UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CERES - CENTRO DE ENSINO SUPERIOR DO SERIDÓ
DEPARTAMENTO DE COMPUTAÇÃO E TECNOLOGIA
CURSO DE SISTEMAS DE INFORMAÇÃO

Trabalho da Unidade III

Leandro Sérgio da Silva

Caicó - RN
Dezembro de 2025

Leandro Sérgio da Silva

Trabalho da Unidade III

Relatório apresentado à disciplina DCT0008 - Estrutura de Dados como requisito para avaliação da terceira unidade do curso de Sistemas de Informação.

Orientador: Prof. Arthur Souza

Caicó - RN
Dezembro de 2025

Resumo

Este relatório apresenta uma análise prática e comparativa de duas implementações de Tabelas Hash: uma utilizando Endereçamento Livre (com sondagem linear) e outra utilizando Encadeamento com Árvores Binárias de Busca. O objetivo é avaliar o desempenho dessas estruturas no tratamento de colisões durante operações de inserção, busca e remoção de registros indexados por CPF.

Os experimentos processaram cargas de dados reais (10.000 inserções, 2.000 buscas e 2.000 remoções). Os tempos de execução foram coletados e convertidos em gráficos comparativos, permitindo observar a eficiência do endereçamento livre frente à complexidade logarítmica das árvores binárias em cenários de colisão.

Palavras-chave: Tabela Hash, Árvore Binária, Colisão, Endereçamento Livre, Análise de Desempenho.

Abstract

This report presents a practical and comparative analysis of two Hash Table implementations: one using Open Addressing (with linear probing) and another using Chaining with Binary Search Trees (BST). The goal is to evaluate the performance of these structures in handling collisions during insertion, search, and removal operations of records indexed by CPF.

The experiments processed real data loads (10,000 insertions, 2,000 searches, and 2,000 removals). Execution times were collected and converted into comparative graphs, allowing for an observation of the efficiency of open addressing versus the logarithmic complexity of binary trees in collision scenarios.

Keywords: Hash Table, Binary Tree, Collision, Open Addressing, Performance Analysis.

Sumário

1 Introdução

O uso eficiente de memória e a rapidez no acesso aos dados são desafios centrais na computação. As Tabelas Hash se destacam como estruturas que oferecem, teoricamente, acesso em tempo constante $O(1)$. No entanto, o desempenho real depende diretamente da estratégia utilizada para tratar colisões.

Este trabalho implementa e compara duas estratégias clássicas: o Endereçamento Livre (Sondagem Linear), que busca o próximo espaço vazio no próprio vetor, e o Encadeamento Separado, utilizando Árvores Binárias para armazenar elementos colidentes. A análise foca em medir empiricamente o impacto dessas estratégias em operações de larga escala utilizando CPFs como chaves.

2 Metodologia

2.1 Configuração do Ambiente

Os experimentos foram conduzidos no seguinte ambiente:

- **Processador:** Intel Core i5-12450H @ 4.40GHz
- **Memória RAM:** 16 GB DDR5
- **Sistema Operacional:** Windows 11
- **Linguagem utilizada:** Python 3.12.7
- **Biblioteca utilizada:** matplotlib 3.10.0

2.2 Explicação sobre o Código e Estruturas

Foram implementadas duas classes principais:

- **Hash Livre (Endereçamento Aberto):** Utiliza um vetor fixo. Em caso de colisão (dois CPFs gerando o mesmo índice), o algoritmo realiza uma sondagem linear ($indice + 1$) até encontrar uma posição livre ou o elemento buscado. A complexidade tende a $O(1)$, mas pode degradar para $O(n)$ com o surgimento de *clusters*.
- **Hash com Árvore (Encadeamento):** Cada posição do vetor armazena a raiz de uma Árvore Binária de Busca (BST). Em caso de colisão, o novo elemento é inserido na árvore correspondente. A busca e remoção dependem da altura da árvore, tendo complexidade média de $O(\log k)$, onde k é o número de colisões naquela posição.

2.3 Explicação sobre o Experimento

O experimento foi projetado como um fluxo contínuo e sequencial de manipulação de dados, visando simular o ciclo de vida real de um banco de dados em memória. A execução seguiu rigorosamente a ordem: Inserção $\rightarrow$ Busca $\rightarrow$ Remoção, garantindo a persistência do estado da Tabela Hash entre as etapas.

