



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CAMPUS DE CRATEÚS**  
**PROFESSOR: RAFAEL MARTINS BARROS**

**CAMILA DE PAIVA CAVALCANTE**  
**KAUAN COELHO VASCONCELOS**

# **BENCHMARK DE ESTRUTURAS DE DADOS**

*TRABALHO PRÁTICO 01*

*RELATÓRIO*

**CRATEÚS**  
**2025**

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Método Experimental</b>	<b>3</b>
2.1	Configuração das Estruturas de Dados . . . . .	3
2.2	Geração de Dados . . . . .	3
2.3	Execução dos Testes e Métricas . . . . .	4
<b>3</b>	<b>Resultado</b>	<b>4</b>
3.1	Desempenho das Tabelas Hash . . . . .	5
3.2	Desempenho das Estruturas de Árvore . . . . .	5
<b>4</b>	<b>Discussão</b>	<b>8</b>
4.1	Análise das Tabelas Hash e Impacto do Fator de Carga . . . . .	8
4.2	Análise das Árvores . . . . .	9
4.3	Síntese Comparativa . . . . .	9
<b>5</b>	<b>Conclusão</b>	<b>10</b>

# 1 Introdução

O presente trabalho tem como objetivo desenvolver e realizar uma análise comparativa do desempenho de três estruturas de dados essenciais: a Tabela Hash, a Árvore AVL e a Árvore Binária de Busca (ABB). A análise visa comparar o desempenho dessas estruturas em diversas condições de carga e diferentes cenários de entrada, empregando experimentos controlados para medir métricas importantes, como tempo de execução, altura da estrutura, frequência de colisões e número de rotações.

Em relação às estruturas de indexação, a Tabela de Espalhamento (Hash) emprega uma função de mapeamento para calcular a posição de armazenamento das chaves, buscando atingir uma complexidade média de busca  $O(1)$ . O principal desafio dessa abordagem reside no tratamento de colisões — a ocorrência de chaves distintas mapeadas para o mesmo endereço. Neste estudo, as colisões são gerenciadas por duas estratégias distintas: o encadeamento externo, que utiliza listas auxiliares para os elementos colididos, e o endereçamento aberto, que resolve os conflitos alocando os elementos na própria tabela.

Por outro lado, as estruturas de árvore proporcionam uma organização dos dados em níveis hierárquicos. A Árvore AVL destaca-se como uma árvore binária de busca balanceada, cuja propriedade fundamental é limitar o desnível entre as subárvores de qualquer nó a, no máximo, uma unidade. Essa característica, mantida por meio de operações de rotação, assegura que as operações de busca, inserção e remoção ocorram em tempo estritamente logarítmico  $O(\log n)$ . Como base comparativa, utiliza-se a Árvore Binária de Busca (ABB) sem balanceamento que, embora estruturalmente mais simples, está sujeita à degeneração para uma lista encadeada, dependendo da ordem de inserção, degradando seu desempenho para  $O(n)$  no pior caso.

Para as implementações da Tabela Hash, adotou-se o Método da Divisão ( $h(k) = k \bmod m$ ) como função de espalhamento. Essa escolha justifica-se por sua eficiência computacional e capacidade de distribuir uniformemente as chaves, especialmente quando a dimensão da tabela ( $m$ ) é um número primo distante de potências de 2, mitigando padrões de agrupamento indesejados.

Por fim, a validação experimental será realizada por meio de um benchmark automatizado. Os experimentos irão mensurar o tempo médio de execução das operações, o número de colisões nas tabelas hash, bem como a quantidade de rotações e a altura final das árvores. Os testes serão conduzidos em três cenários controlados — dados aleatórios, ordenados e quase ordenados —, possibilitando a verificação prática das divergências teóricas de desempenho entre as estruturas de dados.

## 2 Método Experimental

Para a validação prática das estruturas de dados, desenvolveu-se um ambiente de *benchmark* utilizando a linguagem **Java**. Em conformidade com as especificações do trabalho, todas as estruturas — Tabela Hash, Árvore AVL e Árvore Binária de Busca (ABB) — foram implementadas manualmente, sem o uso de bibliotecas de coleções nativas, visando o controle total sobre os algoritmos de manipulação de dados.

