



Lições de  
Programação

# **LIÇÕES DE PROGRAMAÇÃO**

## **RESOLUÇÃO DE PROBLEMAS**



GILSON PEREIRA DO CARMO FILHO



# **LIÇÕES DE PROGRAMAÇÃO RESOLUÇÃO DE PROBLEMAS**

Para adquirir as competências básicas em programação, você precisará praticar bastante, e a melhor forma de fazer isso é resolvendo uma grande quantidade de problemas.

Existem algumas ferramentas que são bastante apropriadas para isso, como por exemplo, os juízes online.

Este material apresentará a você as fases do processo de programação e mostrará como usar uma plataforma de juiz online, para que você possa praticar e desenvolver plenamente as suas habilidades em programação.



## **GILSON FILHO**

Cientista da Computação e mestre em Engenharia de Teleinformática pela Universidade Federal do Ceará. Especialista em Gerenciamento de Projetos pela Fundação Getúlio Vargas. Possui larga experiência profissional na área de TI, com ênfase em Engenharia de Software, Bancos de Dados e Sistemas de Informação. Foi professor na Universidade Federal do Ceará. Atualmente é professor na Universidade de Fortaleza e consultor nas áreas de educação e tecnologia.



# + SUMÁRIO

Introdução	<b><u>05</u></b>
O processo de programação	<b><u>11</u></b>
A ferramenta beecrowd	<b><u>19</u></b>
Conclusão	<b><u>36</u></b>
Referências	<b><u>40</u></b>



1

# +INTRODUÇÃO

## A EXIGENTE ARTE DE PROGRAMAR

Devido à sua velocidade na manipulação de números, o computador é a ferramenta adequada para a resolução de problemas que envolvam cálculos difíceis ou repetitivos.

Se a realização de cálculos rápidos é um trabalho sob medida para os computadores, a tarefa mais racional de escrever os programas é qualidade da inteligência humana. O objetivo de um programador não é apenas resolver um problema, mas também criar instruções que façam o computador descobrir a solução.

No entanto, escrever um programa é uma tarefa muito rigorosa. Os computadores exigem precisão e abrangência totais em suas instruções: eles farão somente o que lhes for ordenado, e as instruções não podem ter a menor ambiguidade.

Assim, para desenvolver plenamente as suas habilidades em programação, será necessário praticar bastante, e a melhor forma de fazer isso é resolvendo uma grande quantidade de problemas.

## COMO PRATICAR PROGRAMAÇÃO

Para melhorar sua habilidade de programação e de resolução de problemas, você pode utilizar algumas ferramentas que são bastante apropriadas para isso, como por exemplo, os **juízes online**.

Também chamados de sistemas de avaliação automática, os juízes online disponibilizam problemas para serem resolvidos por meio da submissão de códigos-fonte, escritos em uma determinada linguagem de programação, e os corrigem automaticamente usando casos de teste.

Cada caso de teste possui um conjunto de entradas e suas respectivas saídas. Para verificar se um determinado código-fonte é uma solução correta, o juiz online o executa utilizando o conjunto de testes do respectivo problema.

Os juízes online são muito utilizados em competições de programação, como a [Maratona de Programação](#), organizada pela [Sociedade Brasileira de Computação \(SBC\)](#). Devido à funcionalidade de correção automatizada dos programas, esse tipo de ferramenta

vem sendo gradativamente adaptado para o ensino introdutório de programação.

Um grande benefício desses sistemas é que eles podem nos ajudar a escrever código otimizado e com mais qualidade. Isso é possível porque, quando você tem um problema e cria um algoritmo para resolvê-lo, com o tempo você percebe que existem diversas maneiras de chegar à solução, e então pode modificar o seu algoritmo para que ele seja executado mais rápido.

