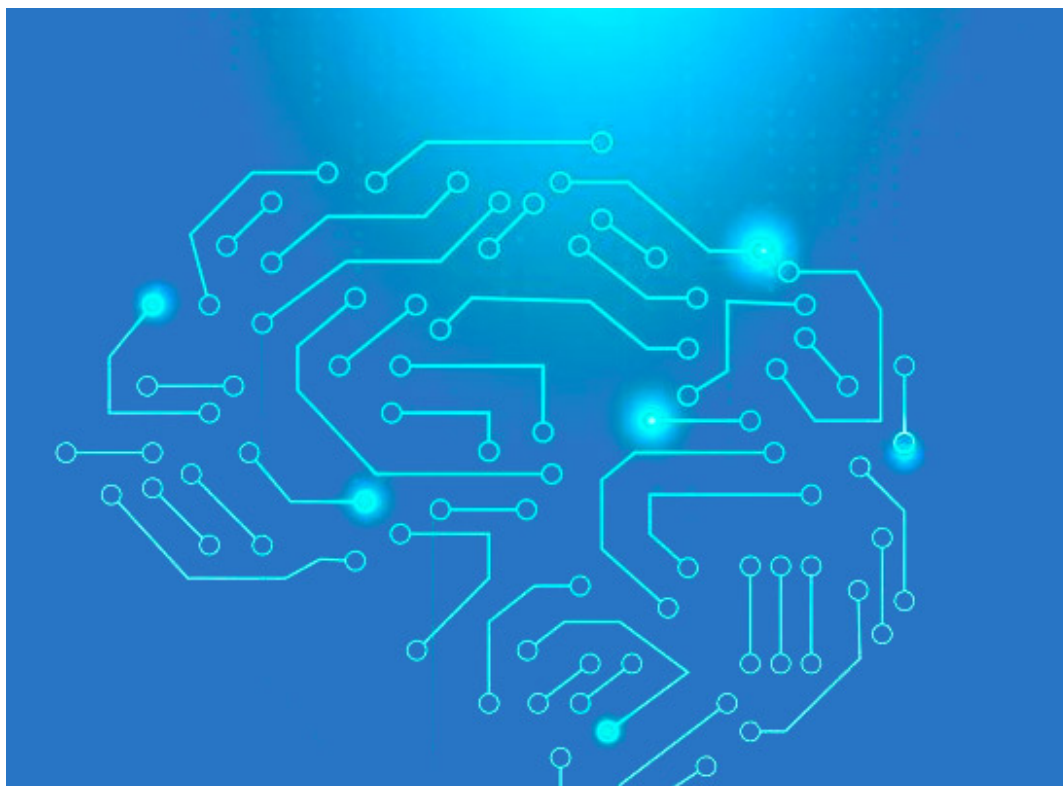


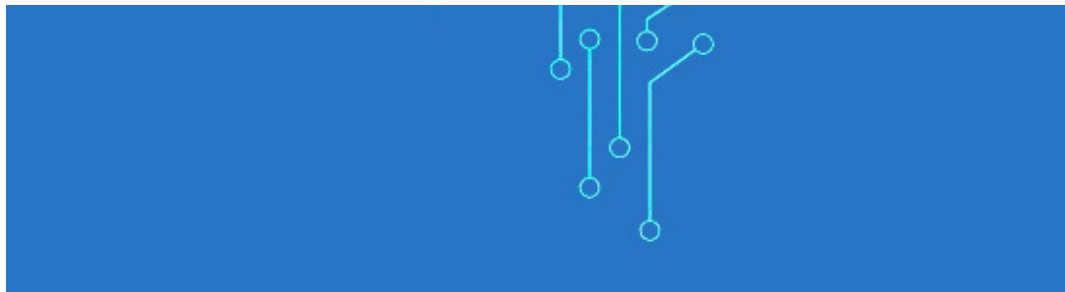


Desenvolvimento de Sistemas

Como sugestão, configure a opção de leitura de caracteres e de pontuação de seu leitor de tela para o grau máximo de leitura, para que os códigos disponibilizados neste material sejam lidos corretamente. No caso do NVDA (Nonvisual Desktop Access), para localizar a opção Grau de pontuação/símbolos, acesse Preferências – Configurações – Fala. Altere o padrão Pouco para Tudo. Dessa forma, você passará a ler os segmentos de código em sua totalidade.

Pensamento computacional: decomposição, reconhecimento de padrões, abstração e algoritmos





O que é pensamento computacional?

Todos sabem que os computadores interpretam toda e qualquer instrução dada de modo literal. Se ele receber o comando de desligar, ele desligará; se for pedido para ele abrir um programa, ele abrirá, e assim é desde a percepção de uma linguagem humano-computador. Porém, antes disso tudo, antes até mesmo da programação de uma linha de código para que as máquinas possam interpretar, existe todo um desenvolvimento. Com isso em mente, é possível começar a pensar um passo antes, por meio do **pensamento computacional** ou, se preferir, **PC**.