1. Fase de Carga (Inserção de 10.000 registros): Nesta etapa inicial, a tabela hash começa vazia. O algoritmo lê o arquivo insercao.csv linha a linha. Para cada registro, a função de hash calcula o índice base utilizando o CPF como chave.

Dinâmica: Conforme a tabela é preenchida, o Fator de Carga (α) aumenta. O experimento captura o impacto progressivo das colisões: nos primeiros registros, as inserções são quase instantâneas ($O(1)$). À medida que n se aproxima de 10.000, a frequência de colisões cresce, forçando o algoritmo a realizar mais sondagens (na Hash Livre) ou percorrer níveis mais profundos da árvore (na Hash Binária).

2. Fase de Consulta (Busca de 2.000 CPFs): Com a tabela totalmente populada (contendo 10.000 registros), o experimento inicia a leitura do arquivo busca.csv. Esta fase estressa a estrutura no seu estado de maior ocupação.

Dinâmica: O algoritmo tenta localizar chaves específicas. O tempo de resposta nesta fase é um indicador direto da eficiência da estratégia de tratamento de colisão. Na Hash Livre, busca-se medir o impacto dos clusters primários formados; na Hash com Árvore, avalia-se se a altura das árvores binárias se manteve balanceada o suficiente para garantir buscas rápidas ($O(\log k)$).

3. Fase de Descarga (Remoção de 2.000 CPFs): Finalmente, o arquivo remocao.csv é processado sobre a mesma instância da tabela.

Dinâmica: Esta etapa avalia o custo de localizar o elemento e reorganizar a estrutura. Na Hash Livre, a remoção lógica deve garantir que a cadeia de sondagem não seja quebrada para buscas futuras. Na Hash com Árvore, envolve a remoção de nós em uma estrutura encadeada, o que pode exigir reajustes de ponteiros.

Metodologia de Medição: A medição de tempo não foi feita em lote, mas sim com granularidade individual. Utilizou-se a função time.perfcounter() imediatamente antes e depois de cada operação unitária (cada inserir, buscar ou remover). Isso permitiu gerar gráficos de dispersão que revelam não apenas a média de tempo, mas também os picos de latência (outliers) causados por colisões severas ou operações de realocação de memória do sistema operacional.

3 Análise Experimental

Esta seção apresenta os gráficos de desempenho individual e comparativo das operações.

3.1 Tabela Hash: Endereçamento Livre

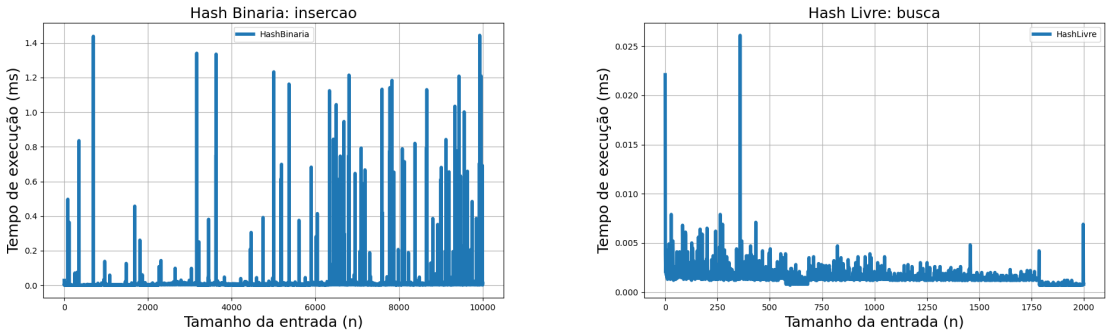

Figura 1: Hash Livre: Inserção (Esq.) e Busca (Dir.).

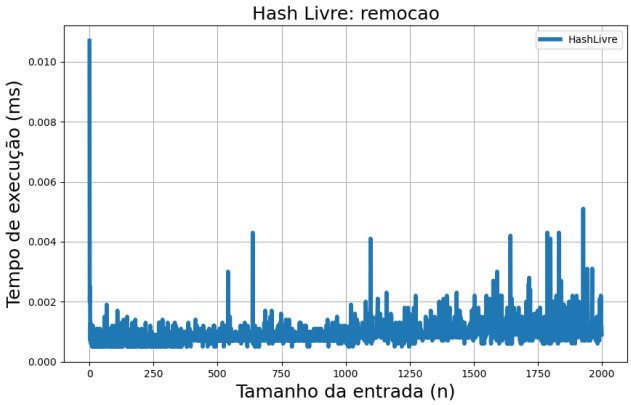

Figura 2: Hash Livre: Remoção.