Para que você possa conhecer mais sobre esse tipo de ferramenta, alguns juízes online são apresentados a seguir:

- ❑ [UVa Online Judge](#): reúne um conjunto de problemas bem interessantes, dos mais variados assuntos. Também tem os problemas usados em maratonas de programação regionais e mundiais. As soluções dos problemas podem ser submetidas em C/C++, Java, Pascal ou Python.
- ❑ [Sphere Online Judge \(SPOJ\)](#): similar ao UVa e tem tradução para o Português.



- ❑ [Topcoder](#): também possui vários problemas interessantes. Algumas vezes tem campeonato de programação patrocinado por grandes empresas, como Google, Yahoo e NASA. Se você vence, além de uma graninha, pode até ganhar também um emprego nessas empresas.
- ❑ [Project Euler](#): problemas matemáticos que só precisam das respostas, mas você precisará criar um algoritmo para descobrir essa resposta.
- ❑ [CodingBat](#): voltado para iniciantes que queiram praticar Java e/ou Python.
- ❑ [Coderbyte](#): também possui uma série de problemas, e já oferece um ambiente para você codificar e testar o seu algoritmo antes de submeter.
- ❑ [CodeChef](#): semelhante ao SPOJ e UVa. As soluções dos problemas podem ser submetidas em várias linguagens: C/C++, Java, Python, Go, Lisp, PHP, entre outras.

- ❑ [HackerRank](#): as soluções podem ser submetidas em diversas linguagens.
- ❑ [beecrowd](#): tem problemas classificados por categoria e nível de dificuldade, e as soluções podem ser submetidas em mais de vinte linguagens. Disponível em Português, Inglês e Espanhol.
- ❑ [CodinGame](#): resolução de problemas por meio de quebra-cabeças e jogos. Suporta várias linguagens e é muito interessante.
- ❑ [Neps Academy](#): semelhante ao beecrowd. As soluções podem ser submetidas em C/C++, Java, JavaScript, Python ou Portugol.

Nos próximos capítulos, apresentaremos as fases do processo de programação e mostraremos como usar a ferramenta **beecrowd**, para que você possa praticar e desenvolver suas habilidades em programação.

Boa leitura!



**2**

**+0**

# **PROCESSO DE PROGRAMAÇÃO**

A programação é indiscutivelmente uma atividade complexa, combinando muitos processos mentais. Para um iniciante em programação, uma das dificuldades naturais é como começar a desenvolver um programa para resolver um determinado problema.

Antes de escrever um programa, é necessário um processo de raciocínio, partindo de uma análise do problema dado, passando por um algoritmo bem abstrato, e chegando em um algoritmo detalhado, composto por uma sequência de passos simples. Esses passos, por sua vez, podem ser traduzidos diretamente para uma linguagem de programação, e então testados.

Portanto, de um modo geral, podemos observar quatro fases durante o processo de programação, conforme apresentadas a seguir:

1. Análise do problema
2. Desenvolvimento da solução
3. Implementação da solução
4. Testes



Figura 1: Fases do processo de programação

## ANÁLISE DO PROBLEMA

Na primeira fase, o programador obtém uma completa compreensão do problema, antes de pensar numa solução. É um processo de natureza altamente cognitiva, no qual existe um aprofundamento no estudo do problema.

Nesta fase devemos identificar dois tipos de dados: os que precisam ser fornecidos para resolvermos o problema (**dados de entrada**) e os que respondem ou resolvem o problema (**dados de saída**).

Para facilitar, você pode dividir a análise do problema em dois estágios:

1. **Familiarização:** Comece pelo enunciado do problema. Visualize o problema como um todo, com tanta clareza e nitidez quanto possível. É preciso compreender o problema, familiarizar-se com ele, gravar na mente o seu objetivo. A atenção concedida ao problema pode também estimular a memória e favorecer a recordação de pontos relevantes.
2. **Aperfeiçoamento:** Quando o enunciado do problema estiver bem claro e gravado em sua mente, verifique os dados de entrada e de saída, considerando-os um a um. Em seguida, examine-os em várias combinações, relacionando cada um dos detalhes entre si e com a totalidade do problema. Deve-se preparar e clarificar os detalhes que mais tarde terão uma função a desempenhar.

