas opções de competições online:

- AtCoder
- CodeForces Contest
- CS Academy
- Google CodeJam
- Facebook Hacker Cup
- HackerRank
- TopCoder Open
- Internet Problem Solving Contest (IPSC)
- Yandex



# Mais informações sobre o ICPC

## ICPC: The International Collegiate Programming Contest

- Cada problema resolvido vale 1 ponto para o time
- Ganha quem tem mais pontos
- A cada resposta errada o time é penalizado:  
+20 minutos no tempo total
- Se houver empate entre times, o desempate é feito com os seguintes critérios:
  - a) menor penalidade de tempo
  - b) quem enviou primeiro a última resposta aceita
- Times podem levar qualquer material impresso (mas isso pode variar)



# Mais informações sobre o ICPC

## ICPC: The International Collegiate Programming Contest

- Os problemas são criados para que:
  - a) todas as equipes consigam resolver pelo menos um
  - b) nenhum time consiga resolver todos
  - c) todos os problemas sejam solucionados por um time
- Os times costumam dividir as 5 horas da competição em três fases, de acordo com o tempo da competição e com a dificuldade dos problemas:
  - a) **Início:**  
de 0h até 2-3h
  - b) **Meio:**  
de 2-3h até terminar os problemas fáceis e médios
  - c) **Fim:**  
quando todos os problemas fáceis e médios já foram solucionados e só sobraram os difíceis



# Mais informações sobre o ICPC

## ICPC: The International Collegiate Programming Contest

### Fase de Início: 0h até 2-3h

- Define se o time vai liderar ou correr atrás dos outros times
- Preparar o "template" de código.
- Encontrar os problemas mais fáceis e resolvê-los o mais rapidamente possível, com nenhuma penalidade (ou com o mínimo possível)
- A maioria das equipes roda em "modo individual" nessa fase, ou seja, cada um dos 3 membros tenta resolver os 3 problemas mais fáceis de modo individual; programar "no papel" é fundamental nessa fase; as soluções são enviadas assim que são encontradas.
- Times que precisam do placar para identificar problemas fáceis ficarão sempre para trás.

# Mais informações sobre o ICPC

## ICPC: The International Collegiate Programming Contest

### Fase do Meio: 2-3h até terminar os problemas fáceis e médios

- Nesse ponto todos os 3 membros já leram todos os problemas (que ainda não foram resolvidos) e farão uma classificação dos problemas restantes com base na dificuldade percebida (olhar o placar é OK agora).
- Com base na classificação o time pode preparar uma fila de prioridade dos problemas a serem resolvidos.
- Trabalho em equipe é fundamental nessa fase! Os times costumam trabalhar em 3+0 ou 2+1 nessa fase.



# Mais informações sobre o ICPC

## ICPC: The International Collegiate Programming Contest

### Fase do Fim: só sobraram os problemas difíceis

- Nesse ponto todos os problemas fáceis e médios já foram resolvidos e só sobraram os difíceis: problemas que o time nunca resolvou antes, ou que o time não sabe muito bem como resolver, ou que o time sabe resolver mas demorará muito tempo.
- Para times de ponta, só haverá 1 a 3 problemas desse tipo.
- Hora do "desespero" com o time todo tentando solucionar mais 1 problema e torcendo para não ficarem "agarrados". O objetivo é enviar mais 1 problema certo até 4:59h da competição!
- Na última hora o placar será congelado e, portanto, não é possível saber quais problemas outras equipes resolveram.

# Jornada para se tornar um programador competitivo

O caminho será tortuoso, inseguro e infinito.



Foto de Davide Ragusa, no Unsplash (<https://unsplash.com/photos/empty-road-RTivRcYz1Bw>)



# Jornada para se tornar um programador competitivo



É preciso tempo, muito tempo mesmo.

Foto de Sonja Langford no Unsplash (<https://unsplash.com/photos/round-timex-analog-clock-at-233-eIkbSc3SDtI>)

# Jornada para se tornar um programador competitivo



Você se sentirá no purgatório.

Gravura de Gustave Doré, para A Divina Comédia (<https://www.meisterdrucke.pt>)



# Jornada para se tornar um programador competitivo



O treino em equipe  
será intenso...

Foto de Brooke Cagle no Unsplash (<https://unsplash.com/photos/a-group-of-women-doing-push-ups-in-a-gym-vCPRAwU42GA>)

# Jornada para se tornar um programador competitivo



... e o treino individual  
será maior ainda.

Foto de Scott Webb no Unsplash (<https://unsplash.com/photos/woman-on-gym-equipment-xwMlVSqP20U>)



# Jornada para se tornar um programador competitivo

O desespero virá rápido e  
você pensará em desistir.



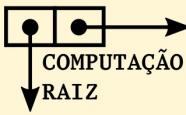
Foto de Valeriia Miller no Unsplash (<https://unsplash.com/photos/a-black-and-white-photo-of-a-woman-sitting-on-a-couch-cTBA-7otLHo>)

# Jornada para se tornar um programador competitivo

Você se tornará seu maior inimigo.



Foto de Oleksandr Kurchev no Unsplash ([https://unsplash.com/photos/a-woman-in-a-sequin-dress-floating-in-water-j3\\_2T66Mv4Y](https://unsplash.com/photos/a-woman-in-a-sequin-dress-floating-in-water-j3_2T66Mv4Y))