### 2.1 Configuração das Estruturas de Dados

#### Tabela Hash:

A tabela foi dimensionada com um tamanho fixo de  $M = 200.003$  posições. A escolha de um número primo é estratégica pois, em conjunto com o Método da Divisão ( $h(k) = k \pmod{M}$ ), minimiza a ocorrência de colisões ao evitar padrões comuns na distribuição das chaves. Foram implementadas duas estratégias para o tratamento de colisões:

- **Encadeamento Externo (*Chaining*):** Utiliza uma lista encadeada em cada posição da tabela para armazenar elementos colidentes.
- **Endereçamento Aberto (*Open Addressing*):** Emprega a técnica de **Tentativa Linear**, onde, em caso de colisão, busca-se a próxima posição vazia sequencialmente através da função:

$$h(k, i) = (h'(k) + i) \pmod{M}$$

#### Árvores:

- **Árvore AVL:** Implementada como uma árvore de busca auto-balanceável. A cada inserção ou remoção, o algoritmo verifica o fator de balanceamento dos nós (diferença de altura entre subárvores). Caso o fator exceda 1 em módulo, aplicam-se rotações (simples ou duplas) para restaurar o equilíbrio e garantir altura  $O(\log n)$ .
- **Árvore Binária de Busca (ABB):** Implementada sem mecanismos de balanceamento. Sua topologia depende exclusivamente da ordem de inserção, tornando-a suscetível à degeneração para uma lista linear ( $O(n)$ ) no pior caso (dados ordenados).

### 2.2 Geração de Dados

Para assegurar a equidade do comparativo, todas as estruturas foram testadas com os mesmos conjuntos de chaves, gerados pseudo-aleatoriamente com valores inteiros no intervalo  $[1, 10^9]$ . Foram definidos três cenários de entrada:

1. **Aleatório:** Chaves geradas sem qualquer ordenação prévia.
2. **Ordenado:** Chaves inseridas em ordem estritamente crescente.
3. **Quase Ordenado:** Conjunto onde 90% dos elementos estão ordenados e 10% foram embaralhados, simulando dados parcialmente classificados.

## 2.3 Execução dos Testes e Métricas

O tamanho padrão da amostra foi definido em  $N = 100.000$  elementos. O protocolo experimental seguiu três etapas sequenciais para cada estrutura avaliada:

1. **Inserção** de  $N$  chaves na estrutura;
2. **Busca** de  $N$  chaves, configurada de modo que 50% das chaves pesquisadas estivessem presentes e 50% fossem valores aleatórios ausentes, simulando um cenário de uso real;
3. **Remoção** de 10% das chaves previamente inseridas.

As métricas de desempenho coletadas incluíram o tempo médio de execução das operações (via `System.nanoTime()`), a altura final das árvores e as métricas de esforço interno específicas: número total de colisões para as Tabelas Hash e quantidade de rotações para a Árvore AVL.

### Ajuste Técnico para a ABB:

Durante a execução dos testes com dados Ordenados e Quase Ordenados, a Árvore Binária de Busca (ABB) apresentou comportamento degenerativo, atingindo uma profundidade excessiva próxima a  $N$ . Tal característica resultou em erros de estouro de pilha (`StackOverflowError`) devido à recursividade profunda dos algoritmos de inserção. Para viabilizar a medição de desempenho sem falhas fatais, a carga de trabalho para a ABB nestes cenários específicos foi reduzida para  $N = 20.000$ , permitindo demonstrar a degradação de desempenho sem exceder os limites de memória da pilha de execução.

## 3 Resultado

Os experimentos foram conduzidos em um ambiente controlado, executando os algoritmos para os três cenários de entrada propostos. A seguir, são apresentados os dados coletados referentes ao tempo de execução, comportamento de colisões e propriedades estruturais das árvores.

### 3.1 Desempenho das Tabelas Hash

A **Tabela 1** resume o desempenho das implementações de Hash com Encadeamento Externo e Endereçamento Aberto (Sondagem Linear) para uma carga de  $N = 100.000$  elementos e tamanho da tabela  $M = 200.003$  (Fator de carga  $\approx 0,5$ ). O valor "Colisões" refere-se ao número total de conflitos de mapeamento (para encadeamento) ou sondagens adicionais necessárias (para endereçamento aberto).