Infelizmente, muitos programadores passam rapidamente por esta fase e, assim, as especificações podem ser mal-entendidas ou mal interpretadas.

## DESENVOLVIMENTO DA SOLUÇÃO

Na segunda fase, o programador deve determinar o que é preciso para **transformar os dados de entrada em dados de saída**. Este é um processo predominantemente criativo, no qual é construído um algoritmo, isto é, uma sequência de passos que levam à solução do problema.

Uma estratégia bastante usada nesta fase é a técnica de refinamentos sucessivos, cujo objetivo básico é a decomposição de um problema grande numa série de subproblemas mais simples até chegar a um nível onde pode-se tentar uma solução.

Nessa técnica, o desenvolvimento da solução é iniciado com uma proposta muito geral ou abstrata da solução, expressa em termos do próprio problema. Passa-se então a refinar esta solução, elaborando detalhes previamente ignorados, resultado numa nova solução consideravelmente menos abstrata. Esse processo continua por meio de estágios de refinamento até chegar em um nível apropriado de detalhe, obtendo-se então o algoritmo.

## IMPLEMENTAÇÃO DA SOLUÇÃO

Na terceira fase, a solução desenvolvida na fase anterior é codificada numa linguagem de programação. Se a solução estiver bem definida, este processo será altamente mecânico.

Teoricamente, a implementação da solução pode ser realizada em qualquer linguagem de programação. No entanto, devido a necessidades específicas de programação, é bem mais fácil criar um programa para uma determinada finalidade usando uma linguagem própria para este fim.

Na escolha da linguagem mais apropriada para a implementação, vários fatores devem ser levados em conta, tais como a rapidez de execução e a facilidade de manutenção, isto é, a facilidade com que o programa pode ser alterado após entrar em operação.

Infelizmente, existe uma tendência muito grande dos programadores passarem para a fase de implementação da solução antes do problema ter sido completamente resolvido, levando a problemas maiores posteriormente.



## TESTES

Na quarta e última fase, o programador precisa comprovar que o programa implementado está correto. Teoricamente, a única maneira de se fazer isso é testando todos os casos prováveis, isto é, todas as combinações possíveis dos dados de entrada (teste exaustivo). Esta é uma situação tecnicamente inviável, mesmo para um programa simples, pois os testes poderiam durar muito tempo, talvez vários dias.

O número de casos a serem testados deve então ser reduzido em relação aos exigidos num teste exaustivo. Com empenho e atenção, o programador pode eliminar muitos casos desnecessários, e um conjunto razoável de dados de teste pode ser preparado.

É importante ressaltar que os testes nunca mostram que um programa não contém erros. No máximo, eles mostram que você não encontrou nenhum erro. Um teste bem-sucedido apenas significa que não foram descobertos erros dentro das condições particulares testadas, e nada pode ser afirmado em relação a outras condições.

Portanto, a escolha apropriada de casos de teste é parte essencial de qualquer projeto de programação. Um bom teste procura antecipar possíveis problemas e forçar os erros a se revelarem, ajudando assim a melhorar nossa confiança no programa.



3

+

A

**FERRAMENTA  
BEECROWD**

O [beecrowd](#) é uma plataforma online que visa auxiliar professores e alunos no processo de ensino e aprendizagem de Algoritmos e Programação. Ela oferece todas as funcionalidades básicas e necessárias a um portal de programação, tais como: correção da submissão em tempo real, utilização de juízes online especiais, alta disponibilidade e fórum de discussão.

Além disso, a ferramenta também permite a aceitação de soluções em diferentes linguagens de programação, a visualização do código-fonte enviado e a apresentação detalhada dos seus erros, caso sejam encontrados.