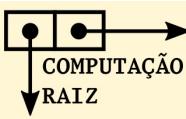


# Jornada para se tornar um programador competitivo

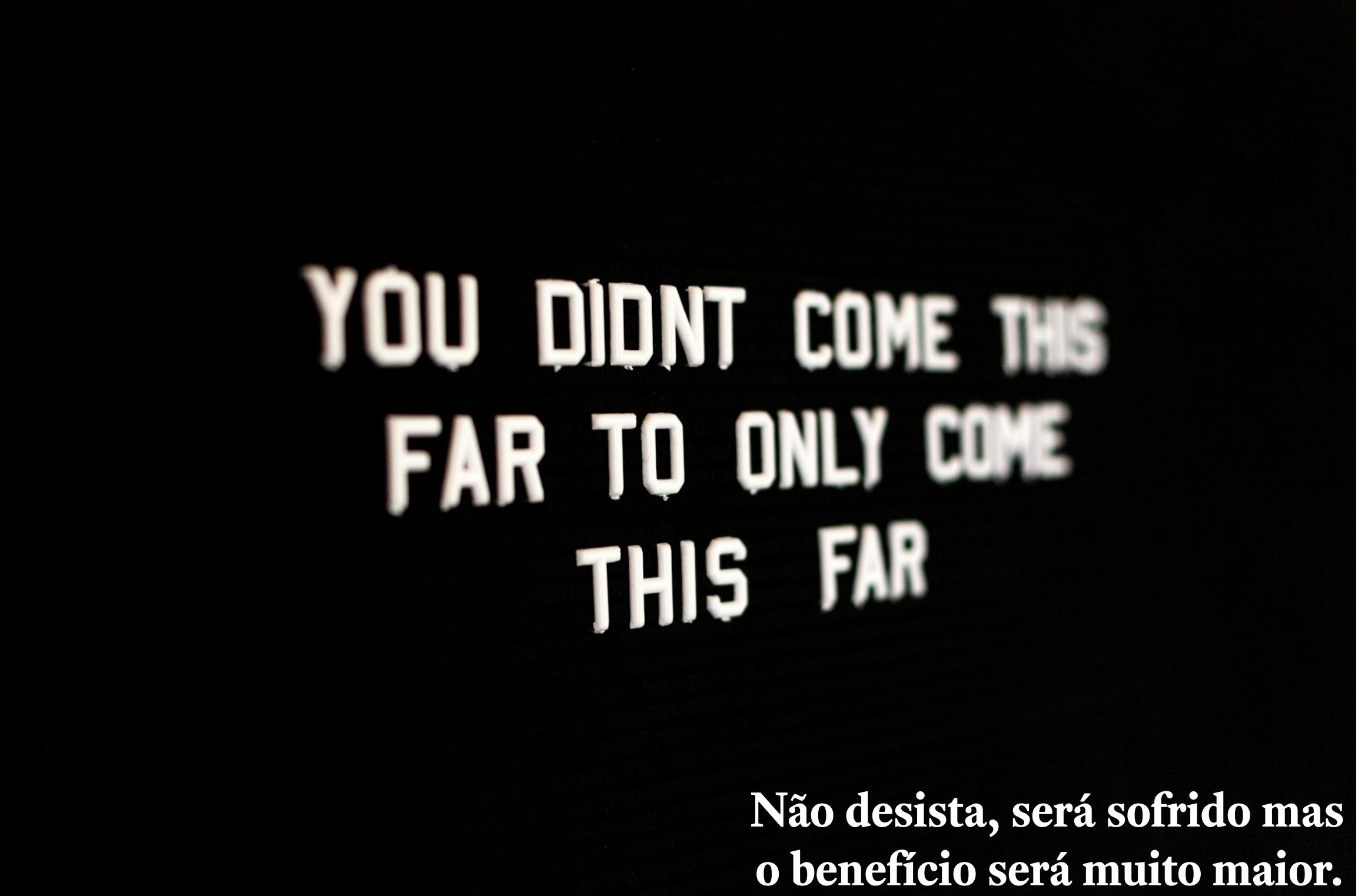


Ajude seu companheiro de time!

Foto de National Library of Scotland, no Unsplash (<https://unsplash.com/photos/a-black-and-white-photo-of-a-group-of-soldiers-Dp8q5tVxz1Y>)



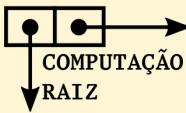
# Jornada para se tornar um programador competitivo



YOU DIDNT COME THIS  
FAR TO ONLY COME  
THIS FAR

**Não desista, será sofrido mas  
o benefício será muito maior.**

Foto de Drew Beamer no Unsplash (<https://unsplash.com/photos/you-didnt-come-this-far-to-only-come-this-far-lighted-text-Vc1pJfvoQvY>)



# Jornada para se tornar um programador competitivo

Por que é tão difícil?

- Problemas realmente complexos
- Algoritmos realmente complexos
- Algoritmos em rápida evolução
- Quantidade de algoritmos aumentando a cada dia

Ser um programador competitivo de ponta não é o objetivo final, mas apenas um meio. O verdadeiro objetivo é se tornar um cientista da computação e um programador de ponta, pronto para produzir software sofisticado e melhor, e de encarar problemas de pesquisa não solucionados em ciência da computação.



# Linguagens de Programação

Não há nem discussão:

**"C++ é a linguagem mais popular na programação competitiva. Em particular, quase todos os programadores competitivos do topo do ranking usam C++ [...]. Em 2023, em nível mundial, C++ foi utilizada em 91% dos envios de respostas."**

(Antti Laaksonen, 2024)

Aprenda C++ e C se quiser ser um programador competitivo. Aprenda outras linguagens para complementar.

Motivos para C++:

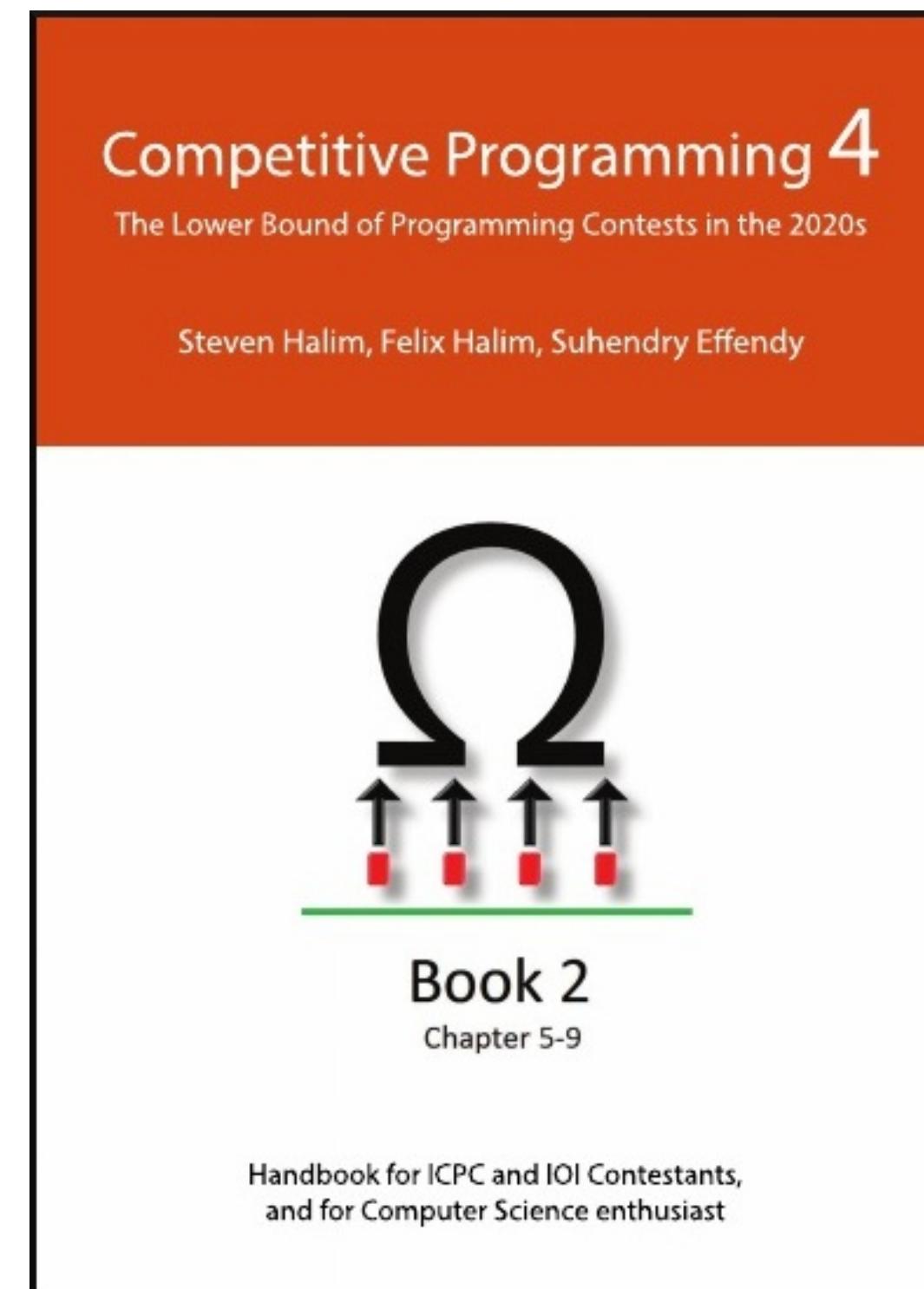
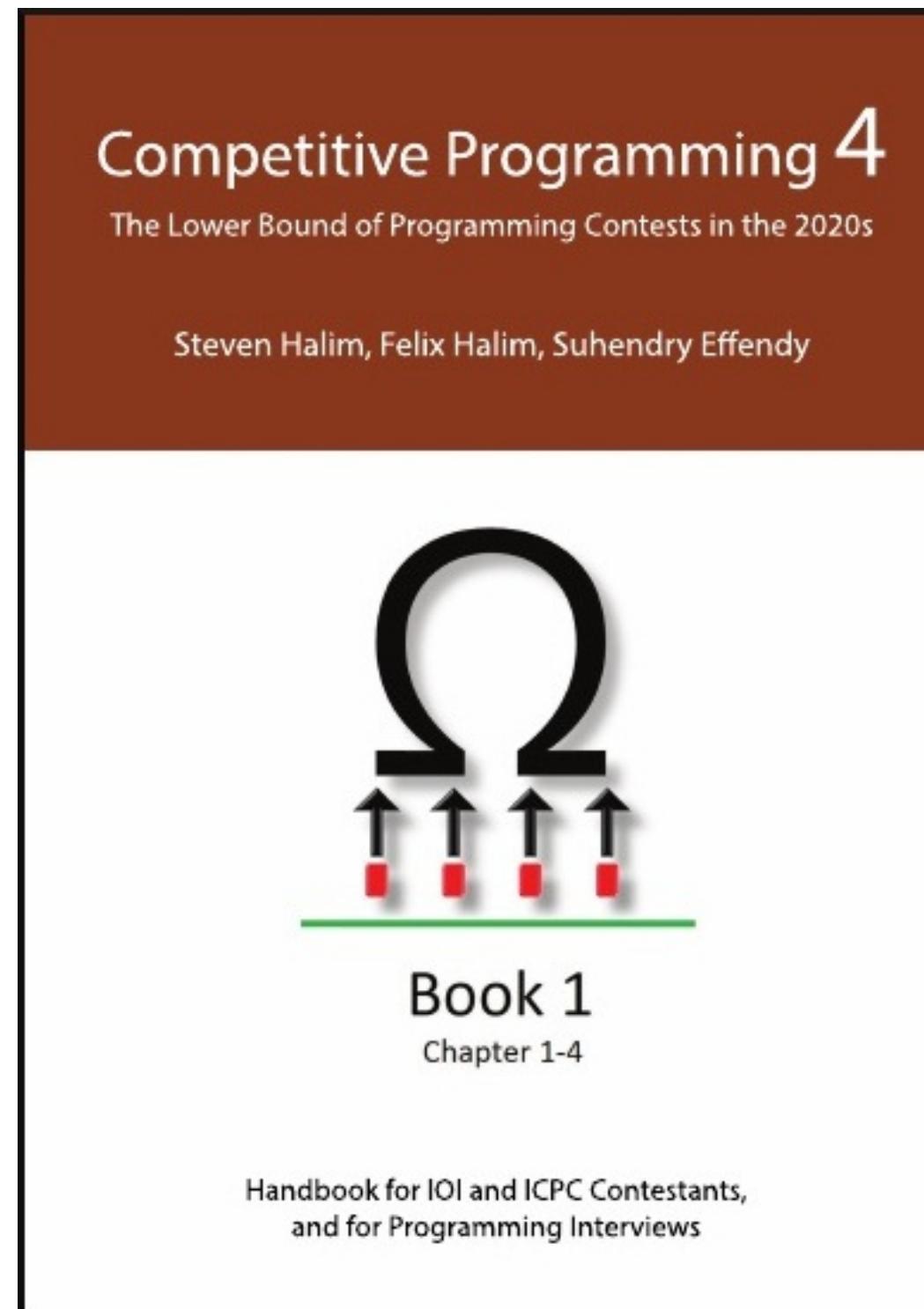
- Eficiência
- Biblioteca com infinidade de estruturas de dados e algoritmos
- Permite o uso de funções C quando necessário

