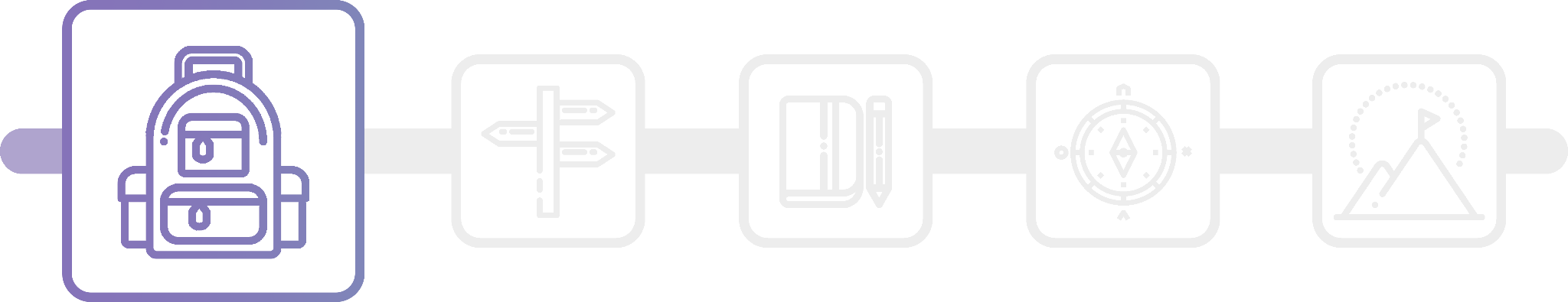
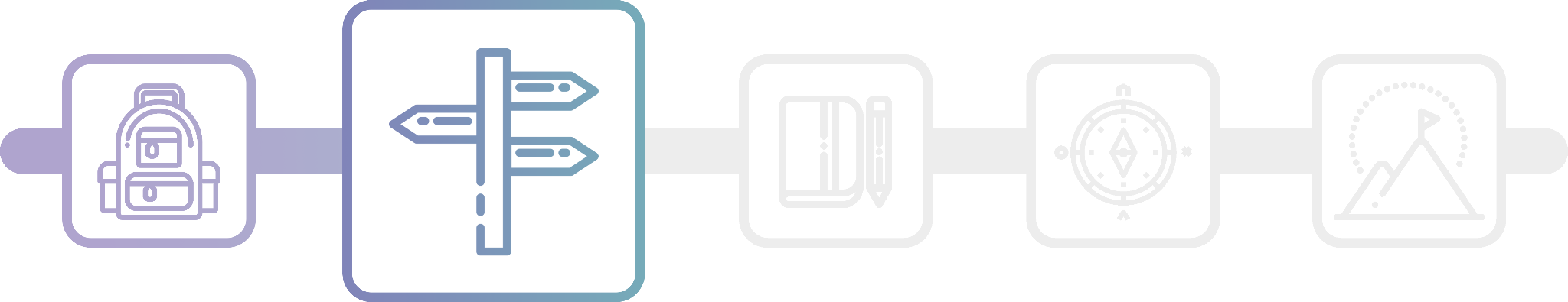
## Apresentação



Na aula sobre pesquisa binária e pesquisa sequencial, o objetivo é entender como esses dois tipos de busca funcionam e suas principais diferenças. A pesquisa binária é mais rápida, mas só pode ser usada em listas organizadas, e a pesquisa sequencial é mais lenta, mas funciona em qualquer tipo de lista. Saber escolher qual usar é importante para melhorar a eficiência dos programas que criamos.

Agora, reflita: Quando você usaria a pesquisa binária e quando a pesquisa sequencial seria mais adequada?

## Etapa Introdução aos estudos



### Estruturas de dados

Na aula sobre pesquisa binária e pesquisa sequencial, o desafio central é compreender como essas duas técnicas de busca se inserem no contexto das estruturas de dados e quais são suas características essenciais. Para explorar isso, nas próximas etapas do módulo, discutiremos as funcionalidades, vantagens e desvantagens de cada método, bem como sua aplicação em diferentes tipos de estruturas de dados.

Ao aprofundar-se nos conteúdos, será fundamental mobilizar os conhecimentos adquiridos para entender como a pesquisa binária, que depende de uma estrutura de dados ordenada, oferece uma busca eficiente com complexidade O(log n). Por outro lado, a pesquisa sequencial, que percorre os elementos um a um, tem uma complexidade O(n), sendo adequada para conjuntos de dados não ordenados ou menores. Esse entendimento é crucial para reconhecer a importância dos Tipos Abstratos de Dados (TAD), já que esses encapsulam as estruturas de dados com operações específicas de busca e manipulação, promovendo modularidade e abstração no desenvolvimento de algoritmos.