Pensamento computacional é uma habilidade fundamental para todos, não apenas para programadores. Com o advento tecnológico crescendo exponencialmente todos os anos, é fácil as pessoas aplicarem a tecnologia no seu dia a dia, inclusive isso acontece quase que inconscientemente. Então, se forem aplicadas essas habilidades de **quebrar problemas, reconhecer padrões, abstrair esses problemas e desenhar os passos e as regras para solucioná-los**, torna-se muito mais simples ir para o próximo passo mais tranquilo e com mais clareza.

Portanto, qual é o próximo passo? **Programar!**

Imagine o pensamento computacional como uma maneira de facilitar o entendimento dos códigos que serão desenvolvidos. O primeiro passo, que são as lógicas proposicional e booleana, ajudam você a entrar no **pensamento computacional** e ir programando a sua mente, preparando-a para interpretar algoritmos, códigos e problemas. O conteúdo

sobre **programação de computadores** desta unidade curricular traz informações interessantes para que você entenda melhor sobre o assunto.

Observe o diagrama a seguir:

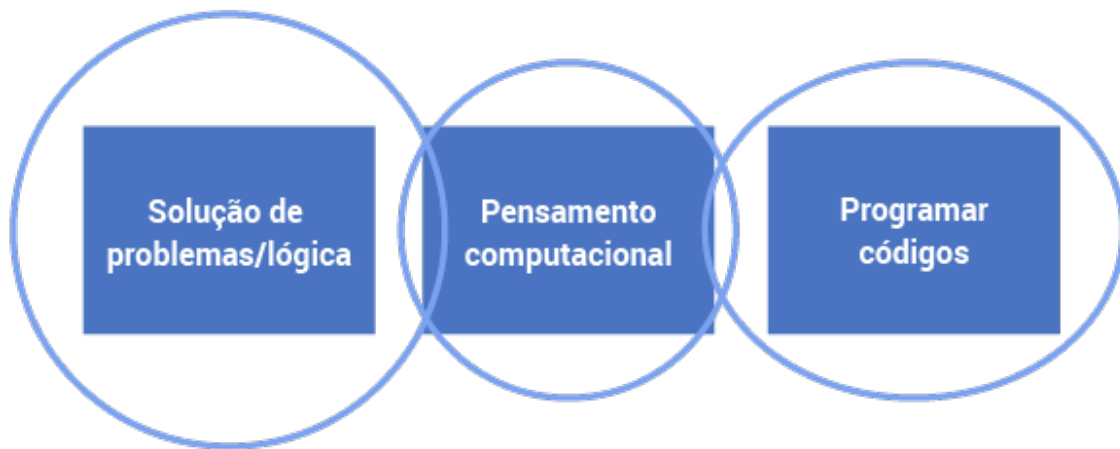



Figura 1 – Diagrama mostrando a hierarquia de informações para o entendimento de lógica e a tecnologia da informação

Fonte: autor

A imagem mostra três círculos sobrepostos. Eles estão alinhados na horizontal: o círculo da esquerda é o maior, dentro dele está escrito “Solução de problemas/lógica”; ao lado direito dele há um círculo menor, no qual está contido o texto “Pensamento computacional”; à direita e no último círculo está escrito o texto “Programar códigos”.

Mas é claro que, assim como o pensamento computacional ajuda você a quebrar os tão temidos algoritmos, ele pode também ser separado em pequenos passos para que facilite o entendimento de como ele funciona. Esses passos são:

- ◆ Decomposição
-  ◆ Reconhecimento de padrões
- ◆ Abstração
- ◆ Algoritmo

Pense em algumas atividades rotineiras e comece a aplicar o **PC**:

Escovar os dentes:

- a. Pegar a escova de dentes
- b. Pegar a pasta de dentes
- c. Abrir a torneira
- d. Limpar a escova
- e. Escovar os dentes
- f. Enxaguar a boca
- g. Guardar a escova de dentes
- h. Guardar a pasta de dentes
- i. Fechar a torneira

Essa é uma atividade simples, e são todos esses passos executados no dia a dia que podem sim ser transformados em uma série de padrões e regras, para que se expanda muito mais o pensamento computacional.

O que se fez anteriormente chama-se **decomposição**. Isto é, escolhe-se uma atividade simples e a separa em “n” passos.

Agora, e se for preciso usar o passo de **reconhecer padrões** para uma atividade similar? Pense em lavar um prato. É necessário pegar o prato, a esponja e o detergente

(escova e pasta), abrir a torneira (mesmo padrão), esfregar o prato (escovar os dentes), lavar o prato para remover o sabão (enxaguar a boca), colocar o prato no secador de louças (guardar os objetos) e fechar a torneira (percebem o padrão?). Tudo isso pode ser aplicado para qualquer atividade, seja ela simples ou complexa!

Para **abstrair** é muito simples: quais são as questões essenciais e quais são as secundárias para tais atividades? Comece pelas essenciais.

