Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO

Lógica de programação e algoritmos

Prática de busca e ordenação

Aula 1 - Introdução à busca e ordenação

Código da aula: [DADOS]ANO1C3B4S25A1

Mapa da Unidade 1 Componente 4

Lógica de programação e algoritmos

semana 25

Você está aqui!

Prática de busca e ordenação

semana 27

Algoritmos de contagem e acumulação

semana 28

Pilhas e filas

Lógica de programação e algoritmos

Mapa da Unidade 1 Componente 4

Você está aqui!

25

Prática de busca e ordenação

Aula 1

Código da aula: [DADOS]ANO1C3B4S25A1

Objetivos da aula

- Introduzir e praticar conceitos sobre algoritmos de busca e ordenação.

Recursos didáticos

- Recurso audiovisual para exibição de vídeos e imagens;
- Acesso ao laboratório de informática e/ou internet.

Duração da aula

50 minutos.

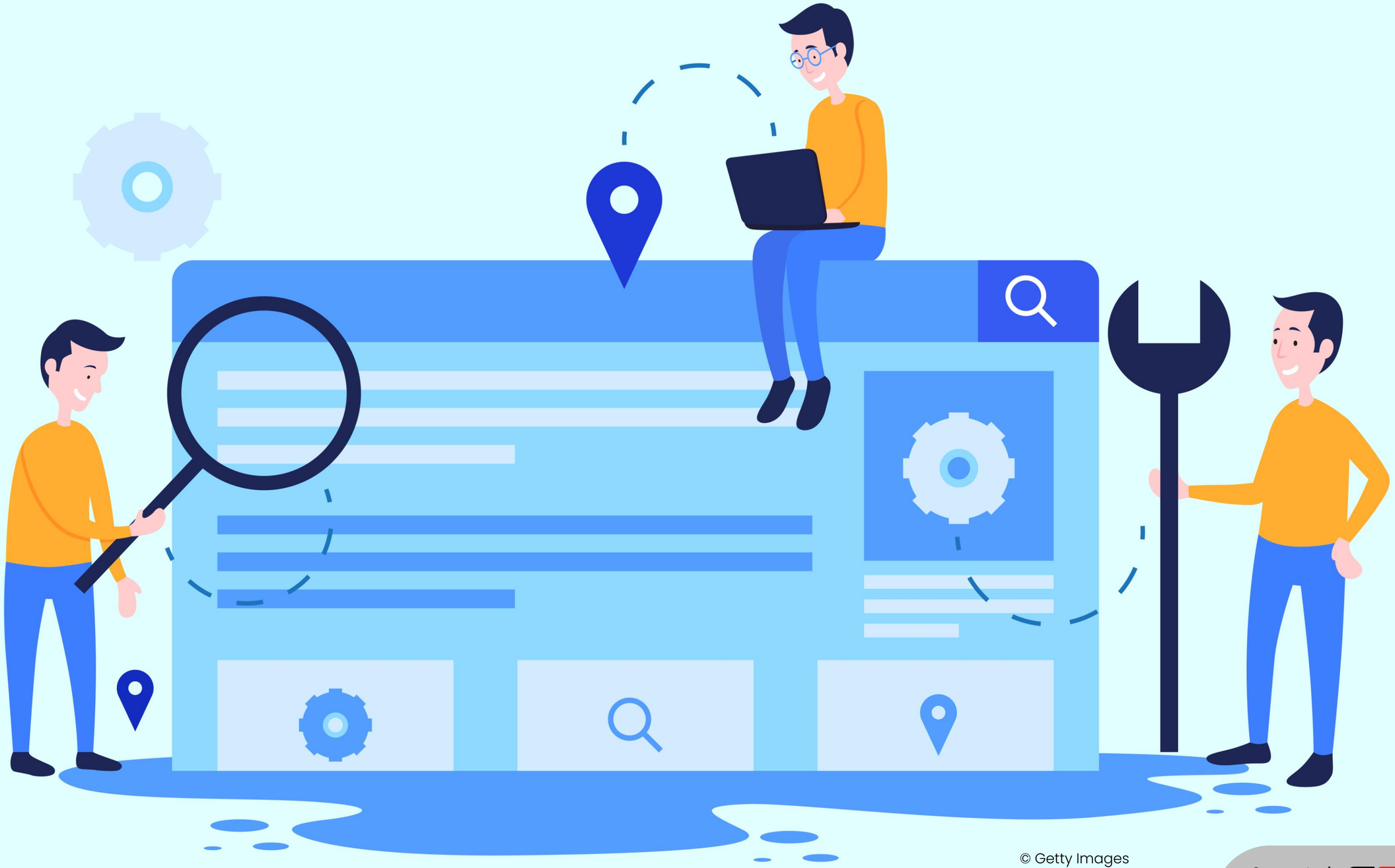
Competências técnicas

- Identificar e resolver problemas relacionados a dados e análises:
- Compreender e dominar técnicas de manipulação de dados.