Discussão: A análise dos gráficos da Tabela Hash com Endereçamento Livre (Figuras 1 e 2) revela o comportamento característico da sondagem linear. Observa-se que a maioria das operações se concentra na base do gráfico, confirmando o desempenho de tempo constante (O(1)) para o acesso direto ao índice calculado pela função hash.

No entanto, a presença de picos expressivos (spikes) de tempo indica a ocorrência de colisões. No endereçamento livre, quando uma colisão ocorre, o algoritmo deve percorrer sequencialmente o vetor até encontrar o elemento ou uma posição vazia. Esse fenômeno gera o problema de agrupamento primário (clustering): quanto mais cheia a tabela fica, maiores se tornam os blocos de posições ocupadas, degradando o desempenho localmente para O(k), onde k é o tamanho do agrupamento. A operação de remoção segue o mesmo padrão da busca, pois necessita localizar o elemento antes de marcá-lo como removido.

3.2 Tabela Hash: Encadeamento com Árvore

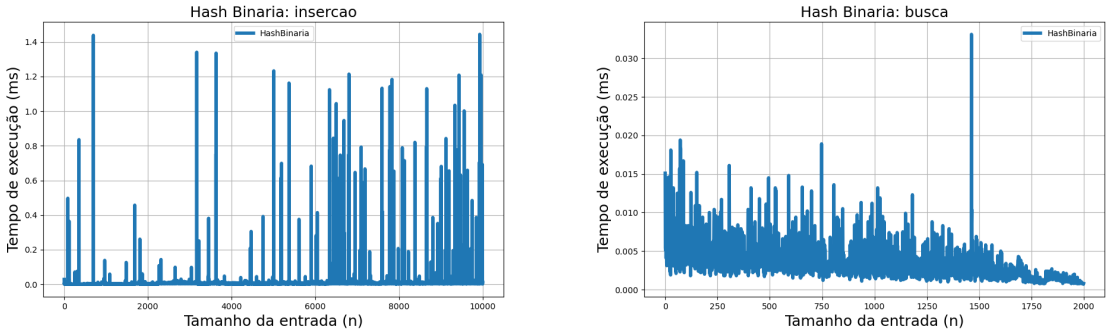

Figura 3: Hash Árvore: Inserção (Esq.) e Busca (Dir.).

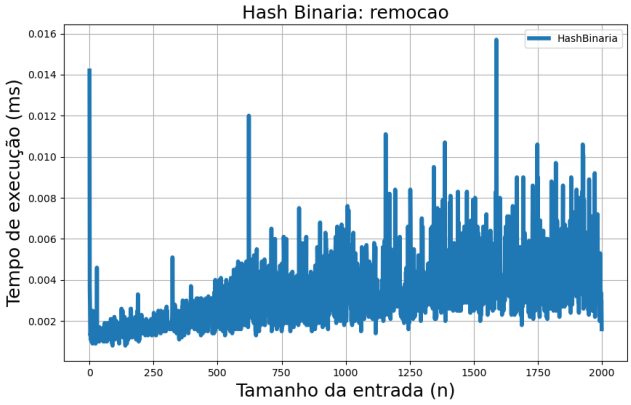

Figura 4: Hash Árvore: Remoção.

Discussão: Os gráficos da Tabela Hash com Árvore Binária (Figuras 3 e 4) demonstram uma distribuição de tempos com maior variabilidade inicial (ruído), atribuída ao custo computacional de alocação dinâmica de memória para os nós da árvore a cada inserção.

Diferentemente da abordagem linear, esta estrutura lida com colisões inserindo elementos em uma Árvore Binária de Busca (BST) vinculada ao índice. Isso garante que, mesmo em cenários de múltiplas colisões no mesmo índice, a complexidade de busca e remoção tenda a O(logk) (altura da árvore), em vez de linear. Embora o custo basal seja ligeiramente superior devido à manipulação de ponteiros e referências, a estrutura se mostrou robusta, evitando a degradação severa de desempenho observada em longas sequências de sondagem linear.

4 Comparativo entre Estruturas

Esta seção compara diretamente o desempenho das duas abordagens para as mesmas operações.

