Algoritmos e Estruturas de Dados II

Capítulo da Aula 1: Apresentação e Visão Geral da Disciplina

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br
CEFET-MG - Campus Timóteo

4 de Março de 2026

0.1 1. O Legado de AED I (Revisão e Limitações)

Em **Algoritmos e Estruturas de Dados I**, o foco foi construir a base do pensamento computacional estruturado e aprender a dialogar com a máquina em um nível relativamente baixo: vetores, ponteiros/referências, laços, funções, recursão.

Lá, o mundo era predominantemente **Linear (1D)**:

- **Vetores (Arrays):** Acesso rápido por índice, tamanho fixo, excelente localidade de cache.
- **Listas Encadeadas:** Tamanho dinâmico, inserções e remoções locais baratas, porém acesso sequencial.
- **Pilhas (Stacks):** Disciplina LIFO (Last In, First Out) — diretamente ligada à pilha de chamadas de função e à recursão.
- **Filas (Queues):** Disciplina FIFO (First In, First Out) — essencial para modelar sistemas de espera e processamento em ordem de chegada.

Os problemas resolvidos eram, em sua maioria, **locais e diretos**: "Insira X", "Remova Y", "Ordene Z". A estrutura de dados era, em geral, escolhida de forma relativamente óbvia: se precisávamos de acesso rápido, vetor; se o tamanho era desconhecido, lista. A complexidade raramente explodia a ponto de inviabilizar a solução em máquinas modernas.

• Teoria

Do ponto de vista de engenharia, AED I ensinou a **pensar em termos de mecanismos de armazenamento**: como alocar e liberar memória, como percorrer sequências. AED II, por outro lado, ensina a **pensar em termos de estrutura da informação**. Não se trata apenas de onde guardar os dados, mas de como a *forma* da estrutura define a eficiência do algoritmo. Aqui, a escolha errada transforma um sistema de milissegundos em um processo de horas.