Desenvolvida pela [Universidade Regional Integrada do Alto Uruguai e das Missões](#) (URI), e anteriormente chamada de URI Online Judge, a plataforma beecrowd possui um repositório com mais de dois mil problemas, classificados em nove categorias e dez níveis de dificuldade.

A ferramenta possui toda a sua interface disponível nos seguintes idiomas: Português, Inglês e Espanhol.

## MÓDULO ACADEMIC

A plataforma beecrowd possui um módulo denominado **Academic**, no qual um professor pode criar disciplinas e elaborar listas de exercícios, com prazos determinados e referentes a cada conteúdo abordado na disciplina. Além disso, o professor também pode monitorar o desempenho individual dos alunos e visualizar o histórico de submissões de cada estudante para cada problema.

Como a correção dos exercícios é realizada de forma automática, o auxílio da ferramenta possibilita que o professor tenha uma visão abrangente dos temas de maior dificuldade e um maior enfoque no ensino de conteúdos técnicos, como algoritmos e linguagens de programação, bem como na interpretação dos enunciados dos problemas propostos.

Por meio do ambiente, também é possível ao professor analisar o desempenho geral, determinando o compasso das aulas, adiantando ou permanecendo no conteúdo previsto no plano de ensino.

A visualização do módulo pelo aluno é feita por meio da opção "ACADEMIC", localizada no menu do beecrowd. As listas de exercícios estarão disponíveis na disciplina criada pelo seu professor.

Há duas formas de se acessar uma disciplina:

- ❑ Aceitando o convite do professor, enviado por e-mail pela ferramenta;
- ❑ Informando o ID e a chave (*key*) da disciplina fornecidos pelo seu professor. Para isso, clique no botão "ACESSAR DISCIPLINA", localizada na região superior direita da página.

As listas de exercícios, visíveis apenas aos alunos convidados pelo professor, apresentarão as seguintes informações, dentre outras: identificação do problema, número de submissões, nível de dificuldade, orientações passadas pelo professor da disciplina e prazo de conclusão, com contagem regressiva do tempo.

Mostraremos a seguir como utilizar o beecrowd para resolver problemas de programação e o que acontece quando você submete a sua solução.

## RESOLVENDO PROBLEMAS

No beecrowd, você vai encontrar mais de dois mil problemas disponíveis para praticar importantes conceitos de programação, dos fundamentais aos mais avançados.

A plataforma adota uma padronização na formatação dos problemas, usando o mesmo estilo do [International Collegiate Programming Contest \(ICPC\)](#), uma competição anual de programação entre universidades do mundo todo, organizada pela [Association for Computing Machinery \(ACM\)](#).


Para resolver um problema, você deve ler e entender o seu enunciado, planejar a solução, escrever o código e, finalmente, submetê-lo para correção. Em seguida, você pode verificar se a sua solução funcionou como esperado ou se falhou de alguma forma.

Conforme apresentada nas Figuras 2 e 3, a tela de descrição do problema e envio da solução é composta por oito elementos, os quais são descritos a seguir.

1

beecrowd | 1001

**Extremamente Básico**

Adaptado por Neilor Tonin, URI  Brasil

Timelimit: 1

2

Leia 2 valores inteiros e armazene-os nas variáveis **A** e **B**. Efetue a soma de **A** e **B** atribuindo o seu resultado na variável **X**. Imprima **X** conforme exemplo apresentado abaixo. Não apresente mensagem alguma além daquilo que está sendo especificado e não esqueça de imprimir o fim de linha após o resultado, caso contrário, você receberá *"Presentation Error"*.

3

Entrada

A entrada contém 2 valores inteiros.

4

Saída

Imprima a mensagem "X = " (letra X maiúscula) seguido pelo valor da variável **X** e pelo final de linha. Cuide para que tenha um espaço antes e depois do sinal de igualdade, conforme o exemplo abaixo.