Na Etapa | Resposta ao Desafio, você deverá refletir sobre a influência das estruturas de dados e dos métodos de busca no desenvolvimento de algoritmos e no desempenho dos sistemas. Será importante destacar como a escolha adequada entre pesquisa binária e sequencial pode otimizar a eficiência de um algoritmo, dependendo da estrutura de dados em questão, como arrays ordenados ou listas não ordenadas. Utilize exemplos práticos para ilustrar como a pesquisa binária em um array ordenado pode ser significativamente mais rápida do que a pesquisa sequencial, enquanto a pesquisa sequencial pode ser mais simples e útil em cenários onde a ordenação não está disponível.

### Pesquisa Binária

Prós:

* Rápida: Funciona em tempo O(log n), o que a torna muito eficiente para grandes listas.
* Eficiente: A cada passo, elimina metade dos elementos restantes, acelerando a busca.

Contras:

* Requer ordenação: Só pode ser usada em listas ou arrays que estejam organizados.
* Complexidade de implementação: Pode ser um pouco mais difícil de implementar do que a pesquisa sequencial.

### Pesquisa Sequencial

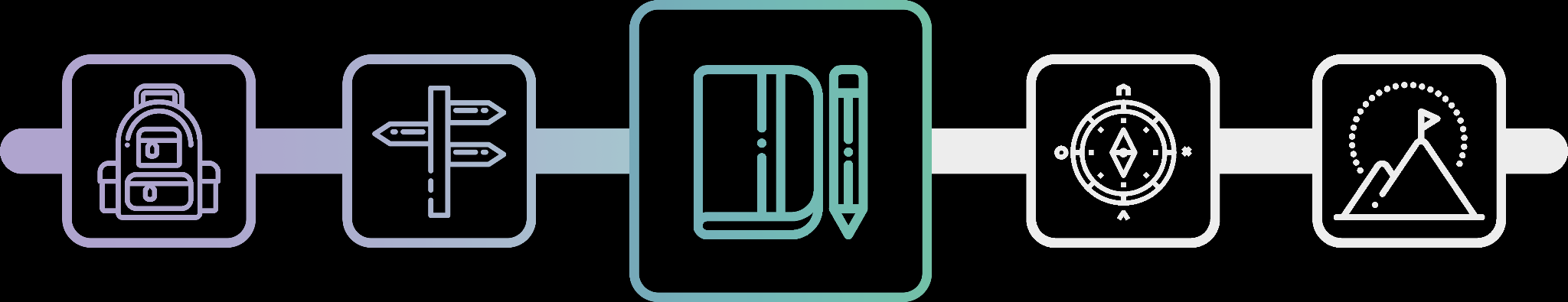
Prós:

* Simples: Fácil de entender e implementar.
* Versátil: Funciona em qualquer tipo de lista, ordenada ou não.

Contras:

* Lenta: Funciona em tempo O(n), o que a torna ineficiente para grandes listas.
* Pouco eficiente: Percorre todos os elementos até encontrar o que procura, o que pode demorar.

## Etapa Aprofundamento de estudos



### Pesquisa Sequencial

A pesquisa sequencial, também conhecida como linear, é uma técnica em que um array é percorrido item por item até que o elemento desejado seja encontrado ou até que todos os elementos tenham sido verificados. Esse método é amplamente utilizado em arrays não ordenados, mas também pode ser aplicado em arrays ordenados.

Vantagens:

* Simplicidade: É fácil de implementar e entender, sendo ideal para casos em que o conjunto de dados é pequeno ou não está ordenado.
* Versatilidade: Pode ser usado em qualquer tipo de estrutura de dados, sem a necessidade de ordenação prévia.

Desvantagens:

* Ineficiente para grandes volumes de dados: A complexidade é ONão, o que significa que, em grandes arrays, o tempo de execução pode se tornar muito lento.
* Comparações desnecessárias: Mesmo que o valor seja encontrado, o algoritmo ainda percorre todo o array em busca do elemento.