Competências socioemocionais

- Adaptar-se a novas tecnologias, técnicas e tendências sem perder o foco, as metas e os objetivos da organização;
- Colaborar efetivamente com outros profissionais, como cientistas de dados e engenheiros de dados; trabalhar em equipes multifuncionais colaborando com colegas, gestores e clientes.

Diálogo



© Getty Images

Primeiras **ideias**

- O que é uma busca binária e por que ela é mais eficiente que a busca linear em listas ordenadas?
- Em quais situações a ordenação por inserção pode ser mais vantajosa do que a ordenação por seleção?
- Como a eficiência dos algoritmos de busca e ordenação impacta o desempenho geral de um software?

Ponto de **partida**

Imagine que você está desenvolvendo uma biblioteca digital onde os usuários podem buscar livros, artigos e revistas. A biblioteca possui uma coleção extensa e crescente de itens.

Nela, os usuários podem procurar por títulos específicos, autores ou palavras-chave, além de visualizar uma lista de itens ordenados por data de publicação, relevância ou popularidade.

Considerando que a eficiência dos algoritmos de busca e ordenação é essencial para fornecer respostas rápidas e precisas às consultas dos usuários e para manter a interface do usuário responsiva, reflita sobre:

1. Qual algoritmo de busca seria mais adequado para encontrar um livro específico, pelo título, em uma lista não ordenada?
2. Como a escolha do algoritmo de ordenação impacta a experiência do usuário ao visualizar uma lista de itens por data de publicação?

Situação fictícia elaborada especialmente para o curso.

Construindo o **conceito**

Conceito de algoritmos de busca

- São técnicas utilizadas para encontrar um elemento específico em uma estrutura de dados, como listas ou *arrays;*

- São essenciais para otimizar a recuperação de dados e garantir a eficiência de programas de computador.

Construindo o **conceito**

Busca linear

A busca linear é o método mais simples de busca. Ela percorre cada elemento da lista até encontrar o elemento desejado ou até chegar ao final da lista.

É fácil de implementar, mas pode ser ineficiente para listas longas, pois seu tempo de execução é proporcional ao tamanho da lista (**O(n)**).

Tome nota

A complexidade linear “O(n)” indica que o tempo de execução cresce de forma diretamente proporcional ao tamanho da entrada. Exemplo: busca linear.

Construindo o **conceito**

Busca binária

A busca binária é uma técnica de busca mais eficiente que a busca linear, mas **requer que a lista esteja ordenada**. Ela funciona dividindo a lista ao meio e comparando o elemento do meio com o elemento desejado.

Se o elemento desejado for menor, a busca continua na metade inferior; se for maior, continua na metade superior. Esse processo se repete até encontrar o elemento ou determinar que ele não está na lista. A busca binária tem um tempo de execução de **O(log n)**.

Tome nota

A complexidade logarítmica "**O(log n)**" indica que o tempo de execução cresce de forma logarítmica em relação ao tamanho da entrada. Exemplo: busca binária.

Construindo o **conceito**

Conceito de algoritmos de ordenação

Os algoritmos de ordenação são métodos para rearranjar os elementos de uma lista em uma determinada ordem, geralmente crescente ou decrescente.

Tome nota

A ordenação é um passo importante em muitas operações de busca e outras operações de processamento de dados, pois muitos algoritmos de busca e outras operações de processamento de dados são mais eficientes quando aplicados a listas ordenadas.

Construindo o **conceito**

Algoritmos de ordenação

Ordenação por seleção (*Selection Sort*)

- ✓ É um algoritmo simples que funciona encontrando repetidamente o menor (ou maior) elemento da lista e movendo-o para a posição correta. O processo é repetido até que toda a lista esteja ordenada;
- ✓ Não é eficiente para grandes listas, apesar de ser fácil de entender e implementar, pois tem um tempo de execução de $O(n^2)$.

Ordenação por inserção (*Insertion Sort*)

- ✓ É outro algoritmo simples que constrói a lista ordenada uma entrada por vez, removendo um elemento de entrada e encontrando a localização que ele pertence dentro da lista ordenada, e inserindo-o lá.
- ✓ É eficiente para pequenas listas ou listas que já estão parcialmente ordenadas, com um tempo de execução de $O(n^2)$ no pior caso, mas $O(n)$ no melhor caso.