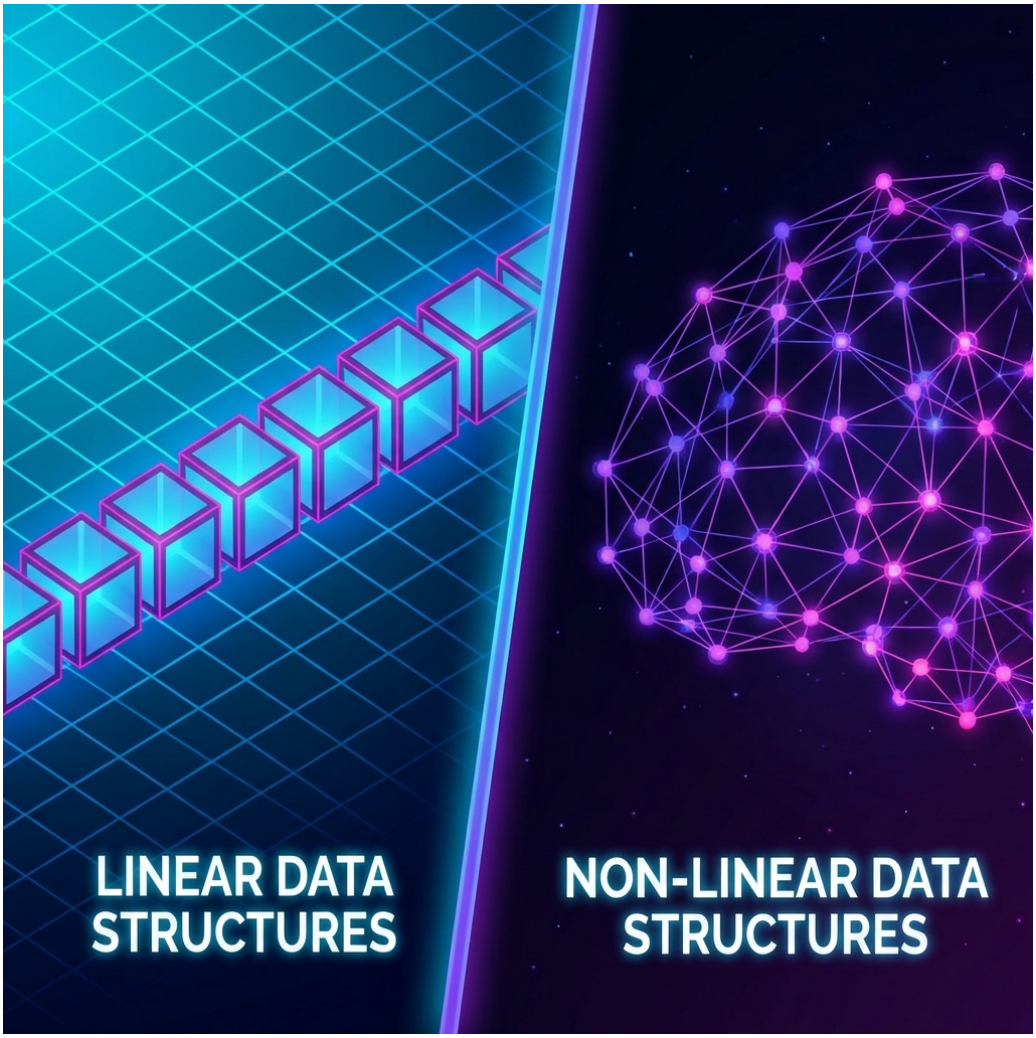

Figure 1: A fronteira entre o mundo Linear de AED I (organização sequencial) e o mundo Não-Linear de AED II (conexo e hierárquico).

1.1. Onde AED I Não é Suficiente

Apesar de todo o arsenal linear, alguns problemas reais de Engenharia de Computação simplesmente não cabem nesse mundo 1D:

- Modelar **hierarquias complexas** (sistemas de arquivos, árvores de cena em jogos, dobras de proteínas em bioinformática).
- Representar **redes de interconexão** com relacionamentos arbitrários (redes sociais, logística de transportes, malhas elétricas).
- Gerenciar **dependências** não sequenciais entre módulos de software ou tarefas em um sistema distribuído.

Nesses cenários, forçar o uso apenas de vetores/listas lineariza artificialmente relacionamentos que são inerentemente multidimensionais. O resultado é um código frágil, difícil de manter e com desempenho proibitivo (buscas lineares em dados que deveriam ser indexados).

0.2 2. Bem-vindos a AED II: O Mundo Não-Linear

Conforme nossa **Ementa Oficial**, aqui os dados ganham complexidade estrutural. O foco é em **Estruturas de Dados Não Lineares** e em algoritmos capazes de tratar volumes massivos de dados navegando por essas estruturas de forma inteligente, evitando varreduras completas.

Figure 2: Estruturas Hierárquicas (Árvores) organizam dados permitindo buscas e atualizações com custo logarítmico, fundamental para escalabilidade.

0.2.1 2.1. Por que estudar isso em Engenharia de Computação?

- **Hierarquia (Árvores):** A espinha dorsal da organização de dados. Do DOM de uma página HTML aos índices B-Tree de um banco de dados PostgreSQL, árvores permitem reduzir o espaço de busca de bilhões de itens para algumas dezenas de operações ($O(\log N)$).
- **Relacionamento (Grafos):** O modelo universal para conexões. Se há objetos e relações entre eles (Cidades e Estradas, Pessoas e Amizades, Transistores e Fios), há um grafo.
- **Otimização Combinatória:** Problemas reais raramente pedem apenas "encontre um caminho". Eles pedem "o melhor caminho", "o menor custo", "o fluxo máximo". Isso exige algoritmos sofisticados sobre grafos.
- **Pensamento Abstrato de Alto Nível:** A capacidade de abstrair um problema barulhento do mundo real em um modelo matemático limpo (Vértices e Arestas) é o que separa o codificador do engenheiro de computação.

Figure 3: Grafos modelam sistemas complexos onde a topologia das conexões define o comportamento da rede, seja ela neural, social ou física.

▷ **Exemplo**

Ao projetar a distribuição de energia de um campus, você pode modelar a rede como um grafo ponderado e aplicar algoritmos de árvore geradora mínima para minimizar custo de cabos, ou algoritmos de fluxo máximo para verificar a capacidade de atendimento da rede em cenários de pico.

0.2.2 2.2. Complexidade Pesada e Fronteira do Possível

Ao contrário de ordenar um vetor em $O(N \log N)$, que é eficiente e tratável, muitos problemas em grafos caem em classes de complexidade inexploradas ou intratáveis. Em AED II, vamos não apenas "resolver problemas", mas também:

- Aprender a classificar a **dificuldade intrínseca** de um problema. Reconhecer quando algo é NP-Completo (como o Caixeiro Viajante) nos poupa de tentar encontrar soluções exatas impossíveis.
- Explorar **heurísticas e aproximações**: quando a solução ótima é inalcançável, como obter uma solução boa o suficiente em tempo hábil?
- Entender o **trade-off** entre pré-processamento (organizar os dados na entrada) e tempo de consulta (queremos respostas rápidas no tempo de execução).

0.2.3 2.3. Linguagem de Programação: Por que Java?

Embora a linguagem C/C++ seja a base tradicional para o ensino de algoritmos e ofereça um controle sem paralelos sobre o hardware (gerenciamento de memória explícito, sem coletores de lixo), nesta disciplina utilizaremos a linguagem **Java**.