Exemplo de implementação:

*# Definição da função busca\_linear que recebe uma lista 'a' e uma 'chave' que se deseja encontrar na lista.*

def busca\_linear(a: list, chave: int) -> int:

*# O loop for percorre cada elemento da lista 'a'. A função enumerate retorna o índice 'i' e o 'elemento' da lista.*

for i, elemento in enumerate(a):

*# Verifica se o elemento atual é igual à chave procurada.*

if elemento == chave:

*# Se a chave for encontrada, retorna o índice onde ela se encontra.*

return i

*# Se o loop terminar e a chave não for encontrada, retorna -1.*

return -1

### Pesquisa Binária

A pesquisa binária é um método mais eficiente, mas só pode ser aplicada em arrays ordenados. O algoritmo funciona dividindo o array ao meio a cada iteração, eliminando metade dos elementos até encontrar o valor desejado.

Vantagens:

* Alta eficiência: Com complexidade O(log n), a pesquisa binária é muito rápida, especialmente em grandes conjuntos de dados.
* Menos comparações: Ao reduzir pela metade o número de elementos a cada iteração, a pesquisa binária é ideal para buscas em grandes arrays.

Desvantagens:

* Requer ordenação: Só funciona em arrays ou listas que já estão ordenados, o que pode exigir um passo adicional de ordenação, se o array não estiver organizado.
* Implementação mais complexa: Comparada à pesquisa sequencial, a pesquisa binária requer mais atenção para implementar corretamente.

Exemplo de implementação:

*# Definição da função busca\_binaria que recebe uma lista 'a' e uma 'chave' que se deseja encontrar na lista.*

def busca\_binaria(a: list, chave: int) -> int:

*# Inicializa o limite inferior do intervalo de busca como o primeiro índice da lista.*

inf = 0

*# Inicializa o limite superior do intervalo de busca como o último índice da lista.*

sup = len(a) - 1

*# Enquanto o limite inferior for menor ou igual ao limite superior, continua a busca.*

while inf <= sup:

*# Calcula o índice do meio entre os limites inferior e superior.*

middle = (inf + sup) // 2

*# Se a chave for igual ao elemento no índice do meio, retorna o índice como resultado.*

if chave == a[middle]:

return middle

*# Se a chave for menor que o elemento no índice do meio, ajusta o limite superior para um índice antes do meio.*

elif chave < a[middle]:

sup = middle - 1

*# Se a chave for maior que o elemento no índice do meio, ajusta o limite inferior para um índice após o meio.*

else:

inf = middle + 1

*# Se a busca terminar e a chave não for encontrada, retorna -1 indicando que a chave não está na lista.*

return -1

Reflexão

A pesquisa binária e a pesquisa sequencial são dois algoritmos fundamentais no estudo de estruturas de dados. É essencial que os programadores compreendam quando utilizar cada um para otimizar o desempenho dos seus programas. Em cenários onde o conjunto de dados é pequeno ou não ordenado, a pesquisa sequencial pode ser suficiente. No entanto, quando se trabalha com grandes quantidades de dados ordenados, a pesquisa binária é uma escolha muito mais eficiente.

### Explicando Pesquisa Sequencial e Binária

Pesquisa Sequencial: Imagine que você tem uma lista de amigos no seu jogo favorito, mas eles estão desorganizados, e você precisa encontrar o "João".

O jeito mais fácil é começar do primeiro nome e ir descendo, um por um, até achar o João.

Essa é a pesquisa sequencial. Você olha um de cada vez, sem pular ninguém.

Exemplo:

* Lista de personagens: Mario, Luigi, Peach, Bowser
* Vamos procurar o "Bowser". Começamos pelo Mario, depois olhamos Luigi, depois Peach, até finalmente achar o Bowser. Isso é pesquisa sequencial.

Pesquisa Binária: Agora, imagine que os personagens estão organizados em ordem alfabética.

Com isso, podemos ser mais espertos e, ao invés de começar do começo, a gente olha direto no meio da lista.

Se o nome que você procura for maior ou menor, você já sabe qual metade eliminar e continua.

Esse é o truque da pesquisa binária – mais rápida, mas precisa que a lista esteja organizada.

Exemplo:

* Lista ordenada de personagens: Bowser, Luigi, Mario, Peach
* Estamos procurando o "Mario". Vamos no meio e encontramos o "Luigi". Sabemos que o "Mario" está mais para a direita. Aí, pulamos para o meio dessa nova parte da lista e encontramos o Mario rapidamente!

Se você fosse procurar um personagem raro em um jogo, qual método usaria: ir um por um ou dividir a lista para achar mais rápido?