Luiz Henrique Nascimento da Silva

Relatório Técnico Final de Estrutura de Dados

Sorocaba
Junho de 2025

Luiz Henrique Nascimento da Silva

Relatório Técnico Final de Estrutura de Dados

Relatório tecnico do trabalho final da disciplina Estrutura de Dados

Instituto de Ciência e Tecnologia de Sorocaba

Orientador: Leopoldo André Dutra Lusquino Filho

Sorocaba
Junho de 2025

Resumo

O presente trabalho tem como objetivo o desenvolvimento e análise comparativa de seis estruturas de dados (Árvore AVL, Lista Encadeada, Lista Encadeada Ordenada, Trie, Tabela Hash e Skip List) aplicadas a um sistema simulado de gerenciamento de dados agrícolas desenvolvido em linguagem C. O sistema tem o objetivo de manipular de forma eficiente um conjunto de dados com 10.262 amostras e nove features (ID, Data, Localização, Tipo de plantio, Preço por tonelada (Dólares/tonelada), Rendimento (quilogramas por hectare), Produção (toneladas), Área plantada (hectares) e Valor total da safra (Dólares)). O sistema foi projetado para executar cinco operações principais: inserção de nova amostra pelo usuário, busca de amostras por ID, busca de amostras por prefixo do estado e cultura, remoção de uma amostra e listagem de todas as amostras presentes. Internamente, as estruturas são capazes de executar dez operações, tais como inserção, inserção pelo usuário, busca, busca com filtros, remoção, entre outras. Para avaliar a eficiência computacional, foram realizados testes de benchmark padronizados, avaliando métricas como tempo de inserção, tempo de remoção, tempo de busca e uso de memória. Além disso, foram realizados testes em cinco condições de restrição: inserção com memória restrita, inserção com delay, busca com latência, inserção com perda e busca com limite de acessos.

1 Introdução

O desenvolvimento de sistemas, nos tempos atuais, demanda cada vez mais eficiência computacional, dado que qualquer processamento de dados desnecessário acarreta desperdício de tempo e recursos. Nesse contexto, é crescente a busca por estruturas de dados que manipulem eficientemente grandes volumes de dados com limitações de hardware. Diante disso, este trabalho apresenta a implementação e análise comparativa de seis estruturas de dados: árvore AVL, tabela hash, lista encadeada, lista encadeada ordenada, skip list e trie em um sistema simulado. O sistema foi desenvolvido em linguagem C e realiza a manipulação de um dataset com 10.262 amostras e nove features, realizando operações de inserção, busca, busca por prefixos, remoção e listagem de dados. A análise do desempenho das estruturas é feita a partir de benchmarks padronizados e testes em condições de restrição, permitindo avaliar o consumo de memória, tempo de execução e escalabilidade das estruturas.

2 Objetivos Gerais

O trabalho possiu por obejtivo implementar um sistema simulado para gerenciamento de dados agricolas eficiente e de facil uso por parte do usuário, visando aplicar os conceitos aprendidos na disciplina e comparar diferentes estruturas de dados.

3 Fundamentação Teórica

3.1 Estruturas de Dados Clássicas

Neste trabalho foram implementadas três estruturas de dados clássicas.

3.1.1 Lista Encadeada

Estrutura onde cada elemento aponta para o próximo, podendo encolher ou crescer à medida que elementos são inseridos ou removidos. Permite inserções e remoções simples, sem a necessidade de realocar memória.

3.1.2 Árvore AVL

Estrutura de árvore binária autobalanceada, na qual a diferença de altura das subárvores é mantida em no máximo um. Isso garante a eficiência da busca, à custa de uma implementação mais complexa, além do custo das operações de autobalanceamento.

3.1.3 Tabela Hash

Estrutura de dados que associa chaves a IDs, permitindo a busca eficiente por ID com base nas chaves. É eficiente para datasets volumosos, mas pode sofrer com colisões.

3.2 Estruturas de Dados Recentes

3.2.1 Skip List

A skip list é uma estrutura probabilística que combina listas com diferentes níveis, tendo uma implementação mais simples que as árvores. A estrutura facilita buscas, inserções e remoções.

3.2.2 Trie

Estrutura que armazena e recupera strings utilizando a forma de uma árvore, onde cada nó representa um caractere e os caminhos, partindo da raiz, formam as strings armazenadas. Assim, é uma estrutura eficiente para busca por prefixos.

3.3 Estrutura de Dados Otimizada

3.3.1 Lista Encadeada Ordenada

Estrutura que funciona como uma otimização da lista encadeada, de forma a manter os elementos ordenados por ID. Essa otimização facilita a busca binária e a listagem ordenada de elementos, mas torna mais custosa a inserção, pois depende da alocação do elemento no lugar correto.