Questões essenciais:

- ◆ Escova e pasta de dentes/prato, esponja e detergente
- ◆ Água/água
- ◆ Enxaguar/lavar
- ◆ Fechar/fechar

Questão secundária:

- ◆ Guardar/guardar

Pense em um aplicativo ou *software* desenvolvido. Independentemente da quantidade de funcionalidades e estruturas que ele tiver, sem dúvidas haverá as **funções** essenciais e que dão toda a funcionalidade do produto desenvolvido, e haverá as **funções** mais adicionais, que não interferem na funcionalidade principal, mas criam um senso de que o aplicativo pode muito mais.

Para finalizar, tem-se os **algoritmos**. Já foram definidos os passos e as regras e compreendidos os padrões para atividades similares, além de já ter sido estabelecido o que é mais importante e menos importante a ser implementado inicialmente. Agora, imagine, literalmente, o passo a passo para isso, para que se comece a documentar e, em seguida, coloque-se em código!

Refleta quão mais fácil é a vida de um desenvolvedor quando todo esse **pensamento** foi desenvolvido e arquitetado antes mesmo de uma linha de código ser escrita. A ideia é justamente tornar o **pensamento computacional** uma atividade rotineira, com isso facilitando a produção e consequentemente a finalização de um produto, seja este simples ou complexo.

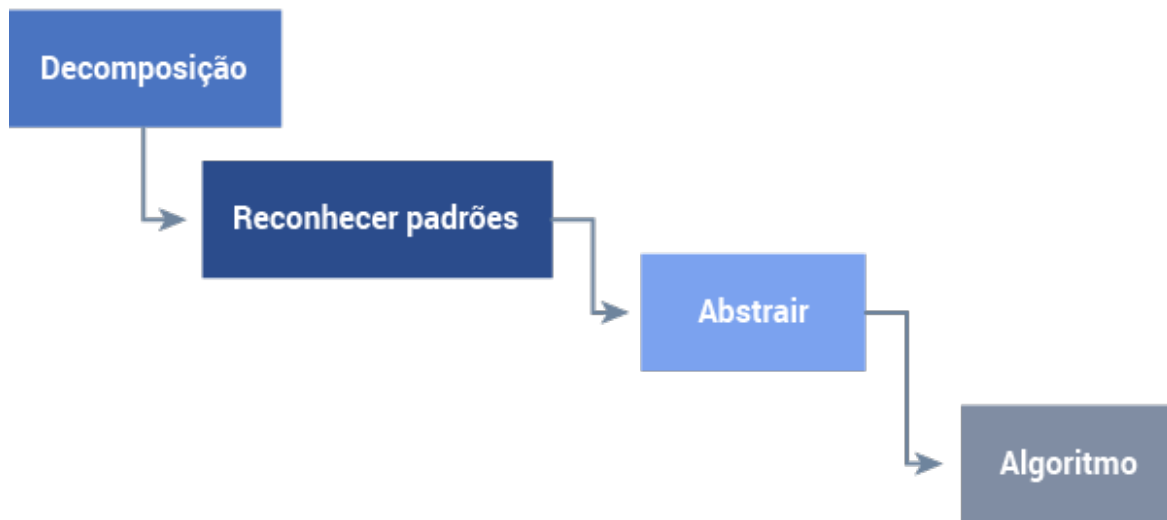


Figura 2 – Hierarquia do estudo do pensamento computacional

Fonte: autor

A imagem mostra quatro retângulos, um abaixo do outro em uma diagonal. O primeiro retângulo está posicionado à esquerda, no canto superior, com o texto “Decomposição”; abaixo deste, no segundo retângulo, está o texto “Reconhecer padrões”; no terceiro retângulo, o texto contido é “Abstrair”, por fim, no canto inferior esquerdo, o retângulo final está escrita a palavra “Algoritmo”.

Com isso em mente, é possível começar a expandir esses **quatro pilares** de modo que se tornará mais fácil compreender e aplicar esse processo dentro dos algoritmos, o que você verá dentro e fora do curso. Que tal dissecar cada um dos pilares, então, em ordem, para que você entenda como aplicá-los?

1. Decomposição



O primeiro e talvez mais importante dos pilares do **pensamento computacional** é basicamente considerar um problema (neste caso, um algoritmo) complexo e quebrá-lo em pequenas partes que se tornarão, conseqüentemente, mais fáceis e mais simples de se entender. Resolvendo essas pequenas partes, passo a passo, será possível chegar na solução do problema maior.