Tabela 1: Comparativo de desempenho das Tabelas Hash

Estrutura	Cenário	Colisões	Inserção (ms)	Busca (ms)	Remoção (ms)
Encadeamento	Aleatório	21.312	14,93	8,10	2,50
Encadeamento	Ordenado	0	1,01	2,11	0,08
Encadeamento	Quase Ord.	4.487	1,47	11,80	0,50
Endereçamento	Aleatório	50.119	7,67	5,44	0,95
Endereçamento	Ordenado	0	1,75	<b>8.568,93</b>	0,05
Endereçamento	Quase Ord.	96.453	2,02	16,97	0,06

Fonte: Dados gerados pelo benchmark.

### 3.2 Desempenho das Estruturas de Árvore

A Tabela 2 apresenta os resultados para a Árvore AVL e a Árvore Binária de Busca (ABB). Devido à instabilidade da ABB em cenários de pior caso (dados ordenados), a amostra N foi reduzida nesses testes específicos para viabilizar a medição, conforme indicado na coluna "Amostra (N)".

Tabela 2: Comparativo de desempenho e métricas estruturais das Árvores

Estrutura	Cenário	Amostra (N)	Altura Final	Rotações	Inserção (ms)	Busca (ms)
AVL	Aleatório	100.000	20	74.549	56,04	12,92
AVL	Ordenado	100.000	17	104.980	9,80	9,87
AVL	Quase Ord.	100.000	19	147.047	14,54	6,03
ABB	Aleatório	100.000	41	-	41,57	26,59
ABB	Ordenado	<b>20.000*</b>	18.000	-	2.424,12	2.564,69
ABB	Quase Ord.	<b>20.000*</b>	16.300	-	1.967,04	345,31

Nota: Nos cenários Ordenado e Quase Ordenado para a ABB, o tamanho da entrada foi reduzido para 20.000 elementos devido à degeneração da estrutura, que inviabilizou testes com 100.000 elementos por estouro de pilha (StackOverflow)..