Construindo o **conceito**

Exemplo 1: Busca linear

Observe que a busca linear percorre cada elemento da lista, um por um, até encontrar o elemento desejado ou até chegar ao final da lista.

```python
def busca_linear(lista, elemento):
    for i in range(len(lista)):
        if lista[i] == elemento:
            return i  # Elemento encontrado, retorna o índice
    return -1  # Elemento não encontrado

# Teste da busca linear
lista = ["maçã", "banana", "leite", "pão", "queijo"]
elemento = "pão"
indice = busca_linear(lista, elemento)
print(f"Elemento '{elemento}' encontrado na posição {indice}" if indice != -1 else "Elemento não encontrado")
```

Construindo o **conceito**

Exemplo 2: Busca binária

Observe que busca binária funciona dividindo a lista ao meio e comparando o elemento do meio com o elemento desejado. Requer que a lista esteja ordenada.

```python
def busca_binaria(lista, elemento):
    inicio = 0
    fim = len(lista) - 1

    while inicio <= fim:
        meio = (inicio + fim) // 2
        if lista[meio] == elemento:
            return meio  # Elemento encontrado, retorna o índice
        elif lista[meio] < elemento:
            inicio = meio + 1
        else:
            fim = meio - 1

    return -1  # Elemento não encontrado

# Teste da busca binária (a lista deve estar ordenada)
lista = [1, 3, 5, 7, 9, 11, 13]
elemento = 7
indice = busca_binaria(lista, elemento)
print(f"Elemento '{elemento}' encontrado na posição {indice}" if indice != -1 else "Elemento não encontrado")
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Construindo o **conceito**

Exemplo 3: Ordenação por seleção

Observe que a ordenação por seleção encontra repetidamente o menor elemento da lista e o move para a posição correta, repetindo o processo até que toda a lista esteja ordenada.

```
def ordenacao_por_selecao(lista):
    n = len(lista)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if lista[j] < lista[min_idx]:
                min_idx = j
        lista[i], lista[min_idx] = lista[min_idx], lista[i]

# Teste da ordenação por seleção
lista = [29, 10, 14, 37, 13]
ordenacao_por_selecao(lista)
print("Lista ordenada por seleção:", lista)
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Construindo o **conceito**

Exemplo 4: Ordenação por inserção

Observe que a ordenação por inserção constrói a lista ordenada uma entrada por vez, removendo um elemento de entrada e inserindo-o na posição correta dentro da lista ordenada.

```
def ordenacao_por_insercao(lista):
    for i in range(1, len(lista)):
        chave = lista[i]
        j = i - 1
        while j >= 0 and chave < lista[j]:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = chave

# Teste da ordenação por inserção
lista = [29, 10, 14, 37, 13]
ordenacao_por_insercao(lista)
print("Lista ordenada por inserção:", lista)
```

Elaborado especialmente para o curso com a ferramenta Jupyter Notebook.

Registro

Vamos fazer um **quiz**

Qual algoritmo de busca é mais eficiente para listas ordenadas?

- Busca linear
- Busca binária
- Busca sequencial
- Busca aleatória

Vamos fazer um **quiz**

Qual é o tempo de execução do algoritmo de ordenação por seleção no pior caso?

- O(n)
- O(log n)
- O(n log n)
- $O(n^2)$

Registro

Vamos fazer um **quiz**

Qual algoritmo de ordenação é eficiente para listas pequenas e quase ordenadas?

- *Selection Sort*
- *Merge Sort*
- *Insertion Sort*
- *Bubble Sort*

Diálogo

© Getty Images

O que nós **aprendemos hoje?**

Então ficamos assim...

O que nós **aprendemos hoje?**

Então ficamos assim...

1. Os algoritmos de busca são essenciais para encontrar elementos específicos em estruturas de dados. A busca linear verifica cada item sequencialmente, enquanto a busca binária é mais eficiente, dividindo a lista ordenada ao meio repetidamente.

2. Os algoritmos de ordenação reorganizam elementos em uma sequência ordenada. A ordenação por seleção encontra repetidamente o menor elemento e o move para a posição correta. A ordenação por inserção insere cada elemento na posição apropriada na lista ordenada.

3. Compreender e aplicar corretamente esses algoritmos melhora a eficiência e o desempenho de programas de computador, especialmente em grandes volumes de dados. As técnicas de busca e ordenação são fundamentais para otimizar a recuperação e organização de informações.

Saiba mais

Está construindo um sistema de busca e precisa encontrar tópicos que contenham palavras-chave específicas?

Neste artigo, você vai descobrir como usar listas invertidas para otimizar sua busca e garantir que os resultados relevantes apareçam para seus usuários.

MATHEUS, Y. *Python: procurando frases com listas invertidas*. Alura: 8 jan. 2020. Disponível em: https://www.alura.com.br/artigos/procurando-frases-com-listas-invertidas/. Acesso em: 11 jul. 2024.

Referências da aula

Identidade visual: Imagens © Getty Images.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. *Lógica de programação: a construção de algoritmos e estruturas de dados com aplicações em Python*. São Paulo: Pearson; Porto Alegre: Bookman, 2022.

Educação Profissional Paulista

Técnico em **Ciência de Dados**

Secretaria da Educação

SÃO PAULO
GOVERNO DO ESTADO