Se um problema não é decomposto, sem dúvidas ele ficará mais complexo de se resolver, ainda mais se tratando da quantidade quase infinita de problemas que a tecnologia da informação apresenta. Veja uma exemplificação:

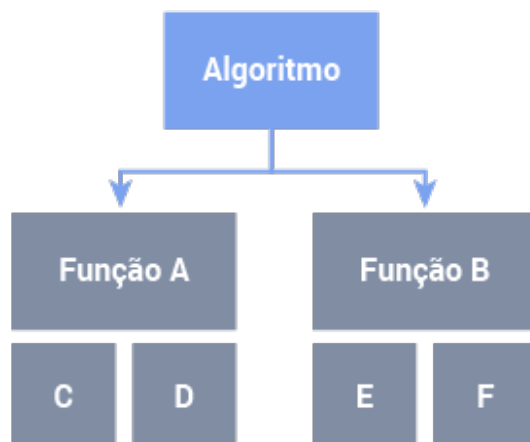


Figura 3 – Exemplo de como a decomposição e a identificação de problemas podem ser feitas

Fonte: autor

A imagem mostra um retângulo na parte de cima com a palavra “Algoritmo”; abaixo e à esquerda está um retângulo com a frase “Função A” e, abaixo desse, dois pequenos quadrados com as letras “C” à esquerda e “D” à direita. Abaixo e à direita do primeiro retângulo está um retângulo menor com a frase “Função B” e dois pequenos quadrados com as letras “E” à esquerda e “F” à direita.

Já se observou a estrutura visual de um exemplo de decomposição, mas pense

agora em um problema mais prático.



Problema: a empresa XYZ contratou você para desenvolver um *software*/sistema que cadastre, em seu banco de dados, pessoas e seus *hobbies* pessoais. XYZ gostaria de fazer com que pessoas com *hobbies* em comum possam se comunicar e compartilhar sugestões e experiências por meio de um *chat* de texto ou até mesmo de grupos e fóruns que possam ser criados dentro desse sistema.

Esse sistema deverá conter informações pessoais do usuário, porém as únicas informações públicas serão o nome e os *hobbies*. O usuário também poderá editar essas informações, excluir seu perfil, recomendar para outros usuários, criar grupos e adicionar pessoas.

Bom, existe aqui o grande problema (o *software*/sistema em si) e vários pequenos problemas que podem ser decompostos por meio do grande (as funções solicitadas). Antes de se pensar nas funções menores, é preciso considerar algumas questões externas (que não envolvem código em si) nessa decomposição. Mas por quê? **Porque deve-se sempre lembrar de que a decomposição não necessariamente envolve a programação por si só, pode-se usar a decomposição em problemas diários, como os citados anteriormente.**

Confira alguns pequenos detalhes que devem ser considerados nessa decomposição antes do desenvolvimento:

- ◆ Como o aplicativo se apresentará?
- ◆ Quem é o público-alvo do seu aplicativo?
- ◆ Quais são os efeitos sonoros que o aplicativo terá?
- ◆ Quais são os *softwares* utilizados para o desenvolvimento?
- ◆ Como será a navegabilidade do aplicativo?
- ◆ Ele será gratuito ou pago?

Todas essas questões que foram **decompostas** não são relativas a códigos, mas, mesmo assim, estão dentro do contexto de desenvolvimento.

Veja agora a utilização do exemplo proposto como base da decomposição:



Figura 4 – Exemplo de decomposição com os problemas do desafio citado

Fonte: autor

A imagem mostra um retângulo na parte de cima com a palavra “Aplicação”; abaixo e à esquerda está um retângulo com a frase “Criar usuário” e abaixo desse, dois pequenos quadrados com “Dados pessoais” à esquerda e “Convidar contatos” à direita. Abaixo e à direita do primeiro retângulo está um retângulo menor com a frase “Criar grupo” e dois pequenos quadrados com as frases “Adicionar amigos” à esquerda e “Criar Publicações” à direita.

Note como apenas com uma simples decomposição os problemas são entendidos e diminuídos. Fazendo isso, é possível entender, programar, testar e compilar cada uma dessas funções antes de se ter o produto completo e ir validando essas etapas.

Desafio não avaliativo: a empresa ABC deseja a criação de um aplicativo que funcionará para que usuários possam adicionar seus filmes e séries assistidas, dar uma nota e criar uma resenha crítica sobre esses artigos. As críticas dos usuários serão públicas, assim como seus perfis. Dentro do perfil de cada usuário haverá um *ranking*, que será dado de

acordo com a quantidade de filmes/séries assistidos e críticas escritas. Os usuários poderão se seguir e mandar mensagens privadas entre si.

Perceba que nesse desafio, existem menos informações que no exemplo anterior. Então, use **decomposição** para separar o problema em pequenas etapas e conseguir um desenvolvimento mais completo e menos complexo.

2. Reconhecer padrões

Como o próprio nome diz, esse passo consiste em **reconhecer padrões por meio dos problemas decompostos**. Isto é, procure dentro dos pequenos problemas padrões, conexões, funcionalidades que são similares entre si, para que facilite a construção e resolução do problema maior.

Amplie o problema que já foi decomposto e procure padrões dentro de uma nova funcionalidade:

Novo problema: a empresa XYZ aprovou a primeira versão do seu aplicativo e agora gostaria de adicionar mais uma funcionalidade. Além dos *hobbies*, os usuários também poderão compartilhar os seus interesses de estudos, sejam eles formações acadêmicas ou desejos futuros.