Outras linguagens:

- Pode ser usadas em situações muito especiais, por exemplo, BigInteger em Java ou Python



# Livros que usaremos

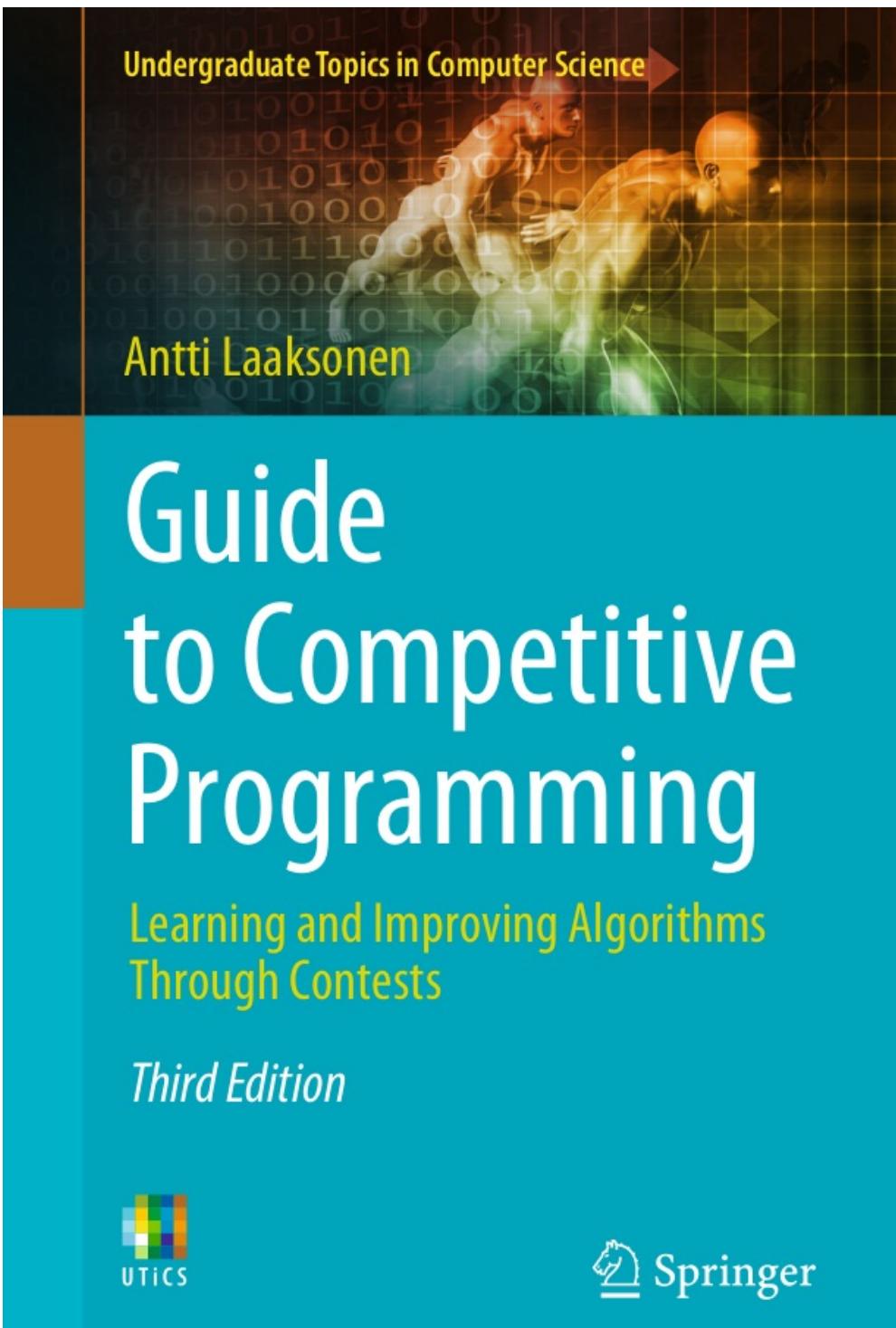


Devem ser lidos várias vezes durante os anos em que você competirá.

