Relatório Técnico: Análise de Desempenho de Estruturas de Dados

Autor: Daniel Gomes Silva **1-24-10445**

1. Metodologia

O objetivo deste trabalho foi comparar o desempenho prático de três estruturas de dados fundamentais — **Vetor**, **Árvore Binária de Busca (ABB)** e **Árvore AVL** — além de algoritmos de ordenação e busca, conforme especificado nos requisitos do projeto.

1.1. Ambiente de Desenvolvimento e Execução

Os testes foram implementados na linguagem **Java**, sem o uso de bibliotecas de estruturas de dados nativas (`java.util.*`), garantindo que todas as estruturas e algoritmos fossem implementações próprias.

- **Linguagem:** Java (JDK).
- **Sistema Operacional:** Windows 11.
- **Hardware Utilizado:**
  - Processador: AMD Ryzen 7 5700X
  - Memória RAM: 16GB DDR4 3000MHz
  - Placa de Vídeo: NVIDIA GTX 1650

1.2. Geração de Dados

Para garantir a integridade dos testes, foi criada uma classe utilitária `GeradorDados` responsável por criar vetores de inteiros em três cenários distintos:

- **Aleatório:** Números gerados randomicamente.
- **Ordenado:** Sequência crescente $(0, 1, 2, ..., n)$.
- **Inversamente Ordenado:** Sequência decrescente $(n, n-1, ..., 0)$.

Os testes foram realizados com cargas de **100**, **1.000** e **10.000** elementos.

1.3. Medição de Tempo

A medição foi realizada utilizando a função `System.nanoTime()` para capturar o tempo de CPU com precisão de nanosegundos. Para mitigar interferências do sistema operacional, cada bateria de testes foi executada **5 vezes**, sendo o resultado final a média aritmética dessas execuções.

2. Resultados

Abaixo são apresentados os tempos médios obtidos para as operações de Inserção, Busca e Ordenação.

2.1. Tempos de Inserção

Tamanho	Cenário	Vetor (ns)	ABB (ns)	AVL (ns)
100	Aleatório	8.260	73.480	110.560

100	Ordenado	6.640	28.880	32.400
100	Inverso	7.420	34.840	36.940
1.000	Aleatório	64.240	108.060	300.520
1.000	Ordenado	47.500	1.096.660	140.520
1.000	Inverso	28.520	1.449.900	71.620
10.000	Aleatório	182.020	785.240	978.780
10.000	Ordenado	41.020	**105.314.040**	**640.720**
10.000	Inverso	25.660	154.659.180	626.420

> **[Sugestão de Gráfico 1]:** *Insira aqui um gráfico de linhas comparando ABB vs. AVL no cenário Ordenado (Eixo X: Tamanho, Eixo Y: Tempo).*

2.2. Tempos de Ordenação (Vetores)

Tamanho	Cenário	Selection Sort (ns)	QuickSort (ns)
100	Aleatório	215.420	59.300
1.000	Aleatório	1.294.520	51.520
10.000	Aleatório	25.209.340	413.920
10.000	Ordenado	24.508.820	33.175.760

> **[Sugestão de Gráfico 2]:** *Insira aqui um gráfico de barras comparando Selection Sort vs. QuickSort para 10.000 elementos (Cenário Aleatório).*

2.3. Tempos de Busca (10.000 Elementos)

Alvo Buscado	Vetor Sequencial (ns)	Vetor Binário (ns)	ABB (ns)	AVL (ns)
Primeiro	280	500	720	2.260
Meio	65.120	540	10.820	720
Último	71.380	540	17.480	180
Inexistente	16.780	420	120	220

3. Análise dos Resultados

Os dados coletados evidenciam as diferenças teóricas de complexidade (Big O) na prática.

3.1. Inserção: O Colapso da ABB

O ponto mais crítico observado foi na inserção de **10.000 elementos ordenados**.

- A **Árvore AVL** manteve um desempenho excelente (~0,6ms), pois suas rotações garantem que a altura da árvore permaneça logarítmica ($O(\log n)$).
- A **Árvore Binária (ABB)** sofreu uma degradação severa, levando cerca de **105ms**. Como os dados entraram ordenados, a ABB não balanceada comportou-se como uma lista encadeada, elevando a complexidade de inserção para $O(n)$ por elemento, ou $O(n^2)$ total. A AVL provou-se cerca de **164 vezes mais rápida** neste cenário.

3.2. Ordenação: $O(n^2)$ vs $O(n \log n)$

Comparando os algoritmos no cenário aleatório com 10.000 itens:

- O **Selection Sort** (25ms) confirmou sua complexidade quadrática $O(n^2)$. O aumento de 10x na entrada (1k para 10k) resultou num aumento de tempo de aproximadamente 100x.
- O **QuickSort** (0,4ms) demonstrou a eficiência do paradigma "dividir para conquistar" ($O(n \log n)$).
- **Nota:** No cenário *Ordenado*, o QuickSort apresentou desempenho inferior (33ms). Isso ocorre devido à escolha do pivô (último elemento), que no vetor já ordenado gera o "pior caso" do algoritmo ($O(n^2)$).

3.3. Busca: A Ineficiência Sequencial

A busca pelo último elemento num vetor de 10.000 posições ilustra a diferença entre busca linear e logarítmica:

- **Vetor Sequencial:** Precisou percorrer todo o vetor (~71.000ns).
- **Busca Binária / AVL:** Foram quase instantâneas (~180ns a 540ns). Isso confirma que, para grandes volumes de dados, estruturas que suportam busca logarítmica são indispensáveis.

4. Conclusão

Este trabalho permitiu implementar e analisar o comportamento de estruturas de dados essenciais. Conclui-se que:

1. Para **pequenos volumes de dados** (100 itens), a simplicidade do Vetor e do Selection Sort é aceitável, pois a diferença de tempo é imperceptível para o usuário.
2. A **Árvore AVL** é superior à ABB em cenários onde a ordem de inserção dos dados não é aleatória, garantindo estabilidade e performance previsível.
3. A escolha do algoritmo correto (ex: QuickSort vs Selection Sort) impacta drasticamente o tempo de execução à medida que o volume de dados escala.

O projeto consolidou o entendimento de que a complexidade teórica (Big O) traduz-se diretamente em ganho ou perda de performance em sistemas reais.