Existem semelhanças e diferenças entre essas duas funcionalidades. Confira uma exemplificação disso no quadro a seguir:

Hobbies

- ◆ Atrelado a um usuário
- ◆ Público no perfil
- ◆ Necessário para cadastro completo
- ◆ Pode-se encontrar pessoas com *hobbies* semelhantes

Acadêmico



- ◆ Atrelado a um usuário
- ◆ Público no perfil
- ◆ Não é necessário para cadastro completo
- ◆ Pode-se encontrar pessoas com interesses semelhantes

Observando essa exemplificação existe a certeza de que a equipe de desenvolvimento já terá uma facilidade muito maior em criar essa nova funcionalidade, pois os padrões estão atrelados a um dos problemas maiores (o cadastro do usuário por si só).

Problemas são mais simples de se resolver quando eles **compartilham padrões**, já que se pode usar as mesmas técnicas de resolução tanto para um quanto para outro. Quanto mais padrões são encontrados, mais fácil e rápido uma tarefa será solucionada e seu fluxo de trabalho já foi documentado e aprendido para novos problemas que possam aparecer.

A empresa ABC aprovou seu sistema sobre séries e filmes e agora traz para a sua equipe um novo problema. É necessário que os usuários possam cadastrar os livros que já leram, dar uma nota e atribuir uma crítica para esses itens. O diferencial é que os livros devem conter uma foto para ilustrar sua capa.

Dica: para este desafio, primeiro **decomponha** essa nova funcionalidade e aplique o **reconhecimento de padrões** em cima dos filmes/séries.

3. Abstração

No **pensamento computacional**, a **abstração** auxilia a entender a complexidade

dos problemas e a encontrar o que é **mais relevante** e o que será **menos relevante** para a construção e resolução dos desafios propostos.

Já foi feita, então, a **decomposição** e também foram **reconhecidos os padrões** do problema principal. Agora, a abstração ajudará justamente a definir o que é necessário para que se possa atender ao que foi pedido primeiro. Depois disso, será possível trabalhar os problemas menores.

Observe novamente os padrões reconhecidos e o que foi pedido:

Problema: antes da entrega final, a empresa XYZ gostaria de ver uma versão beta do seu *software* para mostrar aos investidores. Defina então as funções principais e desenvolva esse pequeno protótipo.

É neste momento que a **abstração** começa a ser trabalhada. Existem os problemas decompostos e os padrões dessa decomposição já prontos, então agora, com a abstração, consegue-se ter um **modelo** do que realmente é essencial para essa construção. Veja uma exemplificação disso de modo mais prático.

Abstraindo as funcionalidades principais e relevantes:

Padrões gerais
O usuário precisa cadastrar seus <i>hobbies</i> .
O usuário pode convidar pessoas/amigos para a aplicação.
O usuário pode cadastrar seus interesses acadêmicos.

Abstraindo funcionalidades que não são tão relevantes:



Padrões específicos
Não é necessário saber quais são os <i>hobbies</i> do usuário.
Não é necessário saber quem será ou quantos serão convidados.
Não é necessário ser pública ou sequer ser preenchida a informação acadêmica.

Apenas com essas duas tabelas já se tem um modelo e uma ideia geral do que realmente é importante ao usuário no aplicativo e do que é necessário para suprir o pedido da empresa. A **abstração** é um passo importantíssimo para que se possa definir um modelo sobre os problemas que se tem para resolver. Esses problemas estão sempre sendo quebrados para que a equipe tenha facilidade e agilidade na sua resolução.

Que tal praticar um pouco mais isso?

A empresa ABC gostaria de ver seu sistema funcionando de uma forma básica. Então, para isso, defina quais são os **padrões gerais** e **padrões específicos** dos problemas dos quais você já fez a decomposição e já reconheceu os padrões e traga isso visualmente.

Dica: coloque lado a lado os padrões que já foram reconhecidos e crie uma tabela para cada função (filmes/séries e livros).

Com esses três passos, é possível expandir isso para o passo final e começar a dar uma solução para os problemas!

4. Algoritmo

Você chegou ao último pilar, que consequentemente juntará e organizará todas as informações que você já tem.

Antes de tudo, entenda o que é um algoritmo para o **pensamento computacional** e como isso impacta sua vida diariamente e também a tecnologia de informação.

Se você quiser que seu computador faça alguma coisa, você terá que escrever um *software* que diga ao computador, **passo a passo**, exatamente o que quer que ele faça e quando ele deve fazer. Para se chegar a esse passo a passo, é necessário planejá-lo de modo completo, para que a saída seja exatamente o que você quer.

Mas como fazer esse planejamento? A resposta é simples.