**Site do CP Book 4:**  
<https://cpbook.net>

**Visualgo:**  
<https://visualgo.net>

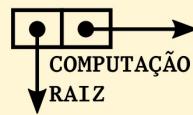
# Livros que usaremos



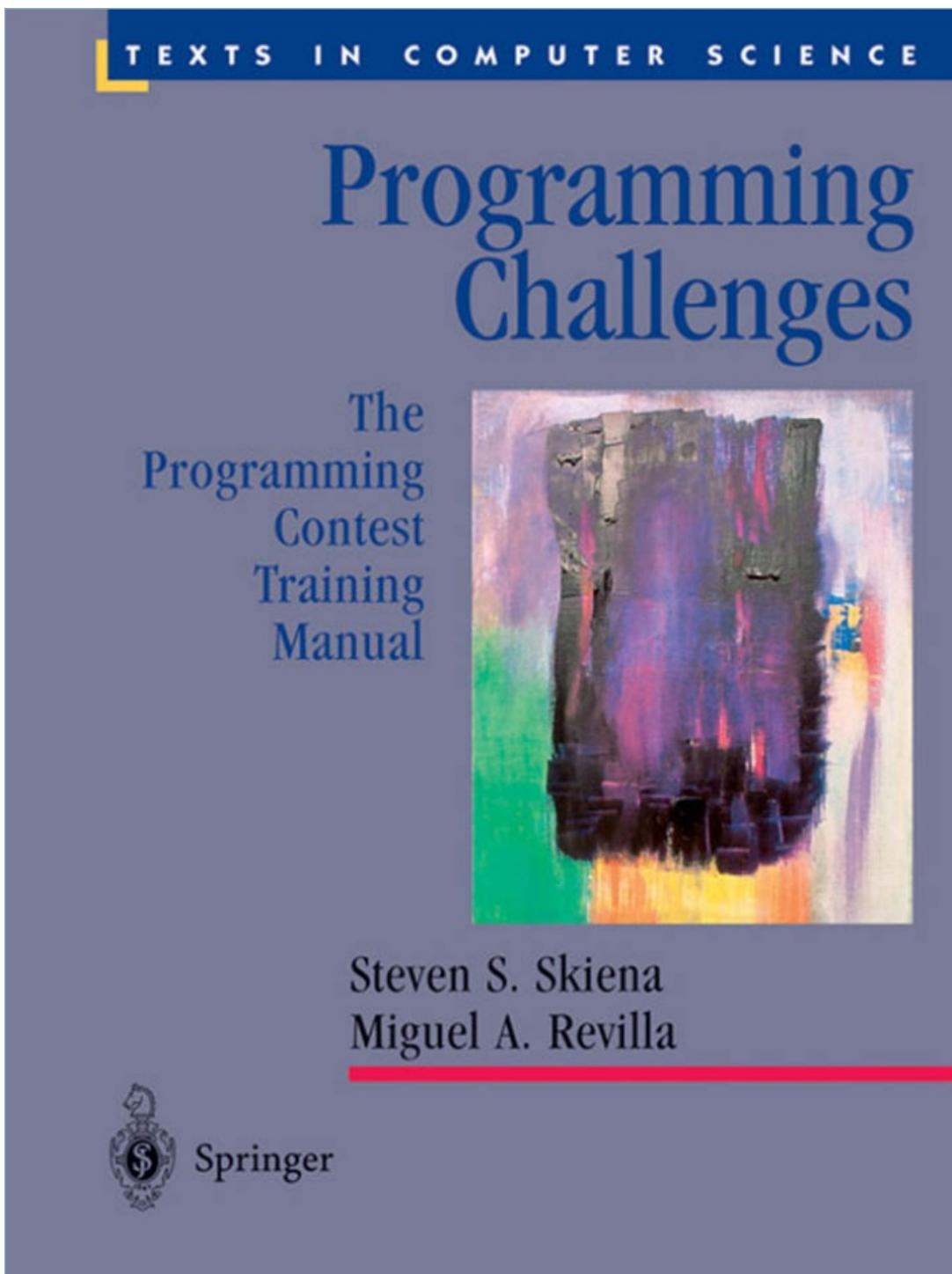
Competitive Programmer's Handbook

Antti Laaksonen

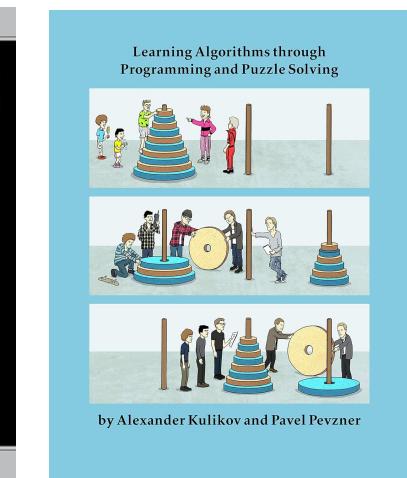
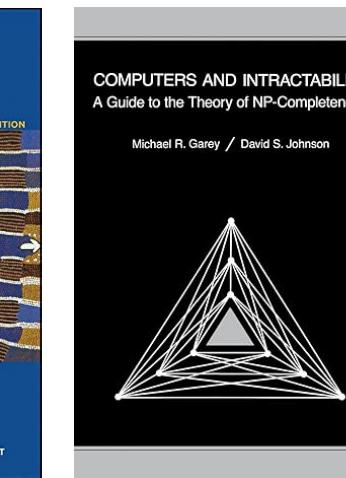
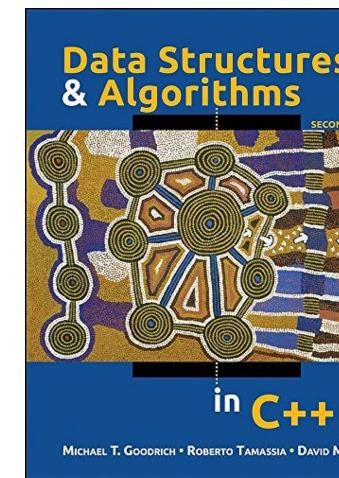
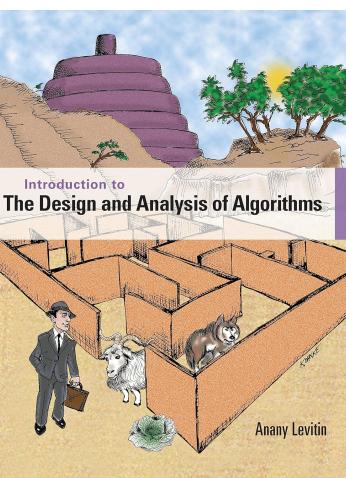
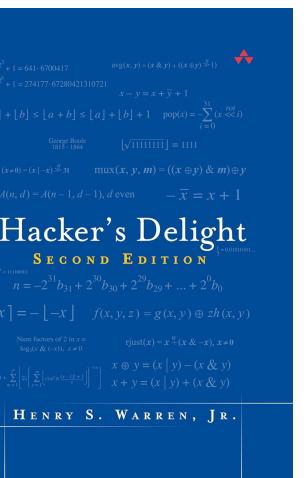
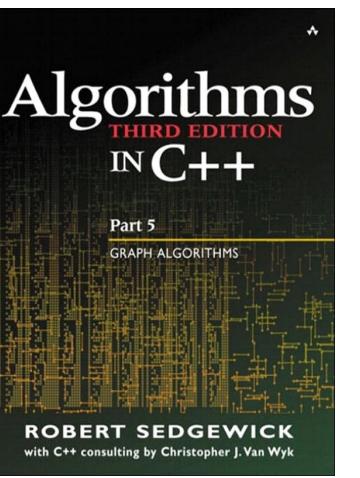
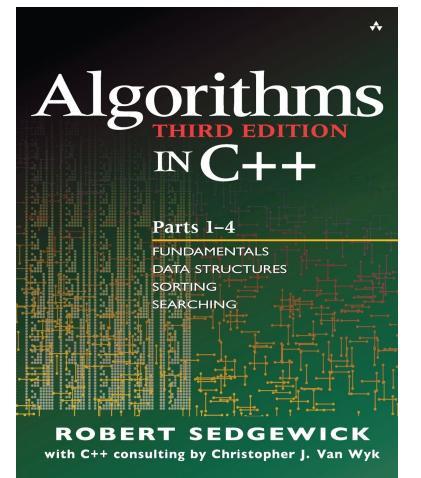
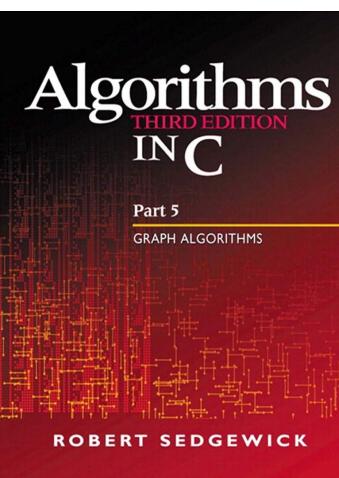
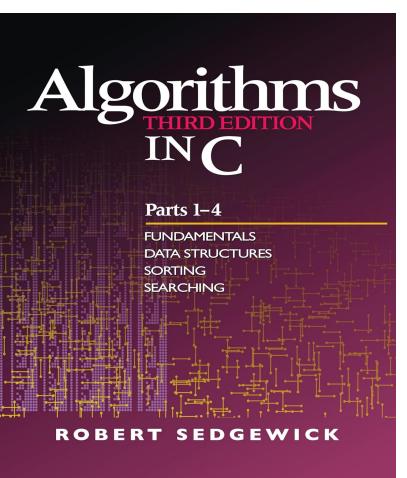
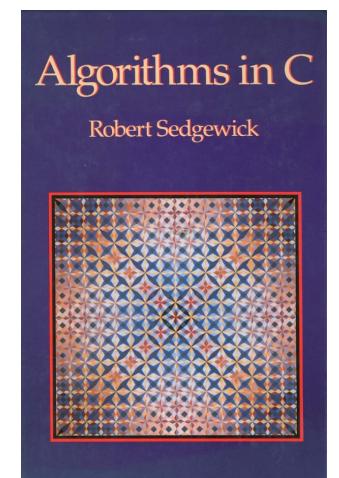
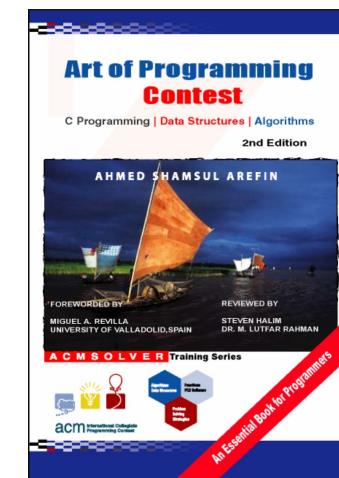
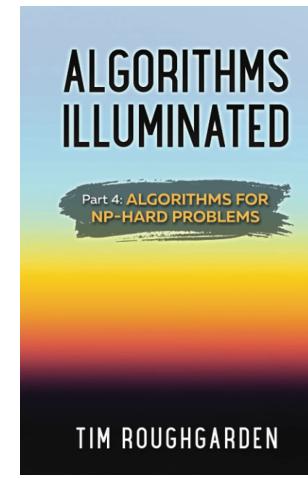
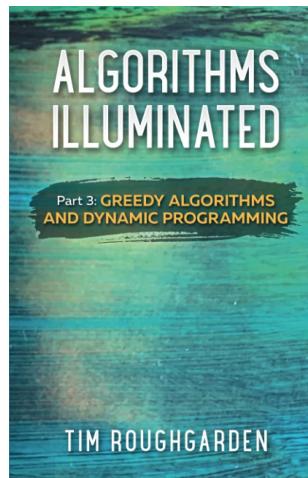
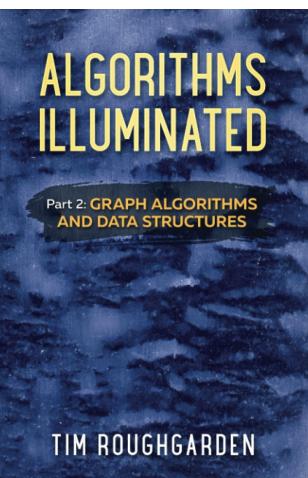
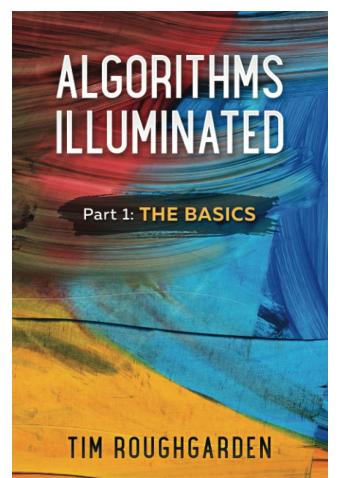