Exemplos de Entrada	Exemplos de Saída
10 9	X = 19
-10 4	X = -6
15 -7	X = 8

5

Figura 2: Exemplo de um problema de programação



## 1. Cabeçalho

Contém as seguintes informações sobre o problema: número de identificação, título, autor e limite de tempo (*timelimit*). No exemplo da Figura 2, vemos que o código executado não pode levar mais que 1 segundo para exibir o resultado.

## 2. Enunciado

Aqui é apresentado o problema a ser resolvido. Em alguns casos, o enunciado é bem simples e direto. Em outros, será necessário interpretar com muita atenção para compreender completamente o problema.

## 3. Entrada

Nessa parte serão descritas as entradas do problema, ou seja, os dados que serão fornecidos ao seu programa para resolver o problema. No exemplo da Figura 2, serão fornecidos dois valores inteiros.

## 4. Saída

Descreve as saídas do problema, ou seja, ele diz como o juiz online espera que o seu programa gere os dados que respondem ou resolvem o problema. É importante que você tenha muito cuidado com esse item, pois precisa segui-lo rigorosamente, não exibir-

do nem mais, nem menos do que foi pedido.

No exemplo da Figura 2, esse item diz que a saída deve ser a mensagem "X = " seguida do valor da variável **X**. Observe que a letra X na mensagem é maiúscula. Se ela for exibida em minúsculo, por exemplo, a resposta será dada como errada, já que não corresponde com o esperado.

## 5. Exemplos

Essa tabela apresenta um conjunto de casos de teste que podem ser usados para testar o seu programa. Para isso, verifique se, para uma determinada entrada, a saída correspondente é obtida.

Antes de submeter o seu código para julgamento, é sempre importante testar todos os casos de teste disponibilizados. Se qualquer um deles falhar, significa que o seu programa será rejeitado pelo juiz online.

Por outro lado, se o seu programa passar em todos os casos de teste do exemplo, ainda assim pode ser que ele não seja aceito, já que o juiz online possui vários outros casos bem mais minuciosos.

PROBLEMA

1001

LINGUAGEM

JavaScript

SOURCE CODE

```
1 var input = require('fs').readFileSync('/dev/stdin', 'utf8');
2 var lines = input.split('\n');
3
4 /**
5  * Escreva a sua solução aqui
6  * Code your solution here
7  * Escriba su solución aquí
8  */
9
```

CONSTRUA A SUA SOLUÇÃO E ENVIE!

ENVIAR

Figura 3: Escrevendo e testando a solução

## 6. Linguagem

Selecione aqui a linguagem de programação que você usou para escrever o seu programa.

## 7. Código fonte

Este item é um editor de texto no qual você poderá programar diretamente pelo navegador, porém esta não é a forma mais recomendada, especialmente se você for um iniciante.

A maneira mais adequada é utilizando o editor de texto ou IDE (*Integrated Development Environment*) de sua preferência. Depois de escrever o seu código, testá-lo com todos os exemplos disponibilizados e verificar se está tudo correto, copie-o e cole-o no editor, não esquecendo de confirmar se a linguagem selecionada é a correta.

## 8. Botão de submissão

Após colar o código fonte no editor de texto da ferramenta, clique no botão ENVIAR, e então começará a correção.

## O PROCESSO DE CORREÇÃO

O processo de correção consiste de vários casos de teste. Para cada um deles, os dados de entrada são passados para o seu programa, e os resultados gerados por ele são comparados com os dados de saída, conforme representado abaixo na Figura 4.