A escolha se justifica por:

- **Mercado:** Java continua sendo uma das linguagens mais requisitadas para o desenvolvimento de sistemas corporativos de larga escala.
- **Abstração:** O foco de AED II é a *estrutura* e a *lógica*, e o Java permite que nos concentremos nos algoritmos sem as distrações de aritmética de ponteiros complexa.
- **Ecossistema:** A robustez das bibliotecas padrão e ferramentas de depuração facilita a implementação de sistemas complexos.

Para o desenvolvimento, recomendamos o uso da **JDK 11** (ou superior) e ambientes como **IntelliJ IDEA**, **VSCode** ou mesmo editores leves como o **Sublime Text**.

0.3 3. Estrutura dos Trabalhos Práticos

A disciplina é fortemente prática. Além das listas semanais (Juiz Online ou Questionários), teremos dois grandes **Trabalhos Práticos (TPs)** que consolidam os módulos:

0.3.1 TP1: Indexação e Estruturas Hierárquicas

- **Objetivo:** Implementar um sistema de índices (como um mini-Banco de Dados ou Buscador).
- **Conteúdo:** Árvores AVL, Rubro-Negra ou Árvores B.
- **Desafio:** Lidar com grandes volumes de dados onde a escolha da estrutura define se o programa roda em 1 segundo ou 1 hora.

0.3.2 TP2: Otimização em Grafos Realistas

- **Objetivo:** Resolver um problema de otimização em uma rede real (ex: Rotas de entregas, Alocação de horários, Rede de distribuição de água).
- **Conteúdo:** Caminhos Mínimos (Dijkstra), Árvore Geradora (Kruskal) ou Fluxo Máximo.
- **Desafio:** Modelagem. O difícil não é codar o Dijkstra, é perceber que o problema *é* um Dijkstra disfarçado.

0.3.3 3.3. Juiz Online e a Maratona de Programação

A prática semanal será realizada através de um **Juiz Online (Online Judge)**. Este é um sistema automatizado onde o estudante submete seu código e recebe um feedback imediato. O código é testado contra diversos casos de teste (inclusive casos de borda e grandes volumes de dados) que o estudante não conhece previamente.

Os feedbacks típicos são:

- **Accepted (AC):** Sua lógica está correta e passou em todos os testes.
- **Wrong Answer (WA):** O programa rodou, mas produziu uma saída incorreta para algum caso.
- **Time Limit Exceeded (TLE):** Seu algoritmo é lento demais para o volume de dados do problema.

Este modelo é o mesmo utilizado na **Maratona de Programação da Sociedade Brasileira de Computação (SBC)**.

Figure 4: A essência competitiva e prática de AED II é inspirada no modelo da Maratona de Programação.

0.4 4. Módulos da Disciplina e Objetivos

0.4.1 Módulo 1: Fundamentos e Árvores de Pesquisa

Revisar complexidade matemática, lidar com espalhamento (Hash) e introduzir a busca hierárquica em profundidade e largura (BST).

0.4.2 Módulo 2: Árvores Balanceadas e Memória Secundária

Garantir a eficiência no pior caso (AVL, Rubro-Negra), explorar filas de prioridade (Heaps) e indexar dados massivos em disco (Árvores B/B+).

0.4.3 Módulo 3: Introdução e Percursos em Grafos

Modelar problemas do mundo real em redes estruturadas e realizar percursos baseados em otimização de busca (BFS/DFS) e conexões de custo mínimo (Árvore Geradora Mínima).

0.4.4 Módulo 4: Caminhos Mínimos e Algoritmos Gulosos

Resolver problemas de roteamento determinando rotas ótimas (Dijkstra, Bellman-Ford, Floyd-Warshall) e analisar cenários de alocação de recursos (Coloração de Grafos).

0.4.5 Módulo 5: Fluxo Máximo em Redes e Avançados

Maximizar o transporte de itens através de gargalos (Ford-Fulkerson) e estudar classes de caminhos rigorosos (Grafos Eulerianos e Hamiltonianos).

0.5 5. Avaliação

Componente	Pts	Descrição
Prova 1	35	Árvores e Estruturas Hierárquicas
Prova 2	35	Grafos e Algoritmos
TP 1	7	Indexação (Árvores)
TP 2	7	Otimização (Grafos)
Listas	16	Juiz Online ou Questionários

0.6 6. Objetivos de Aprendizagem

Ao final da disciplina, espera-se que o estudante seja capaz de:

- Modelar problemas reais de Engenharia de Computação utilizando **árvores**, **grafos** e estruturas associadas.
- Escolher estruturas de dados adequadas considerando **tempo de execução**, **uso de memória** e **complexidade de implementação/manutenção**.
- Implementar e analisar algoritmos clássicos de busca, ordenação avançada, árvores balanceadas, caminhos mínimos, fluxo máximo, entre outros.
- Reconhecer quando um problema é **intrinsecamente difícil** (NP-difícil/NP-completo) e propor abordagens aproximadas ou heurísticas.
- Ler, entender e adaptar implementações de referência em linguagens como C, C++ ou Java, respeitando boas práticas de engenharia de software.