Você já está fazendo isto desde o início, quando começou a ter um problema e o quebrou em pequenas partes (**decomposição**), definiu o que é parecido dentro de cada pequena parte (**reconhecer padrões**) e por fim identificou o que é essencial e o que é extra no problema (**abstração**). Para um algoritmo, é preciso ter um ponto inicial, um ponto final e, entre esses pontos, um passo a passo, um conjunto de instruções bem claras e definidas. Dessa forma:

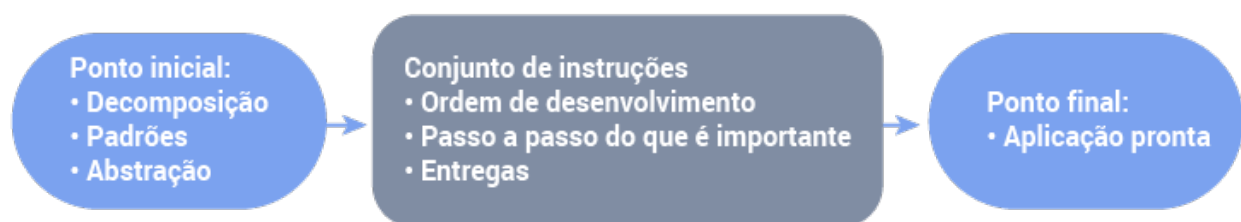


Figura 5 – Exemplo do fluxo do pensamento computacional até o algoritmo

Fonte: autor

Círculo à esquerda com as palavras em ordem: ponto inicial: decomposição, padrões, abstração. Seta à direita ligando com um retângulo as palavras: conjunto de instruções: ordem de desenvolvimento, passo a passo do que é importante, entregas. Por fim, mais uma seta para a direita ligando a um círculo com as palavras: ponto final: aplicação pronta.

Todos esses conjuntos de instruções serão os pequenos passos para definir a ordem de desenvolvimento. Observe a exemplificação, que traz o problema em questão:



- ◆ Instruções
- ◆ Usuário cadastra seu nome
- ◆ Usuário cadastra seu *e-mail*
- ◆ Usuário cadastra seus *hobbies*
- ◆ Usuário decide se quer colocar sua foto
- ◆ Perfil cadastrado?
- ◆ Sim:
 - ◆ Usuário pode convidar amigos
 - ◆ Usuário pode pesquisar pessoas com *hobbies* em comum
 - ◆ Usuário pode criar fóruns/grupos
 - ◆ Usuário pode entrar em fóruns/grupos
- ◆ Não:
 - ◆ Verifica se o *e-mail* digitado é válido ou já não está cadastrado
 - ◆ Retorna para tela de cadastro

Pode-se, também, transformar isso em um fluxograma, para que fique mais claro:



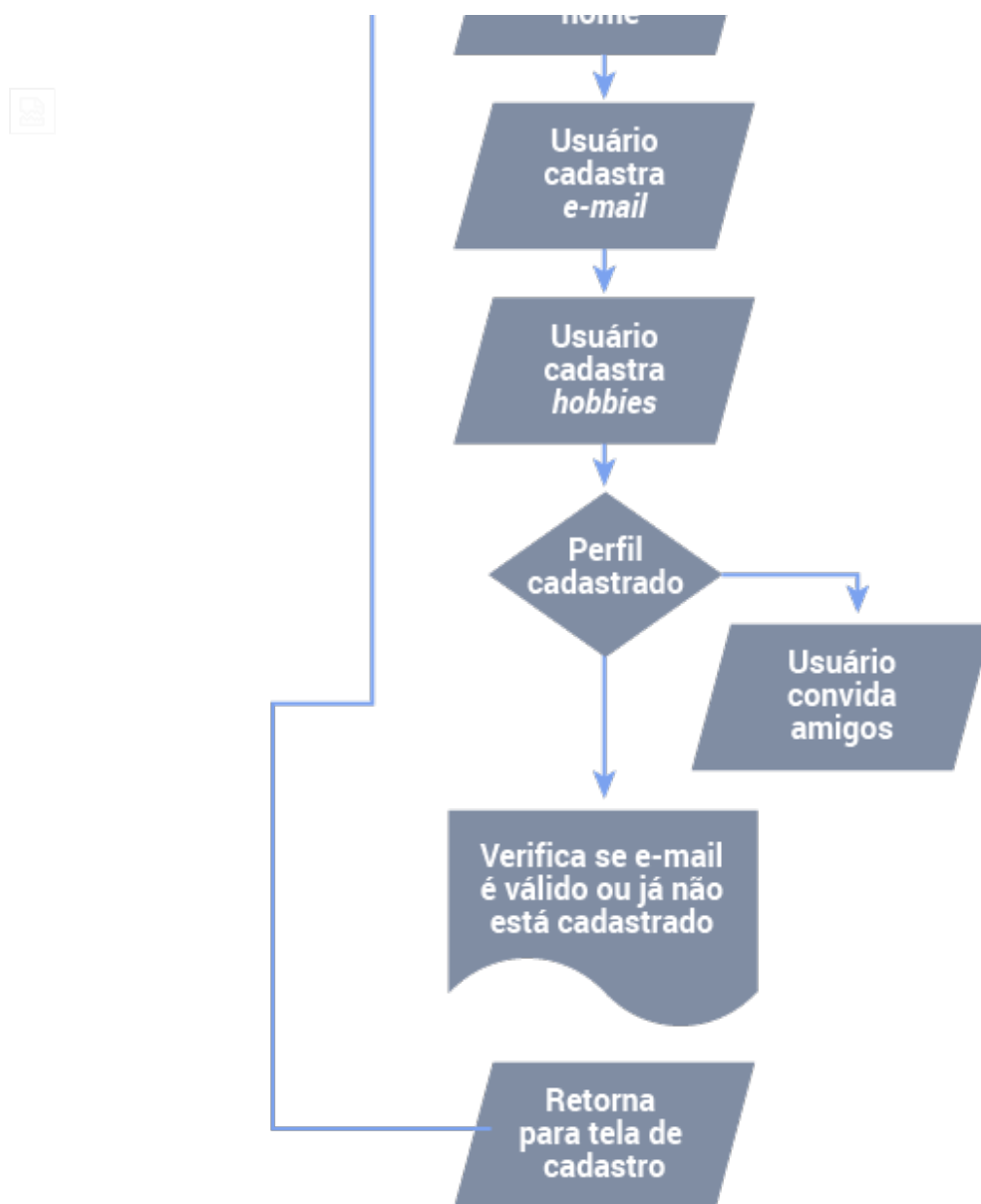


Figura 6 – Fluxograma demonstrando os passos que são possíveis dentro do desafio proposto

Fonte: autor

A imagem mostra um retângulo com a palavra início. A seguir, uma seta abaixo ligando a um paralelogramo com a frase Usuário cadastra nome. Outra seta abaixo ligando a um paralelogramo com a frase Usuário cadastra *hobbies*. Setas abaixo ligando a um losango com a frase Perfil cadastrado? Setas abaixo ligando a um retângulo com a

frase Verifica se e-mail é válido ou já não está cadastrado. Mais uma seta abaixo ligando a um paralelogramo com a frase Retorna para tela de cadastro. Por fim uma seta ligando ao retângulo do início novamente.

Tudo isso (e muito mais) compõe e ajuda a entender a composição de um **algoritmo**. Todos os passos dados anteriormente ajudaram a entender e contribuíram para a quebra de todos esses problemas.

Que tal agora finalizar o desafio utilizando as regras de **algoritmos**? Você pode, se achar necessário, utilizar **fluxogramas** para trazer esse **passo a passo** de maneira mais visual e de mais fácil entendimento. Pense que isso não é uma organização apenas para si, e sim para uma equipe que pode tranquilamente utilizar todos esses passos para construir não só este, mas qualquer outro problema.

Nota: a construção de algoritmos será abordada com mais profundidade na unidade curricular sobre o desenvolvimento de algoritmos deste curso.

O que você aprendeu sobre pensamento computacional?





É notório que o **pensamento computacional** é uma das técnicas vitais para se ter um melhor entendimento de vários processos e como esses processos podem impactar tanto o seu dia a dia quanto o desenvolvimento de um sistema.

Lembre-se de que esses quatro pilares estudados conversam diretamente entre si, e cada um dos passos ajudará você a ter um produto mais coeso e com muito menos alterações e correções de erros durante e até mesmo depois de sua finalização e entrega.

Decomponha os problemas, **reconheça os padrões** dentro de tudo que foi pedido, **abstraia** as informações relevantes e importantes e por fim desenvolva e imagine como os seus futuros usuários e clientes verão o seu **algoritmo** transformado em um sistema funcional.

Para praticar:

Dentro de todos os pilares aprendidos, pode-se também estudar algumas técnicas por meio de pequenos *sites* e/ou aplicativos. Observe esta lista:

LightBot

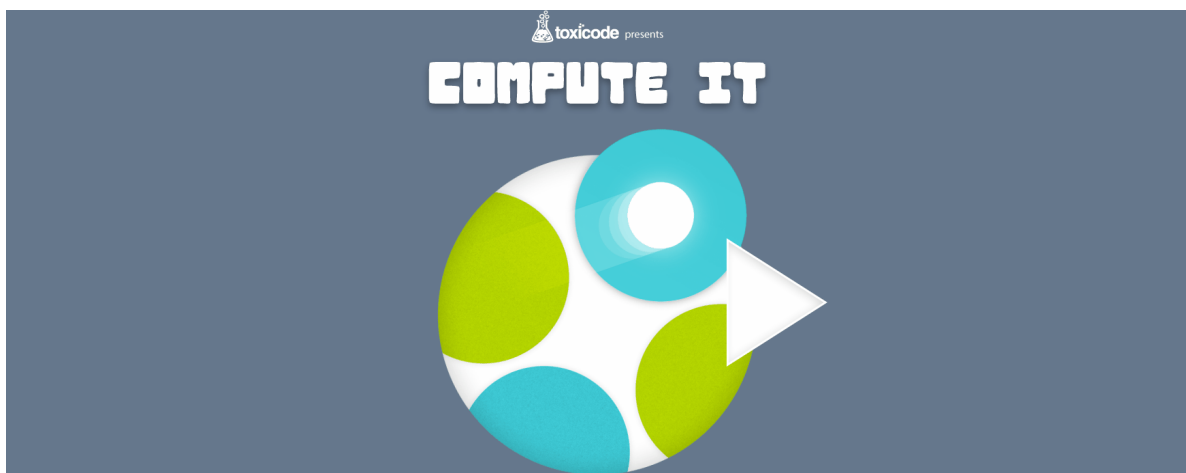


Objetiva praticar algoritmos e suas devidas instruções (aplicativo disponível para Android e iOS).