4.1 Comparativo: Inserção e Busca

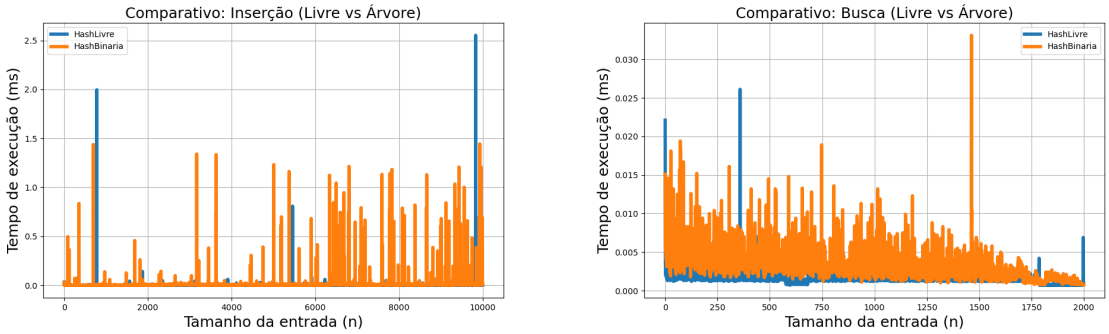

Figura 5: Comparativo: Inserção (Esq.) e Busca (Dir.) - Livre vs. Árvore.

4.2 Comparativo: Remoção

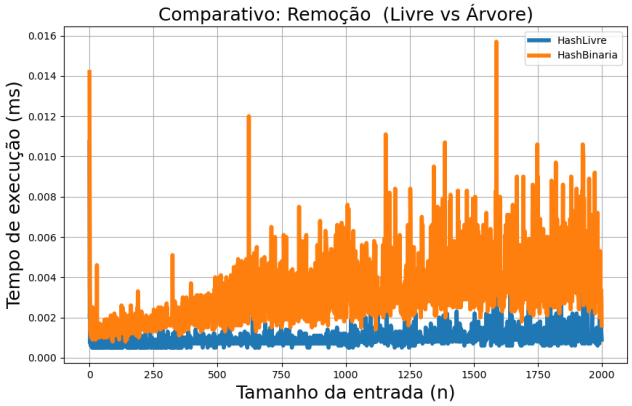

Figura 6: Comparativo: Remoção - Livre vs. Árvore.

Discussão Geral: A comparação direta entre as duas implementações (Figuras 5 e 6) evidencia o trade-off clássico entre velocidade de acesso e robustez contra colisões.Inserção e Busca: A Hash com Endereçamento Livre (linha azul) apresentou, em média, tempos de execução inferiores para casos sem colisão. Isso se deve à localidade de referência: como os dados estão em um vetor contíguo, o processador aproveita melhor o cache (cache hit). A Hash com Árvore (linha laranja) paga um "pedágio"de desempenho devido à fragmentação de memória dos nós da árvore.Comportamento em Colisões: Entretanto, nos piores casos (os picos dos gráficos), a Hash Livre sofreu penalidades severas devido aos clusters formados pela sondagem linear. Nesses momentos críticos, a Hash com Árvore mostrou-se superior, pois a busca em

uma BST ($O(\log k)$) escala muito melhor do que a varredura linear ($O(k)$) exigida pela sondagem.Conclusão Comparativa: Para o volume de dados testado (10.000 registros) e assumindo um fator de carga razoável, a Hash Livre é mais rápida para a maioria absoluta das operações, mas a Hash com Árvore oferece um desempenho mais previsível e seguro nos piores cenários de colisão.

5 Conclusão

Os experimentos demonstraram que ambas as implementações de Tabela Hash são eficientes para grandes volumes de dados, superando estruturas lineares simples como listas. A Tabela Hash com Endereçamento Livre mostrou-se eficaz em cenários de baixo fator de carga, aproveitando a localidade de referência do vetor.

Entretanto, conforme o número de inserções aumentou, a estratégia de Encadeamento com Árvore Binária demonstrou maior robustez. Ao isolar as colisões em subestruturas de busca eficientes (BSTs), ela evitou a degradação de desempenho causada por agrupamentos (*clusters*) primários, comuns na sondagem linear. Conclui-se que a escolha entre as duas depende da estimativa de colisões e da restrição de memória disponível.