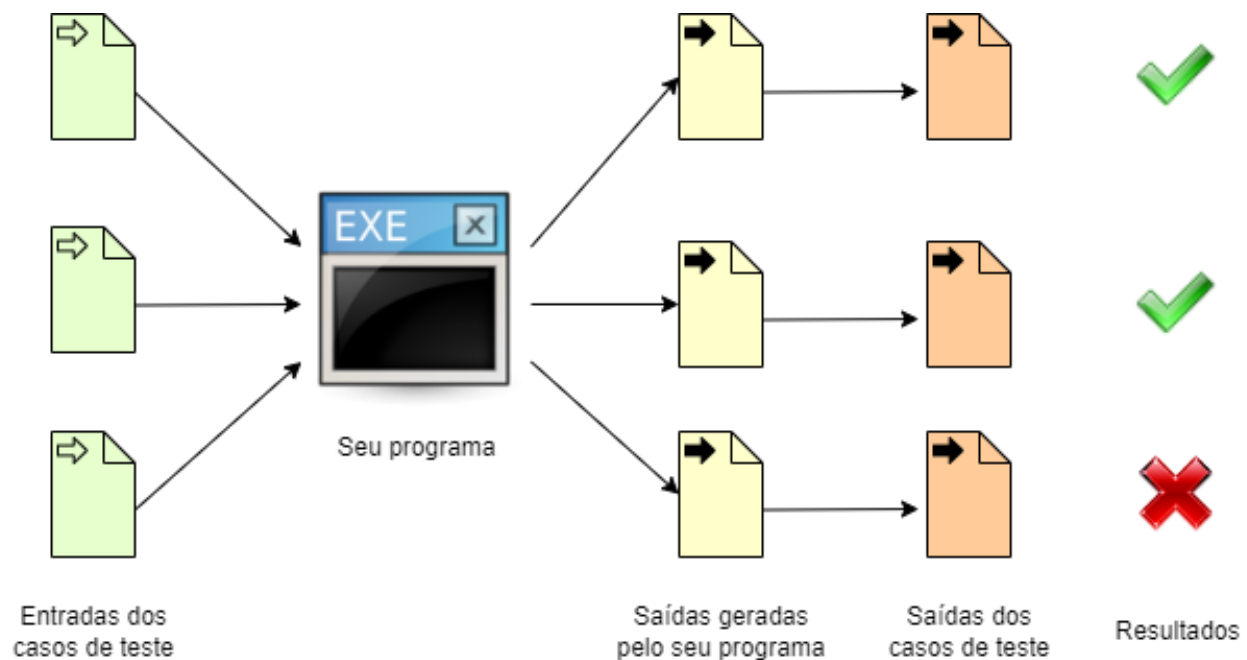


Figura 4: O processo de correção

Ao final da correção, você poderá obter um dos seguintes resultados:

- ❑ **Accepted:** solução aceita, ou seja, passou corretamente em todos os casos de teste.
- ❑ **Compilation error:** o seu programa apresentou algum erro de compilação. Isso acontece, por exemplo, quando falta algum parêntese ou chave no código-fonte enviado. Para um maior detalhamento do erro, basta atualizar a página pressionando F5. A linha com erro então aparecerá destacada na visualização do código-fonte.
- ❑ **Runtime error:** o seu programa apresentou algum erro durante a execução. Isso acontece, por exemplo, quando você define um *array* com menos capacidade do que o necessário para o problema, ou quando você tenta acessar uma área de memória inválida.
- ❑ **Time limit exceeded:** o seu programa demorou mais tempo do que o permitido para rodar todos os testes dos juízes. Se o *timelimit* for 2 segundos e, por exemplo, seu programa demorar 2,5 segundos para rodar todos os casos de teste, então este erro ocorrerá.

- ❑ **Presentation error:** a saída gerada pelo seu programa está com erro de apresentação, ou seja, ela está praticamente correta, mas há divergência na quantidade de espaços ou inversão de letras maiúsculas/minúsculas.
- ❑ **Wrong answer:** o seu programa não apresentou o resultado esperado para todos os casos de teste. Lembre-se de que o seu programa também é testado com outras entradas além das fornecidas na descrição do problema, porém sempre respeitando as restrições do mesmo. O valor mostrado junto à resposta indica o quanto a sua saída está diferente do esperado. Por exemplo, se sua submissão retornou **Wrong Answer** (90%), então saída está 90% diferente das saídas esperadas. Esse valor mostrado em porcentagem é sempre arredondado para a dezena superior, isto é, se a divergência foi de 3,5% você irá visualizar 10%.

