Matéria Adiantada dia 08/10

Semana 27 - Aula 1

Tópico Principal da Aula: Vetores e Matrizes **Subtítulo/Tema Específico:** Matrizes: Conceitos Fundamentais e Estrutura

Código da aula: [SIS]ANO1C1B4S27A1

Objetivos da Aula:

- Compreender os conceitos fundamentais de matrizes, aprofundando o que é, sua declaração, aplicações e desafios acerca de sua utilização.
- Desenvolver sistemas computacionais, utilizando ambiente de desenvolvimento.
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 08 - Matrizes: Conceito

- **Definição:** Uma **Matriz** é uma estrutura de dados fundamental, composta por um conjunto ordenado de elementos dispostos em uma grade retangular, organizada em **linhas** e **colunas**. Essa estrutura é a base para a representação de dados bidimensionais (e multidimensionais) na computação.
- **Aprofundamento/Complemento:** O número de linhas (m) e colunas (n) define a **dimensão** da matriz, representada por m×n. No contexto de programação, uma matriz é frequentemente implementada como um "array de arrays".

Exemplo Prático: A matriz que representa um placar de campeonato com 4 times (linhas) e 3 estatísticas (colunas: Vitórias, Derrotas, Empates).
Time A: [10, 2, 4]

Time B: [8, 5, 3]

Time C: [5, 5, 6]

Time D: [2, 12, 2]

- Isso é uma matriz 4×3.
- **Links de Vídeo:**

    - SOMATIZE PROF EDNA: Tipos de Matrizes. https://www.youtube.com/watch?app=desktop&v=XVpG0LHerNQ (Reforça os tipos matemáticos de matrizes).
    - Row-major vs. Column-major Order. https://www.youtube.com/watch?v=Fj2E67Kq2lY (Sugestão para aprofundar a seção de memória).

Referência do Slide: Slide 15 - Matrizes e Memória

- **Definição:** Em um sistema de memória linear (1D), as matrizes 2D precisam ser mapeadas. O **Row-major order** (ordem por linha) e o **Column-major order** (ordem por coluna) são os dois esquemas principais.
- **Aprofundamento/Complemento:** No **Row-major order** (usado por C e NumPy/Python), os elementos de uma mesma **linha** são alocados em posições de memória adjacentes. No **Column-major order** (usado por Fortran e MATLAB), os elementos de uma mesma **coluna** são adjacentes. O acesso a dados de forma contígua (seguindo a ordem de armazenamento) é mais rápido, pois aproveita a **localidade de cache**.
- **Exemplo Prático:** Ao iterar sobre uma matriz em Python (Row-major), um loop que percorre primeiro as linhas é geralmente mais eficiente do que um que percorre primeiro as colunas.

Semana 27 - Aula 2

Tópico Principal da Aula: Vetores e Matrizes **Subtítulo/Tema Específico:** Declaração, Inicialização e Flexibilidade em Linguagens de Programação **Código da aula:** [SIS]ANO1C1B4S27A2

Objetivos da Aula:

- Compreender as diferentes formas de declaração e inicialização de matrizes, além de sua sintaxe em diferentes linguagensANO1C1B4S27A2.pdf].
- Desenvolver sistemas computacionais, utilizando ambiente de desenvolvimentoANO1C1B4S27A2.pdf].
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazesANO1C1B4S27A2.pdf].

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 07 - Declaração e Inicialização de Matrizes em Python

- **Definição:** Em Python, a forma mais comum de representar matrizes é utilizando **listas aninhadas** (listas que contêm outras listas)ANO1C1B4S27A2.pdf]. A inicialização pode ser feita diretamente ou via *list comprehension*.
- **Aprofundamento/Complemento:** A utilização de bibliotecas como **NumPy** é a abordagem padrão em Ciência de Dados e Engenharia. Ela oferece o tipo `ndarray`, que armazena os elementos em um bloco contíguo de memória (diferente das listas padrão de Python) e é altamente otimizado para operações matriciais.

Exemplo Prático: Inicializar uma matriz 3×3 preenchida com zeros em Python (sem NumPy):
Python
Usando list comprehension

matriz_zeros = [[0] * 3 for _ in range(3)]

-
- **Links de Vídeo:**
  - CURSO EM VÍDEO. Exercício Python #086 – Matriz em Python. https://www.youtube.com/watch?v=EGmlFdwD4C4&t=63s (Focado na prática de matrizes em Python).
  - Matrizes Dinâmicas vs Estáticas em Java (Conceito). https://www.youtube.com/watch?v=7_r500LHCnE (Aprofundamento sobre a diferença estrutural em linguagens tipadas).

Referência do Slide: Slide 09 - Matrizes Dinâmicas vs Estáticas

- **Definição: Matrizes Estáticas** (ex: arrays em C/Java) possuem tamanho fixo definido em tempo de compilação, exigindo memória contígua. **Matrizes Dinâmicas** (ex: listas em Python) podem ter seu tamanho ajustado em tempo de execução, oferecendo maior flexibilidade.
- **Aprofundamento/Complemento:** Embora mais flexíveis, estruturas dinâmicas podem incorrer em **custos de desempenho** para gerenciar a realocação de memória quando o tamanho é alterado. Em contraste, estruturas estáticas são rápidas, mas demandam que o programador preveja o tamanho máximo necessário.
- **Exemplo Prático:** O problema de criar um sistema de cadastro sem saber o número exato de usuários. Uma estrutura dinâmica é a mais adequada, pois evita o desperdício de memória ou a falha por estouro de capacidade de um array estático.