Toxicode

Utilize os passos para chegar à resolução. Cada nível aumenta a dificuldade.



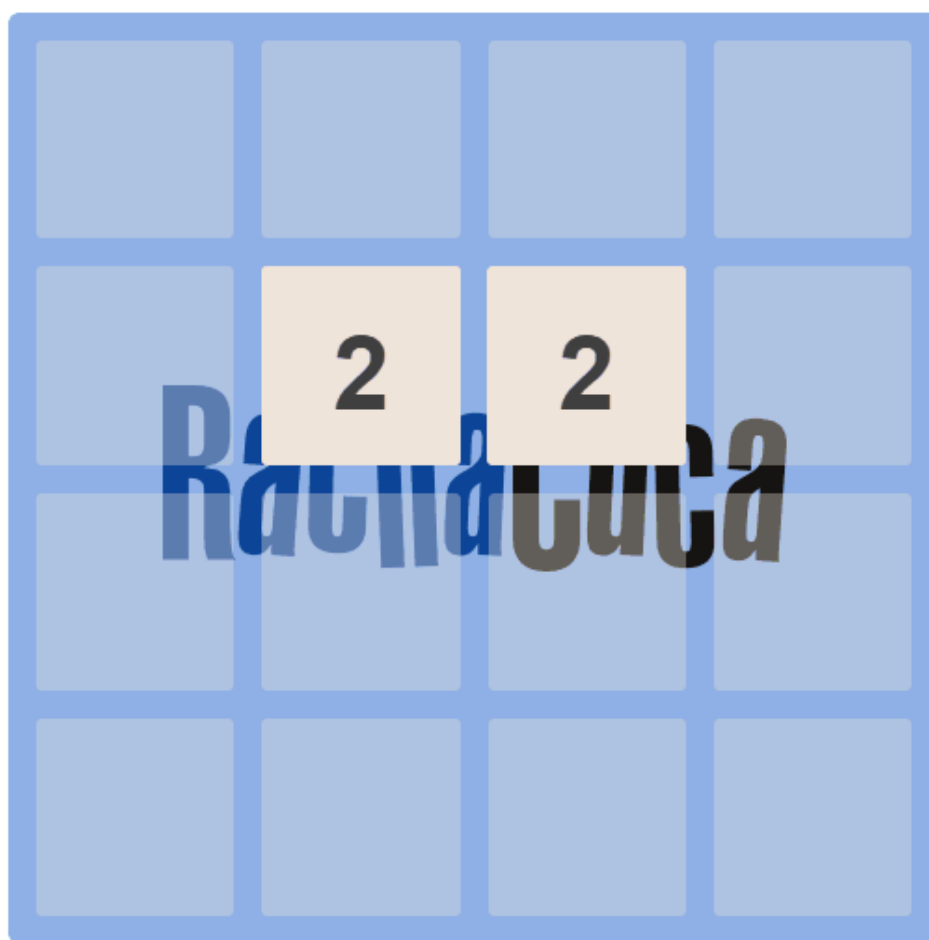
2048



Reconheça os padrões e conecte os blocos para chegar ao maior número possível.

Pontuação: 0 Recorde: 0

Novo Jogo



Jogue no Geniol: [Caça Palavras](#), [Dominó](#) e [Quebra-Cabeças](#).

Travessia

O jogo fornece uma série de instruções. Decomponha-a, reconheça os padrões possíveis e atinja o objetivo.

Travessia do Rio

f Compartilhar 9



Jogo Travessia do Rio

Teste seu raciocínio lógico com este teste divertido. Seu objetivo é fazer todo o mundo atravessar esse rio. Mas a jangada só permite levar duas pessoas. No entanto, o Pai não pode ficar com as filhas, a Mamãe não pode ficar com os filhos e o Criminoso não pode ficar com ninguém, excepto o Policial. Apenas a Mãe, o Pai e o Policial conseguem dirigir a jangada. Resolva o enigma.

Instruções do Jogo

Mouse: interagir com o jogo

LightBot

Objetiva praticar algoritmos e suas devidas instruções (aplicativo disponível para Android e iOS).

Toxicode

Utilize os passos para chegar à resolução. Cada nível aumenta a dificuldade.



2048

Reconheça os padrões e conecte os blocos para chegar ao maior número possível.

Travessia

O jogo fornece uma série de instruções. Decomponha-a, reconheça os padrões possíveis e atinja o objetivo.