Algoritmos e Estruturas de Dados II

Capítulo da Aula 12: Revisão para Prova 1

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

Fevereiro de 2026

A Prova 1 será o marco divisor do conhecimento avançado da disciplina, onde abandonamos o olhar rasante de repasso e passamos a cobrar intensamente **Engenharia de Sistemas, Fundamentação Hierárquica em Alta Performance e Sustentabilidade Assintótica de Big Data**.

1. O Mapa Arquitetural do Conteúdo

• Teoria

A Prova 1 avalia sua compreensão de como estruturas de dados hierárquicas resolvem limitações das estruturas lineares, garantindo operações eficientes mesmo com grandes volumes de dados.

1.1. Análise de Complexidade

- Notação assintótica: O, Ω, Θ.
- Análise de loops aninhados e recursões.
- Teorema Mestre para divisão e conquista.
- Trade-offs tempo vs espaço.

1.2. Estruturas Lineares (Revisão)

- Vetores, listas encadeadas, pilhas, filas.
- Limitações: busca $O(N)$, ordenação $O(N \log N)$ baseada em comparação.
- Motivação para estruturas não lineares.

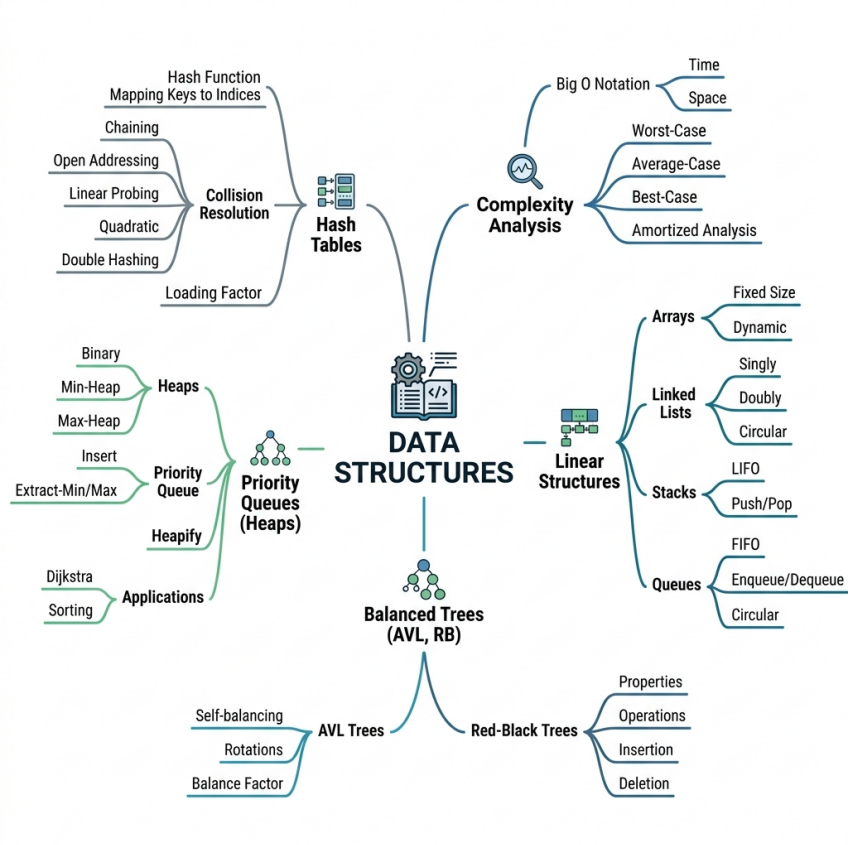

Figure 1: Mapa conceitual: complexidade, BST, AVL, Hash, ordenação, Heap, B-Tree, Segment Tree.

1.3. Tabelas Hash

- Funções de hash e tratamento de colisões.
- Encadeamento vs endereçamento aberto.
- Fator de carga e rehash.
- Complexidade: $O(1)$ médio, $O(N)$ pior caso.

1.4. Ordenação Linear

- Counting Sort: $O(N + K)$ para inteiros em intervalo limitado.
- Radix Sort: ordenação dígito a dígito.
- Bucket Sort: distribuição uniforme.
- Limite inferior $\Omega(N \log N)$ para comparação.