Semana 27 - Aula 3

Tópico Principal da Aula: Vetores e Matrizes **Subtítulo/Tema Específico:** Aplicações de Matrizes em Algoritmos e Sistemas **Código da aula:** [SIS]ANO1C1B4S27A3

Objetivos da Aula:

- Conhecer a aplicação de matrizes e seu uso em algoritmos para **grafos**, para utilização em **processamento de imagens** e para **bases de dados**ANO1C1B4S27A3.pdf].
- Desenvolver sistemas computacionais, utilizando ambiente de desenvolvimentoANO1C1B4S27A3.pdf].
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazesANO1C1B4S27A3.pdf].

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 08 - Algoritmos e Matrizes (Grafos)

- **Definição:** Matrizes são usadas para representar a relação entre os nós (vértices) em um grafo. A **Matriz de Adjacência** é a mais comum: um valor 1 (ou o peso da aresta) na posição M[i][j] indica a existência de uma conexão do nó i para o nó.
- **Aprofundamento/Complemento:** Essa representação é essencial para a execução de algoritmos de busca (como **Busca em Largura - BFS** e **Busca em Profundidade - DFS**) e de cálculo de caminho mínimo (como **Dijkstra** ou **Floyd-Warshall**).
- **Exemplo Prático:** Em um mapa digital, a matriz de adjacência pode indicar quais cidades têm estradas de ligação direta e a distância entre elas.
- **Links de Vídeo:**
    - PRIME CURSOS DO BRASIL. Curso de Python 08 – Aprendendo sobre vetores. https://www.youtube.com/watch?v=7yBXNGVyN3Q&t=157s (Vídeo sugerido pelo material sobre vetores).
    - Convolução Imagens e Aplicações Filtros. https://www.youtube.com/watch?v=SsOHmcSVz0A (Aprofundamento sobre processamento de imagens).

Referência do Slide: Slide 11 - Processamento de Imagens

- **Definição:** Uma **imagem digital** é interpretada como uma matriz. Imagens em tons de cinza são matrizes 2D (onde cada elemento é a intensidade do pixel) e imagens coloridas (RGB) são **tensores** (matrizes 3D).
- **Aprofundamento/Complemento:** Filtros digitais (como detecção de bordas, desfoque ou *sharpening*) são aplicados usando a operação de **convolução**. Esta operação consiste em deslizar uma pequena matriz de pesos (*kernel*) sobre a matriz da imagem e calcular a soma ponderada dos pixels vizinhos.
- **Exemplo Prático:** Para detectar bordas, utiliza-se um *kernel* que realça as diferenças de intensidade entre os pixels, convertendo a imagem original na imagem filtrada.

Semana 27 - Aula 4

Tópico Principal da Aula: Vetores e Matrizes **Subtítulo/Tema Específico:** Complexidade, Desafios e Otimização de Matrizes **Código da aula:** [SIS]ANO1C1B4S27A4

Objetivos da Aula:

- Compreender os principais desafios e as soluções para trabalhar com matrizes, adequando sua complexidade ao contexto de desenvolvimentoANO1C1B4S27A4.pdf].
- Desenvolver sistemas computacionais, utilizando ambiente de desenvolvimentoANO1C1B4S27A4.pdf].
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazesANO1C1B4S27A4.pdf].

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 08 - Complexidade de Tempo e Espaço

- **Definição:** A **Complexidade de Tempo** mede o número de operações que um algoritmo executa em função do tamanho da entrada (matriz), utilizando a Notação O (Big O). A **Complexidade de Espaço** mede a quantidade de memória adicional consumidaANO1C1B4S27A4.pdf].
- **Aprofundamento/Complemento:** Em problemas com matrizes grandes, a complexidade é crítica. Por exemplo, a multiplicação de matrizes N×N utilizando o algoritmo clássico exige ≈N3 operações, resultando na complexidade O(N3). Um aumento pequeno no N gera um aumento exponencial no tempo de processamento.
- **Exemplo Prático:** Multiplicar matrizes 1000×1000 (1 bilhão de operações) é 8 vezes mais lento do que multiplicar matrizes 500×500 (125 milhões de operações), se considerarmos O(N3).
- **Links de Vídeo:**
    - SHARPAX. O que é ARRAY (Vetor – Matriz) – ENTENDA de uma vez por todas. https://www.youtube.com/watch?v=poDFFYkp6g4 (Foco no conceito de array/matriz).
    - Complexidade da Multiplicação de Matrizes. https://www.youtube.com/watch?v=jW7_Hn3E3QY (Sugestão para detalhar O(N3)).

Referência do Slide: Slide 11 - Otimização de Matrizes

- **Definição:** Otimização envolve a busca por algoritmos mais rápidos, como o de **Strassen** (O(N2.807)), e a utilização de ferramentas especializadas.
- **Aprofundamento/Complemento:** Bibliotecas como **NumPy** (Python) e **Eigen** (C++) são essenciais, pois implementam as operações matriciais utilizando código

otimizado em baixo nível (como rotinas Fortran ou C) e aproveitam recursos de hardware como paralelismo e instruções SIMD para executar operações muito mais rapidamente do que o código Python puro.

- **Exemplo Prático:** Para resolver um sistema de equações lineares complexo em um projeto, o desenvolvedor deve optar por usar uma função de biblioteca otimizada (ex: `numpy.linalg.solve`) em vez de tentar implementar o algoritmo de eliminação de Gauss do zero.