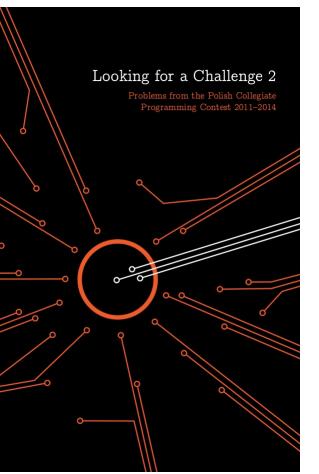
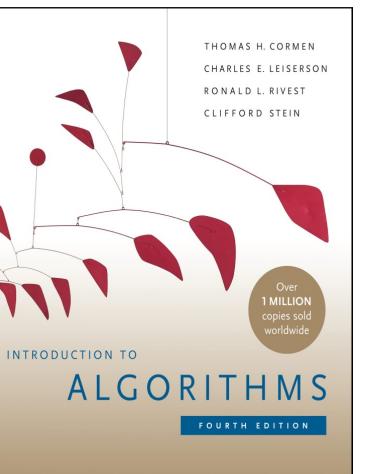
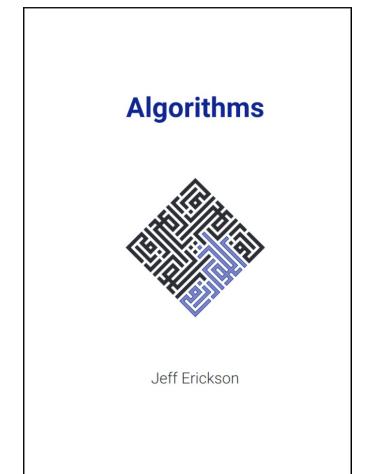
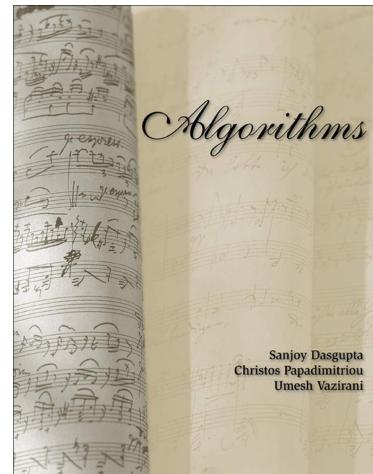
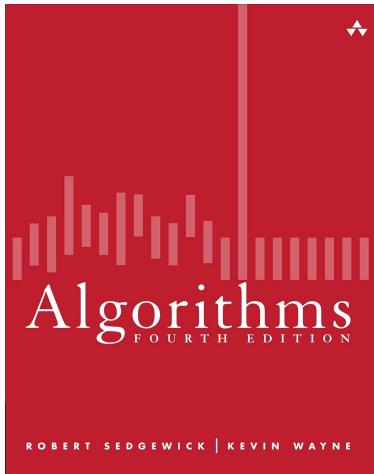
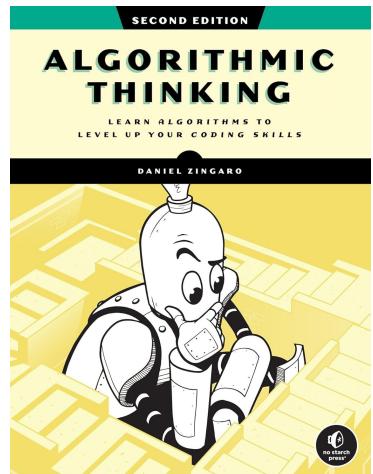
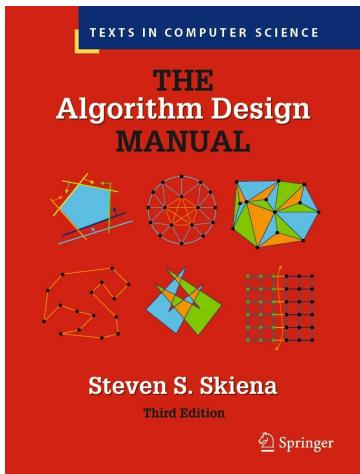
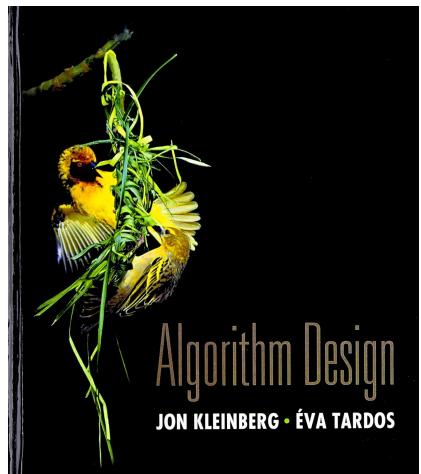
Draft August 19, 2019



# Livros que usaremos



# Livros de referência



# Sites que usaremos

**University of Valladolid (UVa) Online Judge:**

<https://onlinejudge.org>

**Kattis Online Judge (Kattis):**

<https://open.kattis.com>

**Sistema Online BOCA UFES:**

<https://boca.pet.inf.ufes.br/boca/>

**Code Submission Evaluation System (CSES):**

<https://cses.fi/problemset>

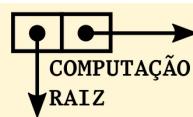
**uHunt:**

<https://uhunt.onlinejudge.org>

**UDebug:**

<https://www.udesign.com>

Crie uma conta!