1.5. Árvores Binárias de Busca (BST)

- Propriedade de ordenação: esquerda $<$ raiz $<$ direita.

- Operações: inserção, busca, remoção (casos: 0, 1, 2 filhos).
- Percursos: pre-order, in-order, post-order, level-order.
- Problema: degeneração para lista ($O(N)$ altura).

1.6. A Guerra do Balanceamento: AVL vs. Red-BlackTrees

- **AVL:** A blindagem achatada estrita.
  - Fator de balanceamento amarrado agressivamente $\in \{-1, 0, +1\}$.
  - Minimização letal de varreduras na via; reduz *Cache Misses* na CPU. Suprema e absoluta em **Sistemas Read-Heavy** contínuos.
  - O torque extremo das Rotações O(1) ancoram a altura dura máxima teórica em: $\leq 1{,}44 \log_2(N+2)$.
- **Red-Black Trees:** O Motor Elástico e Resiliente de Kernels.
  - Flexibilidade elástica baseada em fendas (Os 3 Domínios de Inserção) e na superação do infernal Abismo da Remoção com o resgate dos nós **"Duplo-Pretos"**.
  - Resiste imponente a cargas agressivas temporais de **Sistemas Write-Heavy**. Não sofre colapso por cascata de refatoração, sendo adotada no coração do escalonador do Kernel Linux (CFS) e `TreeMap`.
  - Altura relaxada e maleável: $\leq 2 \log_2(N+1)$.

1.7. Filas de Prioridade (Heaps) e a Teoria da Informação

- **O Motor de Min-Heap:** Árvores engessadas blindadas contíguas em memória mapeada; indexação direta em matriz $2i$ (esq) e $2i+1$ (dir).
- Traciona filas sem desbalanceamento por inércia gravitacional de topo: inserção (*sift-up*) e extração (*sift-down*) blindadas invariavelmente em funil $\mathcal{O}(\log N)$.
- **A Compressão de Huffman:** Fiel usuária contínua da Min-Heap. Funde pacotes passivos vitais focada nos abismos de entropia da representação textual.
- Explora arquiteturalmente os **Códigos Livres de Prefixo** e exige análise severa dos *overheads* rotineiros acoplados a tabelas e descritores injetados em cabeçalhos literais para avaliar salvamento limpo bit a bit.

1.8. Árvores B e B+

- Árvores multiway para memória secundária.
- Altura: $O(\log_M N)$ onde M é a ordem.
- Split e crescimento para cima.
- B+: dados apenas nas folhas, folhas encadeadas para range queries.

1.9. Segment Trees e Fenwick Trees

- Segment Tree: consultas e atualizações em intervalo em $O(\log N)$.
- Lazy Propagation: atualizações em intervalo eficientes.
- Fenwick Tree (BIT): otimizada para soma, código mais simples.