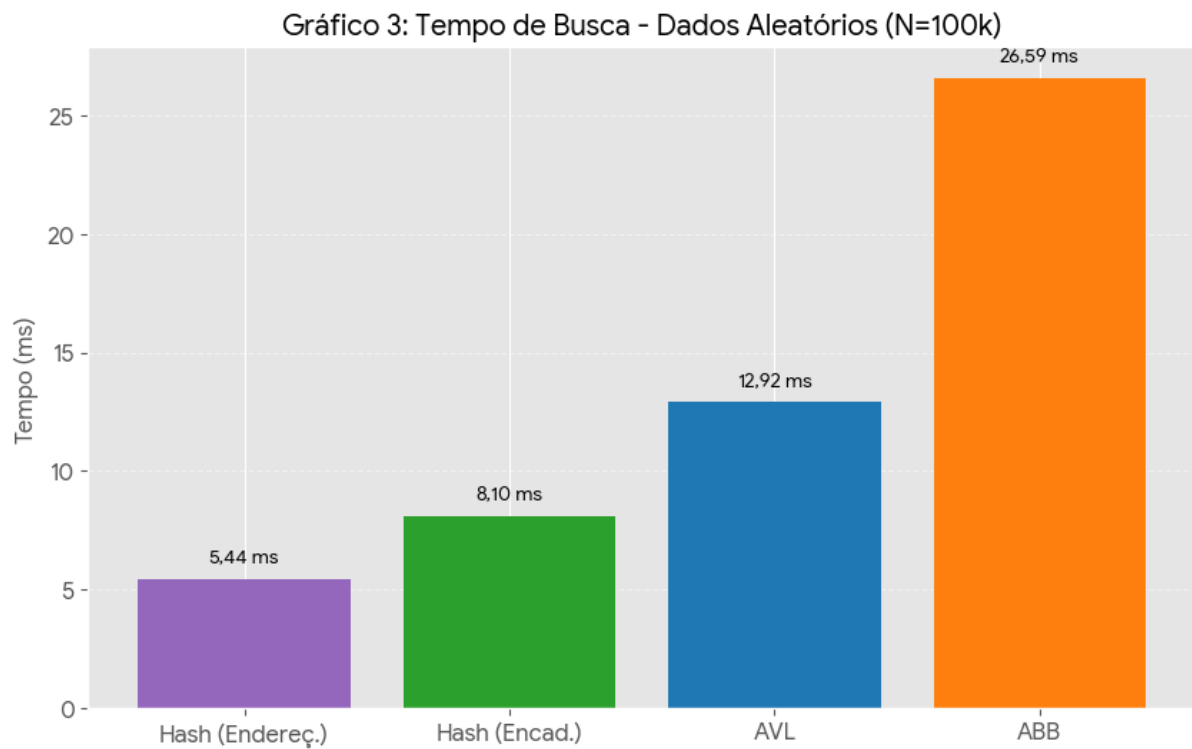


Figura 1: **Tempo Médio de Busca em Dados Aleatórios ( $N = 100k$ )**. Resultados obtidos em condições ideais de distribuição uniforme. As Tabelas Hash demonstram o melhor desempenho ( $< 10ms$ ), validando a complexidade média  $O(1)$ . A Árvore AVL apresenta tempo intermediário consistente, e a ABB, embora mais lenta que as demais (26,59ms), ainda opera de forma eficiente quando os dados não induzem o desbalanceamento.

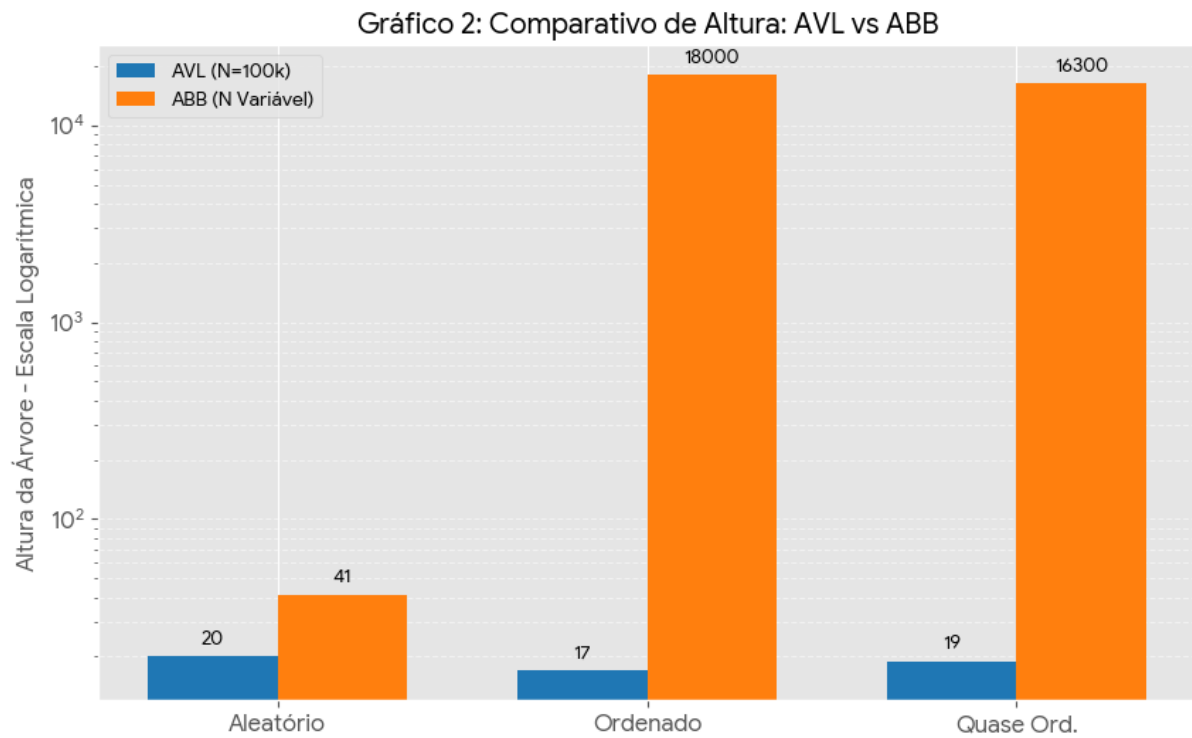


Figura 2: **Comparativo de Altura Final: AVL vs. ABB.** Análise da profundidade das árvores nos três cenários de teste. Enquanto a Árvore AVL mantém sua altura controlada ( $H \approx 17 - 20$ ) graças ao balanceamento automático, a ABB degenera para uma estrutura linear nos cenários Ordenado e Quase Ordenado, atingindo alturas próximas ao número de elementos (18.000 para  $N=20k$ ), confirmando o comportamento de pior caso  $O(N)$ .