0.7 7. Estudos de Caso em Engenharia de Computação

Nesta disciplina, sempre que possível, conectaremos a teoria com estudos de caso:

- **Banco de Dados Relacionais/NoSQL:** uso de árvores B/B+ e hash para índices, impacto em consultas complexas.
- **Sistemas Operacionais:** escalonamento de processos (filas, prioridades, grafos de dependência), detecção de deadlock.
- **Redes de Computadores:** algoritmos de roteamento baseados em caminhos mínimos e árvores geradoras.

- **Sistemas Embarcados e Tempo Real:** garantias de pior caso, limites de latência e uso eficiente de memória.
- **Computação Gráfica e Jogos:** uso de árvores de cena, grafos de estados, grafos de navegação (nav-meshes).

▶ Prática

Ao preparar seus TPs e soluções de listas, tente sempre se perguntar: *"Se eu fosse colocar esta solução em produção em um sistema real, ela se sustentaria em termos de desempenho e manutenção?"*

0.8 8. Exercícios

8.1. Exercícios Conceituais

1 Explique, com suas palavras, a diferença entre **modelar** um problema com vetores/listas e modelá-lo com grafos. Dê um exemplo de problema que fica artificial se forçado a usar apenas vetores.

2 Para cada estrutura a seguir (vetor, lista encadeada, pilha, fila), descreva um cenário de aplicação típico em software de sistemas (por exemplo, compiladores, SO, rede).

3 Considere um sistema de arquivos hierárquico (pastas dentro de pastas). Justifique por que estruturas lineares são inadequadas para representar de forma eficiente esse tipo de dado.

8.2. Exercícios Analíticos

1 Monte um quadro comparando, para vetores, listas encadeadas, pilhas e filas:

- Complexidade de inserção/remoção no início, meio e fim.
- Complexidade de acesso aleatório.
- Impacto qualitativo em consumo de memória e localidade de cache.

2 Desenhe um pequeno grafo que represente um subconjunto da rede de um campus (laboratórios, roteadores, switches) e identifique pelo menos dois problemas práticos que podem ser formulados sobre esse grafo (por exemplo, caminho mínimo, árvore geradora mínima, detecção de articulações).

8.3. Exercícios de Programação

1 Implemente um programa que leia a descrição de um sistema simples de diretórios (usando um formato textual) e construa uma representação em árvore na memória. Forneça operações para:

- Listar o conteúdo de um diretório.
- Calcular a profundidade máxima.

- Contar quantos arquivos existem em um determinado subdiretório.

2 Modele a grade curricular do curso de Engenharia de Computação como um grafo direcionado em que há uma aresta de A para B se A é pré-requisito de B. Escreva um programa que:

- Verifique se há ciclos de pré-requisito (situações impossíveis).
- Gere uma possível ordem de disciplinas a ser cursada respeitando todos os pré-requisitos (ordenação topológica).

3 Pesquise no juiz online da disciplina (ou em plataformas como Beecrowd/Codeforces) um problema em que a solução ideal exija uma estrutura de dados não linear (árvore ou grafo). Resolva o problema e, em um pequeno relatório, explique como você identificou a estrutura adequada.

• Teoria

Donald Knuth e a Bíblia da Programação: Donald Ervin Knuth é frequentemente chamado de "pai da análise de algoritmos". Sua obra monumental, *The Art of Computer Programming*, iniciada na década de 1960 e ainda em desenvolvimento, é a base matemática de quase tudo o que estudamos aqui. Knuth não apenas inventou algoritmos, mas criou o rigor matemático necessário para provar que um algoritmo é melhor que outro. Como curiosidade, ele é famoso por oferecer recompensas financeiras (em cheques de frações de dólar, que se tornaram itens de colecionador) para qualquer pessoa que encontre um erro em seus livros.

Bibliografia Principal:

- CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 2002.
- GOODRICH, Michael T.; TAMASSIA, Roberto. *Estruturas de Dados e Algoritmos em JAVA*.
- KNUTH, Donald E. *The Art of Computer Programming*. Addison-Wesley.

  "Gates afirmou que qualquer pessoa que lesse e compreendesse a série de livros, considerada a 'bíblia da programação', deveria enviar-lhe um currículo para trabalhar na Microsoft."

- ZIVIANI, N. *Projeto de Algoritmos: com implementações em Pascal e C*.