## Ao usar os juízes online

Leia, interprete e tenha certeza de que você **entendeu todas as especificações do problema** corretamente.

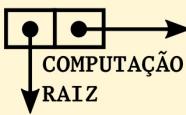
Preste muita atenção aos **limites de input** e **output**, bem como aos **limites de tempo e espaço** do programa.

**Nunca assuma nada** que não esteja explicitamente escrito:

- **não assuma que o input está ordenado**
- **não assuma que um grafo está conectado**
- **não assuma que os inteiros serão sempre positivos**
- **não assuma nada sobre o tamanho máximo dos números**
- **etc.**

O feedback do sistema de avaliação:

- **nunca informará nada** sobre o que pode estar errado
- **nunca mostrará os casos de teste**



## Ao usar os juízes online

A maioria dos juízes online retorna os seguinte vereditos, com ou sem pequenas variações:

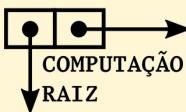
**Accepted (AC)**: seu programa está correto!

**Accepted (PE)**: programa está correto com um errinho de apresentação, mas foi aceito.

**Presentation Error (PE)**: seu programa pode estar correto, mas não está no formato correto e não foi aceito.

**Wrong Answer (WA)**: seu programa está errado.

**Compile Error (CE)**: o compilador não conseguiu compilar seu programa.



## Ao usar os juízes online

A maioria dos juízes online retorna os seguinte vereditos, com ou sem pequenas variações:

**Runtime Error (RE)**: seu programa falhou durante a execução (segfault, exception, etc.)

**Time Limit Exceeded (TL)**: seu programa está errado ou é ineficiente.

**Memory Limit Exceeded (ML)**: seu programa é ineficiente e está usando mais memória do que o permitido.

**Output Limit Exceeded (OL)**: seu programa está errado e está imprimindo muito mais output do que é esperado.



## Ao usar os juízes online

A maioria dos juízes online retorna os seguinte vereditos, com ou sem pequenas variações:

**Restricted Function (RF)**: seu programa está tentando usar uma função não permitida, como `fork()` ou `fopen()`.

**Submission Error (SE)**: ao enviar seu programa algo não está correto.

Outros: depende do sistema.



# Ao usar os juízes online

Tenha **cuidado com a versão dos compiladores utilizados pelos sistemas**, pois você pode ter problemas! Para cada juiz, busque essas informações!

## UVa Online Judge:

- GCC 5.3.0 ANSI C: -lm -lcrypt -O2 -pipe -ansi -DONLINE\_JUDGE
- GCC 5.3.0 C++ 11: -lm -lcrypt -O2 -std=c++11 -pipe -DONLINE\_JUDGE
- Problemas:
  - "**return 0**" obrigatório
  - Variáveis declaradas no início, não dentro de blocos (ex.: **for**)
  - Não tem o que fazer, temos que nos acostumar
- Nas competições o GCC será uma versão mais nova e não teremos problemas, mas você deve ter cuidado e fazer os ajustes necessários.

# Você está pronto para se tornar um programador competitivo?

## Descrição:

Seja  $(x, y)$  um par de números inteiros que representam as coordenadas da casa de um estudante em um plano bidimensional. Existem  $2N$  estudantes e nós queremos agrupá-los em **pares** para formar  $N$  grupos. Seja  $d_i$  a distância entre as casas de 2 estudantes no grupo  $i$ . Forme  $N$  grupos tais que o custo =  $\sum_{i=1}^N d_i$  seja **minimizado**. Imprima o custo mínimo como um número de ponto flutuante com duas casas decimais em uma linha.

## LIMITES:

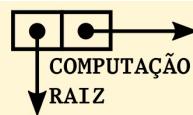
$1 \leq N \leq 8$  e  $0 \leq x, y \leq 1000$ .

## INPUT TÍPICO:

$N = 2$ ; Coordenadas das  $2N = 4$  casas são  $\{1, 1\}$ ,  $\{8, 6\}$ ,  $\{6, 8\}$ ,  $\{1, 3\}$ .

## OUTPUT ESPERADO:

4.83



# Você está pronto para se tornar um programador competitivo?

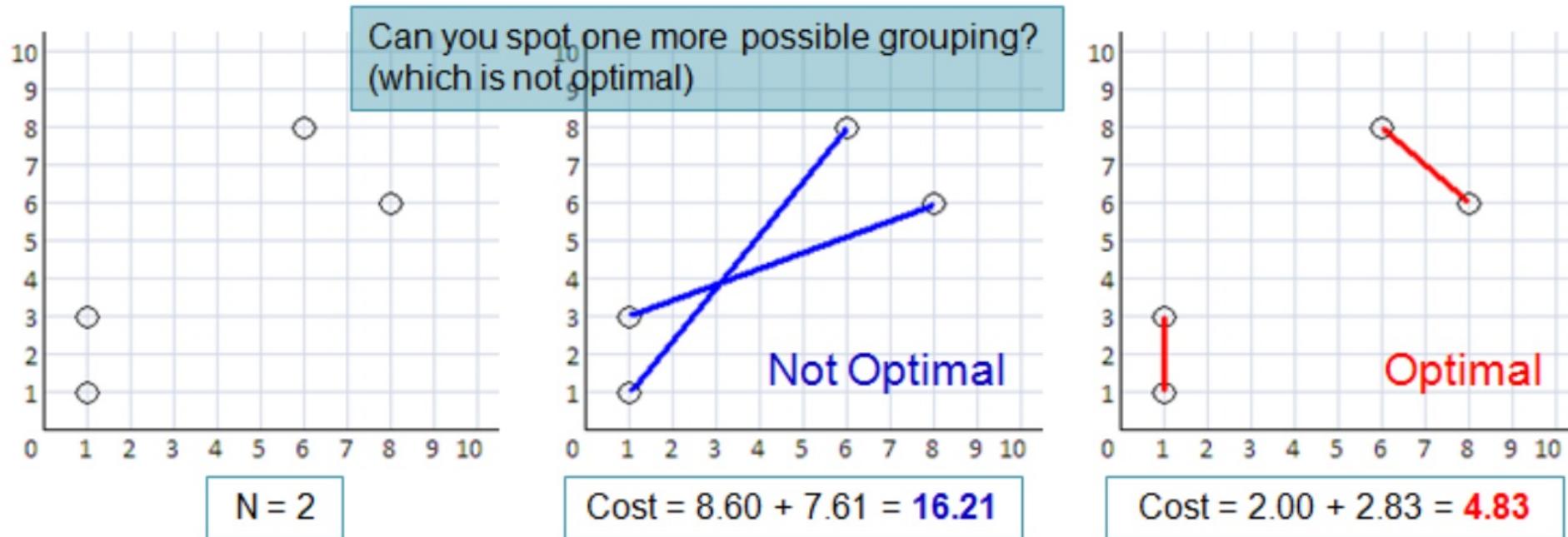


Figure 1.1: Illustration of UVa 10911 - Forming Quiz Teams

**Programador não competitivo A: o perdido**  
Não sabe o que fazer. Esse tipo de problema é novo para ele. Tenta algo por força bruta e dá errado; tenta algum algoritmo guloso e também não acerta; tenta busca completa com backtrackint e excede o limite de tempo.  
Desiste.