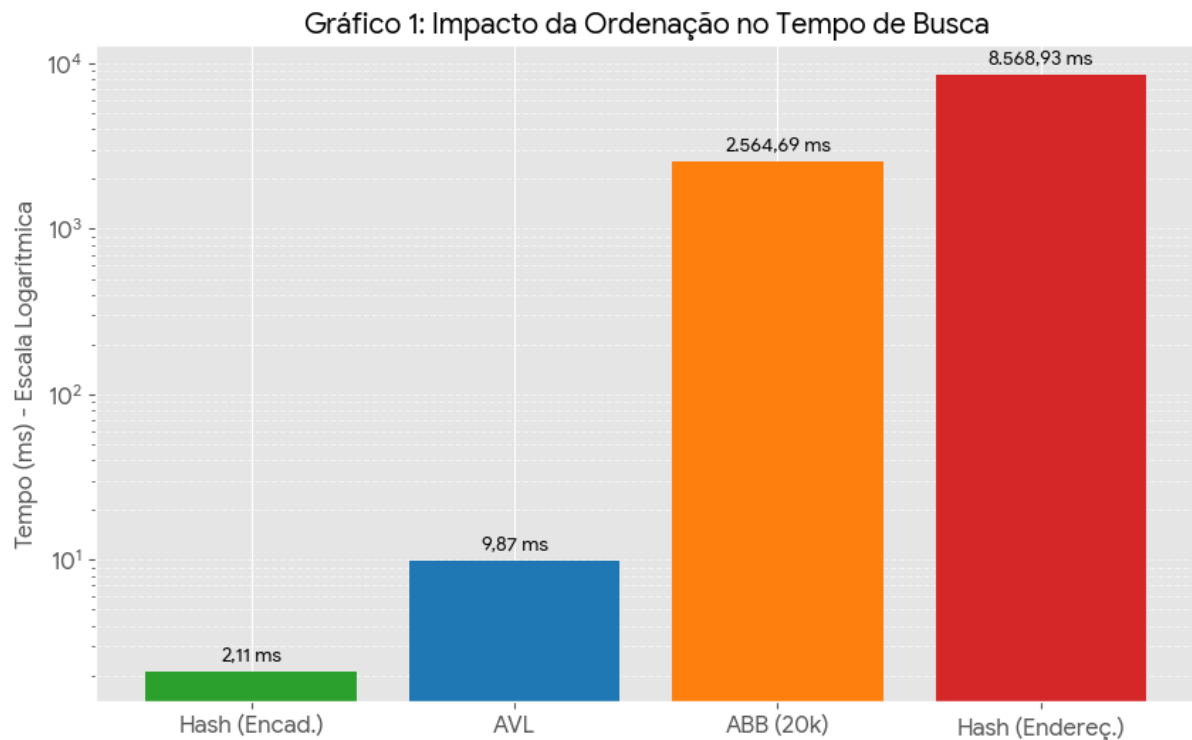


Figura 3: **Impacto da Ordenação no Tempo de Busca (Escala Logarítmica)**. O gráfico evidencia a disparidade de desempenho no cenário de dados Ordenados. Nota-se a eficiência robusta da Tabela Hash com Encadeamento e da Árvore AVL, que mantiveram tempos baixos. Em contraste, o Hash com Endereçamento Aberto e a ABB sofreram degradação severa devido ao agrupamento primário e à degeneração estrutural, respectivamente, exigindo o uso de escala logarítmica para visualização e redução da amostra da ABB para 20k elementos.

## 4 Discussão

A análise dos resultados experimentais permite correlacionar o desempenho prático das estruturas com suas propriedades teóricas, evidenciando como características específicas dos dados de entrada afetam a eficiência computacional.

### 4.1 Análise das Tabelas Hash e Impacto do Fator de Carga

Os resultados obtidos demonstram uma disparidade significativa entre as estratégias de tratamento de colisão. O fator de carga utilizado foi de  $\alpha \approx 0,5$  (100.000 elementos em 200.003 posições). Teoricamente, um  $\alpha$  baixo garante desempenho próximo de  $O(1)$  para ambas as estratégias em dados uniformes, o que foi confirmado no cenário Aleatório, onde ambas tiveram tempos de busca reduzidos ( $< 10$  ms).