Há ainda mais dois resultados além dos mencionados acima, que são usados em caso de problema na submissão. São eles:

- ❑ **Closed:** houve um problema com a sua submissão. Provavelmente o código-fonte não foi recebido, e por isso a submissão foi finalizada.
- ❑ **In queue:** a sua submissão está na fila para ser julgada.

## UDEBUG

É importante ressaltar que o processo de correção utiliza muito mais casos, e geralmente mais complexos, do que os apresentados no item 5 da Figura 2, justamente para testar se o seu programa é capaz de suportá-los. Assim, há alguns casos que poderão não estar evidentes nos exemplos.

Para ter acesso a mais casos de teste, além daqueles fornecidos como exemplo, você pode usar a plataforma [uDebug](#). Nela, uma comunidade entusiasmada de programadores ajudam uns aos outros respondendo perguntas no *chat*, dando dicas e soluções para problemas de vários juízes online, fornecendo casos de teste e compartilhando *feedback*.



No uDebug, você pode selecionar um problema para o qual codificou uma solução, indicar uma entrada e obter a saída “aceita”. Você pode então verificar se essa saída corresponde à saída do seu próprio programa.

Para utilizar esse recurso, clique no botão "UDEBUG" localizado no canto superior direito da página de exibição do problema, conforme mostrado na Figura 5 abaixo.

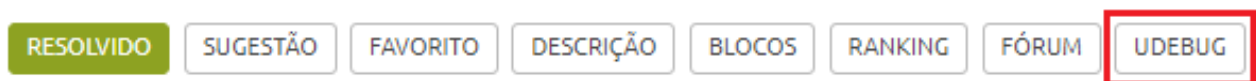


Figura 5: Acesso ao uDebug

Você será então direcionado para uma página no site da plataforma, contendo casos de teste específicos para o seu problema. A Figura 6 a seguir apresenta um exemplo dessa página.

[Problems](#)
[Contributors](#)
[API](#)
[Sign In](#)

Select Input (11)

Sign Up to Vote

Input

	User	Date	Votes	
1	URI Online Judge	27 Jul 2016 01:48:30	145	👍 🗑️ 🗑️
2	icarodantas123	29 Jul 2016 03:43:55	125	👍 🗑️ 🗑️
3	URI Online Judge	27 Jul 2016 01:49:53	124	👍 🗑️ 🗑️
4	Shah Shishir	20 Aug 2016 21:51:30	100	👍 🗑️ 🗑️
5	matheuslealv	08 Jun 2017 01:40:54	70	👍 🗑️ 🗑️
6	terencevitor	10 Aug 2017 00:55:01	65	👍 🗑️ 🗑️

Add Input

Delete Input

10 7

Copy Input

Get Accepted Output

Accepted Output

X = 17

Copy Output

Clear

Your Output

1. Run your code with the same input as above.  
 2. Paste your output here.  
 3. Press "Compare Outputs".

Clear

Compare Outputs

Figura 6: Tela do uDebug com casos de teste

Para usar um caso de teste, siga os passos abaixo:

1. Comece selecionando uma entrada na caixa "Select Input". A caixa "Input" será então preenchida com os dados da entrada selecionada.
2. Pressione o botão "Get Accepted Output" e observe que o campo "Accepted Output" será preenchido.
3. Execute o seu programa com os dados da entrada selecionada e então copie e cole sua saída na caixa "Your Output".
4. Pressione o botão "Compare Outputs" e verifique se essa saída corresponde à saída do seu programa.

Se a saída do seu programa for idêntica à saída aceita, parabéns! Caso contrário, isso é uma indicação de que algo em seu programa precisa ser corrigido.