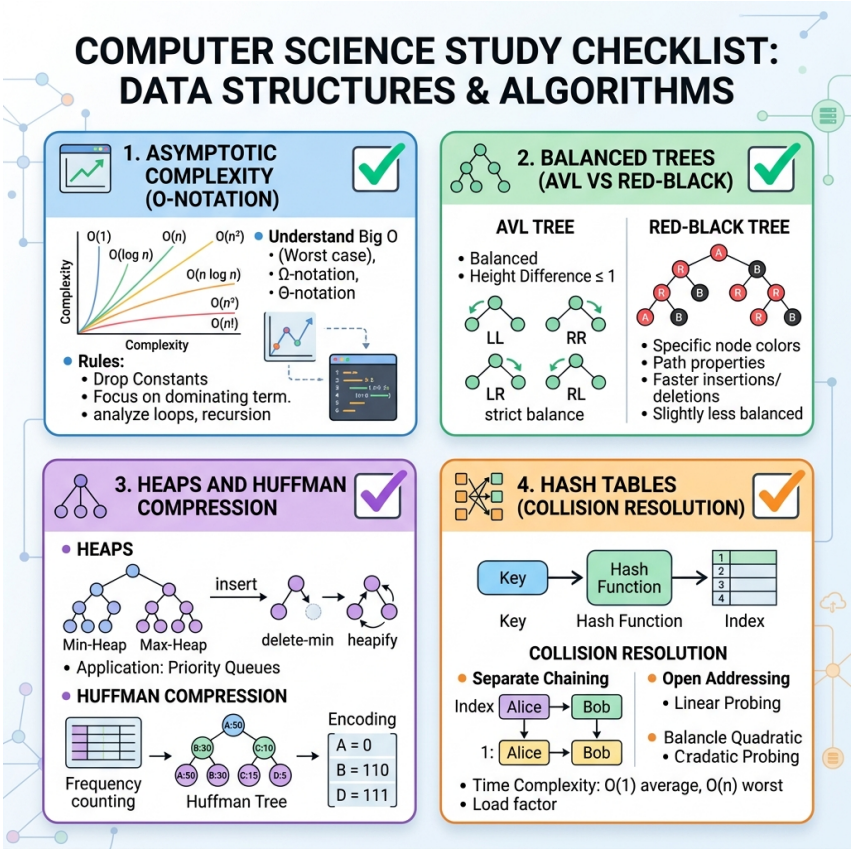

Figure 2: Checklist de estruturas: saber definir, implementar e analisar cada uma.

2. Checklist de Estudos

2.1. Conceitos Teóricos

1 **Complexidade:** Calcular $O(N)$ de loops aninhados e recursões simples. Aplicar Teorema Mestre.

2 **BST:** Propriedades formais, inserção, remoção (todos os casos), percursos.

3 **AVL:** Identificar desbalanceamento, escolher rotação correta (LL, RR, LR, RL).

4 **Hash:** Resolver colisões manualmente (desenhar encadeamento ou tabela com sondagem linear).

5 **Ordenação:** Simular Radix Sort e Counting Sort passo a passo.

6 **Heap:** Simular inserção e remoção em vetor representando heap.

7 **Árvores B:** Calcular altura, simular inserção e split.

2.2. Comparações e Escolhas

- Quando usar Hash vs Tree vs Heap?
- AVL vs Red-Black: trade-offs.
- Segment Tree vs Fenwick Tree: quando cada um?
- Estruturas em memória vs disco (B-Trees).

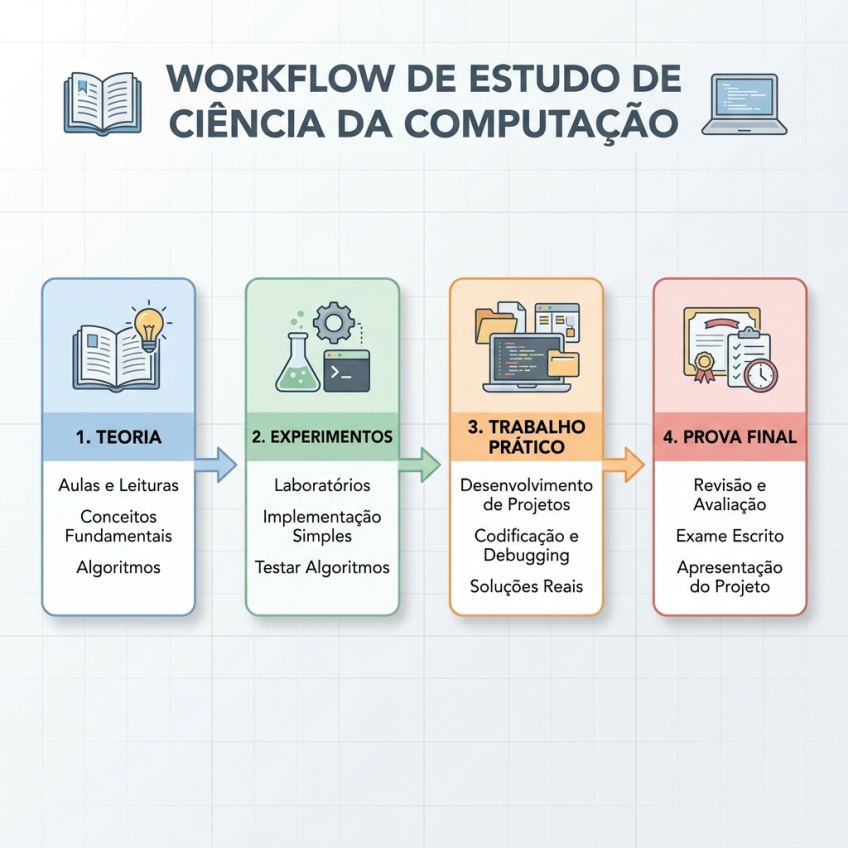

Figure 3: Do conteúdo ao dia da prova: revisão, exercícios, TP1, prática.