No entanto, no cenário Ordenado, observou-se um comportamento divergente:

- **Encadeamento Externo:** Manteve-se robusto com 0 colisões. Isso é explicado pela natureza da função de hash ( $h(k) = k \pmod{M}$ ) e pelo dimensionamento da

tabela. Como as chaves inseridas (1 a 100.000) eram menores que  $M$  (200.003), cada chave foi mapeada para uma posição única, resultando em acesso direto sem necessidade de percorrer listas auxiliares.

- **Endereçamento Aberto:** Sofreu uma degradação severa, com o tempo de busca atingindo 8.568 ms (contra 1,75 ms de inserção). Este comportamento confirma o fenômeno de Agrupamento Primário. A inserção sequencial criou um bloco contíguo de 100.000 posições ocupadas. As buscas por chaves inexistentes foram forçadas a percorrer linearmente todo esse bloco, elevando a complexidade para  $O(N)$ , invalidando a eficiência teórica esperada mesmo com o fator de carga controlado.

## 4.2 Análise das Árvores

A comparação entre AVL e ABB ilustra a importância crítica do balanceamento. A Árvore AVL apresentou desempenho estável e previsível. Mesmo sob a carga de dados ordenados, a altura da árvore manteve-se em 17, o que é consistente com a cota teórica logarítmica, visto que  $\log_2(100.000) \approx 16,6$ . Isso comprova que a estrutura mantém altura  $O(\log n)$  independentemente da entrada. O “preço” dessa eficiência foi o alto número de 104.980 rotações, indicando que a estrutura realizou ajustes constantes para corrigir os desbalanceamentos provocados pela inserção sequencial.

Em contraste, a Árvore Binária de Busca (ABB) falhou em lidar com dados ordenados. A necessidade de reduzir a amostra para 20.000 elementos foi causada pela degeneração da estrutura. Ao inserir dados em ordem crescente sem balanceamento, cada novo nó foi adicionado como filho direito do anterior, transformando a árvore em uma lista encadeada. Os dados corroboram isso: para apenas 20.000 elementos, a altura atingiu 18.000, aproximando-se da relação linear  $h \approx N$ . Isso explica o tempo excessivo de busca (2.564 ms para apenas 20k elementos), confirmando que a ABB possui complexidade  $O(N)$  no pior caso.

## 4.3 Síntese Comparativa

O experimento evidenciou que, embora a Tabela Hash ofereça o melhor desempenho médio teórico ( $O(1)$ ), ela é sensível à distribuição dos dados e à estratégia de colisão, podendo degradar drasticamente em casos de agrupamento. Já entre as árvores, a AVL provou ser a escolha superior para aplicações gerais, garantindo tempos de resposta rápidos ( $O(\log n)$ ) independentemente da desordem dos dados, enquanto a ABB simples mostrou-se inadequada para cenários reais onde a aleatoriedade da entrada não pode ser garantida.

## 5 Conclusão

Os experimentos realizados validaram as previsões teóricas e demonstraram que a **Tabela Hash com Encadeamento Externo** obteve o melhor desempenho global, mantendo tempos próximos a  $O(1)$  e imunidade ao padrão dos dados. Em contrapartida, o **Endereçamento Aberto** mostrou-se instável no cenário ordenado, onde o *Agrupamento Primário* degradou severamente a performance.

No comparativo das árvores, a **AVL** provou ser a estrutura mais robusta, garantindo eficiência  $O(\log n)$  independente da entrada. Já a **ABB** revelou-se inviável para uso geral: sua degeneração com dados ordenados não apenas elevou o tempo de execução, mas esgotou os recursos físicos da máquina (levando ao *StackOverflow*), comprovando que a complexidade assintótica tem consequências práticas críticas.

Conclui-se, portanto, que não existe uma estrutura universalmente superior, sendo o contexto o fator determinante. A Tabela Hash Encadeada é a melhor escolha para recuperação rápida sem requisitos de ordenação, enquanto a AVL é indispensável para cenários que exigem garantia de performance no pior caso. Por fim, o estudo evidenciou que detalhes de implementação — como a escolha de um número primo (200.003) para o dimensionamento da tabela e o uso de marcadores lógicos (DELETED) — são tão decisivos para a eficiência do sistema quanto a escolha do algoritmo em si.