**Programador não competitivo B: desiste**  
Pelo menos percebe que está diante de um problema de emparelhamento em grafos. Mas não conhece a solução usando programação dinâmica. Nem tenta, já desiste e tenta resolver outro problema da lista.

**Programador não competitivo C: lento**  
Percebe que é um problema difícil, o **emparelhamento perfeito de peso mínimo em um grafo completo ponderado**. Percebe também que como o input é pequeno, pode ser solucionado com **programação dinâmica** através do uso de **bitmask**. Codifica, debuga e acerta a resposta após 3 horas.

# Você está pronto para se tornar um programador competitivo?

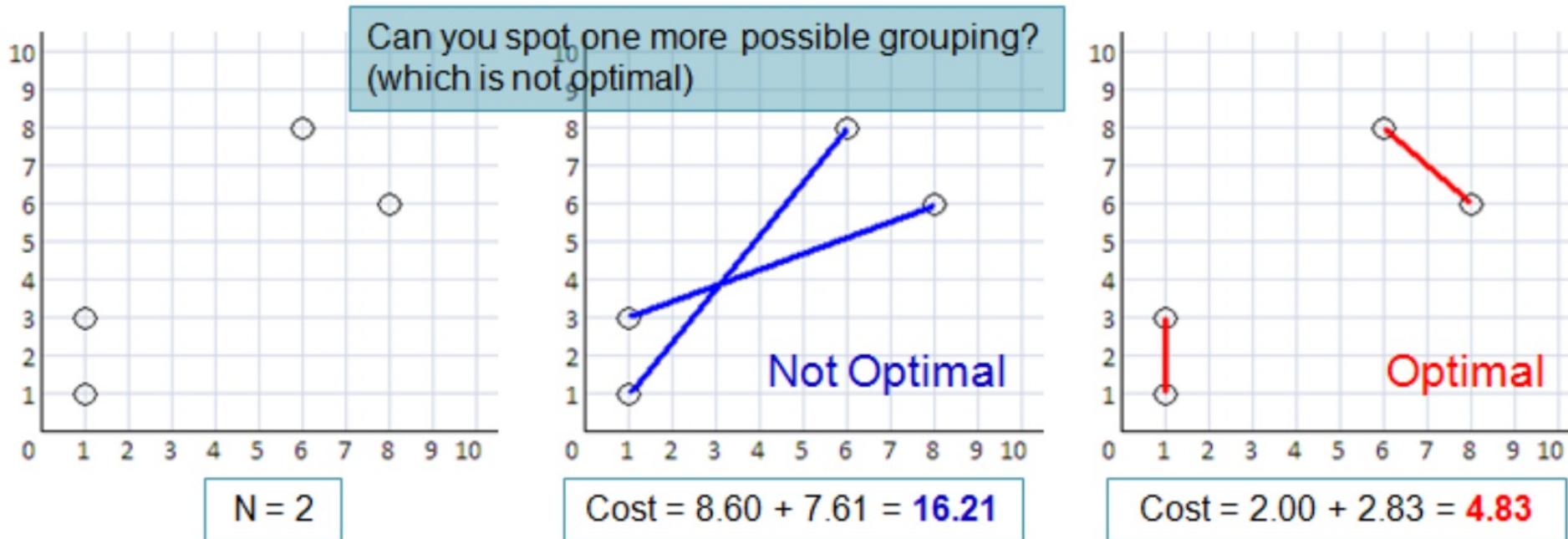


Figure 1.1: Illustration of UVa 10911 - Forming Quiz Teams

## Programador competitivo D:

Faz tudo o que o programador não competitivo C fez, mas em um tempo menor do que 30 minutos.

## Programador competitivo E:

Resolve o problema em menos de 10 minutos e talvez conheça diversas soluções possíveis para variações mais difíceis desse problema.

# Dicas iniciais de estudo

Por toda a vida:

- Funciona para atletas
- Funciona para músicos
- Funciona para arte
- Funcionará para você!

```
#include <stdio.h>

int main (void)
{
    while (1)
    {
        estudar();
        praticar();
        ensaiar();
        ensaiar_com_figurino();
        realizar();
    }
}
```



# Dicas iniciais de estudo

**Estude e programe MUITO:**

- Resolva muitos problemas
- Resolva muitos problemas difíceis
- A quantidade é menos importante do que a dificuldade

**Siga EXATAMENTE o cronograma indicado**

**Faça EXATAMENTE os programas e exercícios indicados**

**Faça os programas e exercícios NA ORDEM que estão indicados**

**Tente FAZER POR CONTA PRÓPRIA antes de buscar ajuda**



# Dicas iniciais de estudo

## 1) Aprenda a digitar rápido!

- Quando você consegue resolver a mesma quantidade de problemas que seu oponente, a capacidade de digitar mais rápido faz muita diferença!

- Se você não souber digitar com rapidez, há muitos sites de aulas de digitação.

- Faça um teste de digitação: tente obter pelo menos um mínimo de 60 palavras por minuto (PPM).

<https://10fastfingers.com/typing-test/portuguese>

- Não use teclados US ou outros modelos para competições no Brasil, pois provavelmente os teclados serão ABNT2. Use modelos US se competir fora.



# Dicas iniciais de estudo

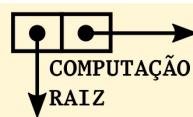
## 2) Identifique rapidamente os tipos e a dificuldade dos problemas!

ID	Categoria	Freqüência
1	Ad Hoc	1-2
2	Estrutura de Dados (principalmente)	0-1
3	Busca Completa (iterativa/recursiva)	1-2
4	Dividir e Conquistar	0-1
5	Gulosos (os não-clássicos)	1
6	Programação Dinâmica (os não-clássicos)	1-2
7	Grafos (exceto fluxo em rede/emparelhamento em grafo)	1
8	Matemática	1-2
9	Processamento de string	1
10	Geometria computacional	1
11	Problemas difíceis, raros, emergentes	2-3

Tipo	Categoria	Confiança na Resolução
A1	Eu já resolvi esse tipo de problema antes	E tenho certeza de que posso resolver de novo (rapidamente).
A2	Eu já resolvi esse tipo de problema antes	E tenho certeza de que posso resolver de novo (lentamente).
B	Eu já vi esse tipo de problema antes	Mas daquela vez eu sabia que ainda não conseguia resolver.
C	Eu nunca vi esse tipo de problema antes	Danou-se... ver abaixo algumas dicas.

Para ser **competitivo** você deve ser capaz de maximizar os problemas do tipo A1 e minimizar os outros. Isso é feito através do estudo de algoritmos, estruturas de dados e habilidades de programação. Para **ganhar** você também deve desenvolver habilidades de resolução de problemas para tratar os problemas B e C. Algumas habilidades que você precisará desenvolver:

- Reduzir um problema em outro mais fácil
- Identificar dicas ou propriedades especiais
- Comprimir dados de input
- Listar e observar padrões
- etc.
- Reduzir um problema NP-Hard non problema em questão
- Atacar o problema de outro ponto de vista
- Trabalhar e reordenar as fórmulas matemáticas
- Realizar análise de casos e sub-casos do problema
- etc.