4

+ **CONCLUSÃO**

## CONSIDERAÇÕES FINAIS

O processo de criação de um programa sempre inicia com a **compreensão de um problema**, isto é, o programador começa organizando os dados conhecidos e definindo precisamente o problema. Só então ele **elabora o algoritmo**, isto é, a sequência lógica de passos a serem seguidos para encontrar a solução.

Depois de construir o algoritmo, cada passo deve ser **transformado em instruções precisas**, de acordo com as regras da linguagem de programação escolhida.

Frequentemente, o programador usará casos extremos do problema para **verificar se a lógica do algoritmo se sustenta**. Esses testes muitas vezes revelam passos omitidos ou instruções ambíguas que provocam mensagens de erro.

À medida em que o programador vai adquirindo maior amadurecimento na construção e implementação de algoritmos, várias das fases descritas acima vão sendo feitas automaticamente.

Recomenda-se então que, no início do aprendizado, ou quando se tratar de problemas mais complexos, essas fases sejam seguidas formalmente, pois nenhum detalhe, por mais transparente que seja para a mente humana, pode ser omitido ou tomado como óbvio em programas de computador.

## **COMO CONTINUAR SEUS ESTUDOS**

Para adquirir as competências básicas em programação, é necessário que você se envolva em um nível significativamente alto de prática individual e experimentação.

Conhecer as fases e as estratégias de resolução de um problema ajuda a obter conhecimentos prévios para entender e resolver novos problemas. Assim, quanto mais você praticar, mais problemas conseguirá resolver.

Para agilizar o seu aprendizado e desenvolver a maturidade em programação, sugiro que você use alguma ferramenta do tipo juiz online, como beecrowd, SPOJ ou UVA, por exemplo.

E não se esqueça: um bom programador precisa ter **visão ampla**, para construir algoritmos eficientes, e **atenção rigorosa**, para transformar esse algoritmos em código de uma linguagem de programação, sem ambiguidades.

Bons estudos!



**5**

**+ REFERÊNCIAS**



ASCENCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da programação de computadores: algoritmos, PASCAL, C/C++ (padrão ANSI) e JAVA.** 3. ed. São Paulo: Pearson Education do Brasil, 2012.

ENTENDA o computador: as linguagens do computador. São Paulo: Nova Cultural, 1988.

GUIMARÃES, A. M.; LAGES, N. A. C. **Algoritmos e estruturas de dados.** Rio de Janeiro: Livros Técnicos e Científicos, 1985.

LOPES, A. F.; SANTANA, T. S.; BRAGA, A. H. **Um relato de experiência sobre o ensino de programação de computadores no Ensino Básico por meio da Olimpíada Brasileira de Informática.** In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 27., 2019, Belém. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 151-160.

PIVA JUNIOR, D. et al. **Algoritmos e programação de computadores.** 2. ed. Rio de Janeiro: Elsevier, 2019.

POLYA, G. **A arte de resolver problemas**: um novo aspecto do método matemático. Rio de Janeiro: Interciência, 1995.

SELIVON, M.; BEZERRA, J. L.; TONIN, N. A. URI **Online Judge Academic: Integração e Consolidação da Ferramenta no Processo de Ensino/Aprendizagem**. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 23., 2015, Recife. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2015. p. 188-195.

TREMBLAY, J. P.; BUNT, R. B. **Ciência dos computadores**: uma abordagem algorítmica. São Paulo: McGraw-Hill do Brasil, 1983.



# **LIÇÕES DE PROGRAMAÇÃO RESOLUÇÃO DE PROBLEMAS**

Copyright © 2022 Systema Educação e Tecnologia

Todos os direitos reservados. Este e-book ou qualquer parte dele não pode ser reproduzido ou usado de forma alguma sem autorização expressa, por escrito, do autor ou editor, exceto pelo uso de citações breves em uma resenha do e-book.



**Systema**

EDUCAÇÃO E TECNOLOGIA