3. Exercícios de Revisão Integrada

3.1. Exercícios Conceituais

1 Compare a complexidade de busca em uma BST não balanceada, AVL, Red-Black Tree e tabela hash para $N = 10^6$ elementos. Discuta quando cada estrutura é preferível.

2 Explique por que Counting Sort pode ordenar em $O(N)$ enquanto algoritmos baseados em comparação têm limite inferior $\Omega(N \log N)$.

3 Descreva a relação entre altura de uma árvore e complexidade de operações. Por que balanceamento é crucial?

3.2. Exercícios Analíticos

1 Dada a sequência de inserções `50, 30, 70, 20, 40, 60, 80, 10` em uma BST:

- Desenhe a árvore resultante.
- Converta para AVL aplicando rotações necessárias.
- Calcule a altura antes e depois do balanceamento.

2 Para uma tabela hash de tamanho $M = 11$ com função $h(k) = k \mod 11$, insira as chaves `22, 1, 13, 11, 24, 33` usando:

- Encadeamento exterior.
- Sondagem linear.

Calcule o fator de carga em cada caso.

3 Simule Heap Sort no vetor `[4, 10, 3, 5, 1]`: construa o max-heap e depois extraia elementos ordenadamente.

3.3. Exercícios de Programação

1 Implemente uma comparação prática entre estruturas:

- Meça tempo de inserção de 10^5 elementos em BST, AVL, `TreeSet` e `HashSet`.
- Meça tempo de busca de 10^4 elementos aleatórios em cada estrutura.
- Analise os resultados e discuta trade-offs.

2 Resolva problemas do Trabalho Prático 1 novamente, focando em:

- Justificativa formal de complexidade.
- Escolha adequada de estrutura de dados.
- Otimizações possíveis.

4. O Mindset para a Prova Analítica

▶ Prática

A Prova não é um teste matemático mecânico e inocente! A nossa Avaliação Teórica 1 foi projetada integralmente em cima do aspecto de **Decisão Analítica de Projeto**. Memorizar a receita de bolo de uma rotação Esquerda-Direita não é o diferencial aqui. O diferencial é argumentar **porque** o Kernel do seu S.O. abraça a Árvore Rubro-Negra num surto caótico, e porque uma tabela passiva derrete quando sofre varreduras sofrendo de *Cache Miss*. Estude projetando a melhoria do software como Engenheiro/a, não apenas calculando!

4.1. Estratégia de Resolução

- Leia cuidadosamente o enunciado e identifique qual estrutura de dados é mais adequada.
- Para questões teóricas, use definições formais e propriedades matemáticas.
- Para simulações, trabalhe passo a passo, verificando cada etapa.
- Sempre justifique suas respostas com análise de complexidade quando aplicável.

4.2. Pontos de Atenção

- **Complexidade:** Não confunda melhor caso, caso médio e pior caso.
- **Árvores:** Verifique sempre se a propriedade de ordenação/balanceamento é mantida após operações.
- **Hash:** Lembre-se do impacto do fator de carga e do processo de rehash.
- **Heaps:** Cuidado com indexação (base 0 vs base 1) e cálculo de índices de pai/filhos.

5. Objetivos de Aprendizagem para a Prova

Ao final desta unidade, você deve ser capaz de:

- Analisar complexidade de algoritmos usando notação assintótica.
- Escolher estruturas de dados adequadas para diferentes problemas.
- Implementar e manipular árvores binárias de busca e árvores balanceadas.
- Aplicar técnicas de hash e ordenação linear quando apropriado.
- Justificar escolhas de estruturas com argumentos de complexidade e aplicabilidade prática.