# Dicas iniciais de estudo

## 3) Faça a Análise de Complexidade do algoritmo!

- Dado os limites máximos de input, o algoritmo atual pode resolver o problema dentro da complexidade exigida de tempo/espaço, e dentro dos limites de tempo/memória?
- Escolha a solução mais simples que funciona!
- Como uma regra grosseira, considere que um processador moderno executa aproximadamente 1 BIPS. Faça a análise levando em conta essa aproximação.
- Exemplo:
  - Input: 1 milhão de números
  - Limite: 1s

$$O(n^2) \approx (10^6)^2 \approx 10^{12} \text{ instruções} \therefore \frac{10^{12} \text{ instruções}}{10^9 \text{ IPS}} \approx 1000 \text{ s}$$

$$O(n \log n) \approx 10^6 \times \log_2 10^6 \approx 2 \times 10^7 \text{ instruções} \therefore \frac{2 \times 10^7 \text{ instruções}}{10^9 \text{ IPS}} \approx 0.02 \text{ s}$$

MIPS = milhão de instruções por segundo

BIPS = bilhão de instruções por segundo



## Dicas iniciais de estudo

### 3) Faça a Análise de Complexidade do algoritmo!

- Dependendo dos limites do problema, até um algoritmo ruim como  $O(n^4)$  pode funcionar se n é pequeno. Se esse for o caso, não perca tempo otimizando o algoritmo, é mais importante terminar na frente de outras equipes do que criar o algoritmo mais otimizado possível. Mas isso só é possível se você fizer a análise de complexidade levando em conta os limites do problema!
- Essa análise de complexidade "grosseira" tem uma grande margem de erro mas, mesmo assim, à luz dos limites de tempo, espaço, memória e limites do problema, você pode decidir se precisa otimizar o algoritmo antes de fazer o primeiro envio. Uma rejeição a menos é uma penalidade a menos para o time!



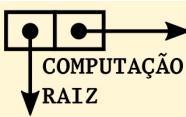
# Dicas iniciais de estudo

## 3) Faça a Análise de Complexidade do algoritmo!

- Evite implementar o algoritmo antes de fazer essa análise e ter certeza de que seu algoritmo é correto e rodará dentro dos limites estabelecidos.

- A tabela ao lado considera que a CPU executa 1MIPS e traz o pior algoritmo que seria aceito dependendo de N:

$n$	Worst AC Algorithm
$\leq [10..11]$	$O(n!), O(n^6)$
$\leq [17..19]$	$O(2^n \times n^2)$
$\leq [18..22]$	$O(2^n \times n)$
$\leq [24..26]$	$O(2^n)$
$\leq 100$	$O(n^4)$
$\leq 450$	$O(n^3)$
$\leq 1.5K$	$O(n^{2.5})$
$\leq 2.5K$	$O(n^2 \log n)$
$\leq 10K$	$O(n^2)$
$\leq 200K$	$O(n^{1.5})$
$\leq 4.5M$	$O(n \log n)$
$\leq 10M$	$O(n \log \log n)$
$\leq 100M$	$O(n), O(\log n), O(1)$

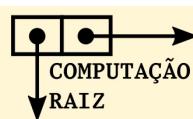


# Dicas iniciais de estudo

## 4) Domine as linguagens de programação!

- Dominar, compreender e saber usar toda a capacidade de sua linguagem é fundamental! Isso inclui: a **linguagem em si** e a **biblioteca** da linguagem.
- A melhor atual é **C++** (com uso ocasional de **C**), mas é importante saber algo de Java (**BigInteger**, **BigDecimal**, **GregorianCalendar**, **Regex**, etc.) e de Python.
- O uso de **regex** das bibliotecas C é fundamental, mesmo que você programe apenas em C++.
- O **tempo de codificação não deve ser um fator limitador!** Depois de achar o algoritmo correto e eficiente, você deve ser capaz de codificá-lo rapidamente. Para isso você deve ter domínio da linguagem e bibliotecas.

Use todas as funcionalidades das bibliotecas! Não implemente estruturas de dados do zero se a biblioteca já disponibilizar.

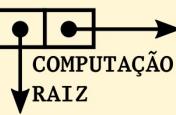


# Dicas iniciais de estudo

## 4) Domine as linguagens de programação!

```
3  
0.1227...  
0.517611738...  
0.734123122344344389923899277...
```

```
1 #include <bits/stdc++.h>  
2  
3 using namespace std;  
4  
5 int main (void)  
6 {  
7     int N;  
8     scanf("%d\n", &N);  
9  
10    while (N--)  
11    {  
12        char x[110];  
13        scanf("0.%[0-9]...\\n", &x);  
14        printf("the digits are 0.%s\\n", x);  
15    }  
16    return 0;  
17 }
```



# Dicas iniciais de estudo

## 5) Domine o debug de código!

- Em primeiro lugar você deve ser capaz de testar seu código com **casos de teste** que você mesmo (ou alguém do time) criou. Nunca confie apenas nos casos de teste que estão no problema, pois eles estão ali apenas para explicar o formato do input e o output esperado.
- **Não gaste envios e acumule penalidades antes de criar os seus próprios casos de teste**, que devem incluir:
  - Os casos do problema
  - Casos fora do limite e valores extremos como  $N = 0$ ,  $N = 1$ , valores negativos, valores iniciais, finais ou intermediários que causem overflow, linhas vazias, árvores, grafos cíclicos, acíclicos, complexos, e desconexos, "out of bonds", etc.
  - Casos grandes, com grande  $N$  (gere aleatoriamente)

# Dicas iniciais de estudo

## 5) Domine o debug de código!

- Não assuma que o input sempre será formatado do jeito correto, principalmente se a descrição do problema não disser nada sobre o formato do input.
  - Coloque espaços, tabs, etc., no input e verifique se seu código funciona com inputs mal formatados.
- Inclua dois ou mais casos de teste idênticos em um mesmo conjunto de inputs e veja se fornecem o mesmo resultado. Isso ajuda a achar variáveis não inicializadas ou variáveis que você esqueceu de resetar.
- Domine ferramentas como o GDB e Valgrind!
- Use diff para comparar outputs.
- Pode ser útil imprimir o código e debugar com lápis.



## Dicas iniciais de estudo

### 6) Estude e resolva problemas TODOS OS DIAS, SEM FALTA!

Talvez este seja o fator mais importante. Não se aprende programação competitiva sem estudar e resolver MUITOS problemas de programação competitiva. Alguns dados interessantes (referentes ao ano de 2020):

#### UVa Online Judge:

- a) Para estar no "Top 500" você deve ter resolvido mais de 700 problemas de diversos níveis de dificuldade;
- b) O n.<sup>º</sup> 124 do ranking resolveu 1262 problemas;
- c) O n.<sup>º</sup> 72 do ranking resolveu 1550 problemas;
- d) O n.<sup>º</sup> 39 do ranking resolveu 2074 problemas.

#### Kattis Online Judge:

- a) Para estar no "Top 100" você precisa de 1793+ pontos;
- b) O n.<sup>º</sup> 9 do ranking tem 5743 pontos.



# Dicas iniciais de estudo

## 7) Trabalhe em equipe para o ICPC!

- a) Treine codificar com papel e lápis! Isso é importante para quando seu colega está usando o computador e você quer adiantar seu algoritmo de algum outro problema.
- b) Use a estratégia "envie e imprima" para o debug, e deixe seu colega usar o computador enquanto você debuga.
- c) Se o seu colega está codificando e você não está com idéias para fazer nenhum outro problema, prepare casos de teste para o programa de seu colega. Ajude seu colega a ver se o código está correto.
- d) Mapeie os pontos fortes de cada um e passe o problema para quem é mais forte, principalmente no início.
- e) Pratique programação em duplas ou trio para a fase final!



# Vai encarar? Então assine o contrato de compromisso!

Se você chegou até aqui e não desistiu, e REALMENTE quer participar, assine o contrato de compromisso agora! Resumo:

- a) Você concorda em ser TORTURADO durante 15 semanas, incluindo todos os sábados com possibilidade de aulas extras a serem marcadas?
- b) Você irá programar em C++/C, Python ou Java (ou se compromete a aprender essas linguagens)?
- c) Você se esforçará para aprender estruturas de dados e algoritmos avançados até chorar lágrimas de sangue?
- d) Você entende que seu estilo de codificação será bastante ESTRAGADO durante esse tempo? (o estrago é reversível)
- e) Você concorda com as políticas de EXPULSÃO dos times e dos treinamentos?



# Dúvidas?



Foto de Towfiqu Barbuiya, no Unsplash (<https://unsplash.com/photos/a-blue-question-mark-on-a-pink-background-oZuBNC-6E2s>)