3.4 Operações Realizadas pelas Estruturas

As estruturas suportam as operações de inserção pelo usuário, busca por ID, remoção e listagem. Internamente, todas as estruturas executam funções de inserção, listagem, liberação da memória alocada, carregamento dos dados a partir de um arquivo, busca por filtros, criação de amostras pelo usuário, criação de IDs e remoção por IDs. Além disso, há também funções auxiliares utilizadas como suporte para as operações listadas acima e benchmarks. A única exceção é a estrutura trie, que, por ser feita para strings, não possui funções ligadas a operações feitas com ID. As estruturas possuem também, com exceção da trie, testes padronizados de benchmarks em condições normais e em condições de restrição.

3.5 Métricas de Avaliação e Benchmark

Todas as estruturas possuem testes de benchmark padronizados que avaliam as métricas de tempo de inserção, remoção, busca e uso de memória. A trie aparece novamente como exceção, visto que todos os testes utilizam o ID enquanto a trie trabalha apenas com strings.

3.6 Cenários de Restrição e Simulação

Todas as estruturas também foram avaliadas em cenários de restrição, sendo estes: inserção com memória restrita, inserção com delay, busca com latência, inserção com perda e busca com limite de acessos.

3.7 Características do Dataset Utilizado

O dataset utilizado se chama "Farm Produce Data | 80 years"e está disponível no site kaggle.com.Este, por sua vez, possui 10.262 amostras e nove features: ID, Data, Localização,

Tipo de plantio, Preço por tonelada (Dólares/tonelada), Rendimento (quilogramas por hectare), Produção (toneladas), Área plantada (hectares) e Valor total da safra (Dólares).

4 Metodologia

4.1 Execução dos Benchmarks

As métricas analisadas incluem tempo de inserção, tempo de remoção, tempo de busca e uso de memória. Os testes de benchmark são executados obrigatoriamente com o mesmo arquivo que é solicitado quando o sistema se inicia, garantindo assim uma comparação justa dos resultados. Todas as medições de tempo são feitas com a função `clock_gettime` e com o relógio `CLOCK_MONOTONIC`, garantindo igualdade de condições nos testes. Além disso, os delays e simulações de latência são feitos com `Sleep()` para garantir uniformidade nos testes.

4.2 Simulação de Restrições

As estruturas foram avaliadas em cinco cenários de restrição: inserção com memória restrita, inserção com delay, busca com latência, inserção com perda e busca com limite de acessos. Cada restrição foi implementada de uma forma específica, mas segue padronizada em relação às diferentes estruturas. A limitação de memória foi simulada restringindo a quantidade de memória disponível para alocação. O delay foi inserido artificialmente nas operações de inserção. A latência foi adicionada nas buscas. A perda foi simulada descartando aleatoriamente algumas inserções. O limite de acessos restringiu a quantidade de operações de busca permitidas.

4.3 Tratamento e Utilização do Dataset

Antes da utilização, foram feitas modificações no dataset, incluindo a feature ID para identificação das amostras, tradução dos dados para o português, padronização das medidas e garantia de que não houvesse amostras incompletas. O arquivo final contém 10.262 amostras e nove features, sendo utilizado para alimentar todas as estruturas implementadas durante os testes.

5 Desenvolvimento

5.1 Desenvolvimento do Sistema

O sistema foi inteiramente escrito em linguagem C, optando-se por uma abordagem modular, ou seja, cada estrutura foi implementada em um arquivo .c diferente, mas seguindo o mesmo padrão do projeto. Os menus foram escritos em um único arquivo de nome "menu.c", que chama as funções dos outros arquivos conforme necessário por meio dos headers (.h).

5.2 Implementação das Estruturas de Dados

Cada estrutura possui funções padronizadas para inserção, remoção, busca, filtragem, impressão, carregamento de dados a partir de arquivo e liberação da memória alocada. Para facilitar a identificação, todas as funções recebem um sufixo referente à estrutura a que pertencem.

5.3 Operações Realizadas

Inicialmente, o sistema pede que o usuário insira o nome do arquivo. Após isso, o usuário é encaminhado para o menu abaixo (ver Figura 1), onde pode escolher entre CRUD (1), Modo Benchmark (2) e Sair (0).

5.3.1 Menu CRUD

A opção CRUD (acrônimo em inglês para Create, Read, Update e Delete) leva o usuário a outro menu, onde ele pode interagir com o arquivo a partir das opções: Inserir nova amostra (Árvore AVL), Buscar amostra por ID (Hash), Busca estado/cultura por prefixo (Trie e Hash), Remover uma amostra (Skip list), Listar todas as amostras ordenadas (Lista Ordenada), Voltar ao menu principal. Cada operação é feita pela estrutura considerada mais eficiente e evidenciada entre parênteses.

5.3.2 Menu Benchmark

O Modo Benchmark oferece ao usuário as opções de benchmark que podem ser realizadas pelo sistema (Tempo de inserção, Tempo de remoção, Tempo de busca, Uso de memória, Inserção com memória restrita, Inserção com delay, Busca com latência, Inserção com perda, Busca com limite de acessos, Voltar). Ao selecionar inserção com memória

restrita, inserção com delay e busca com limite de acessos, o sistema requer do usuário, respectivamente, a restrição de memória em MB, o delay em ms e o limite de acessos.

A opção Voltar retorna ao primeiro menu.

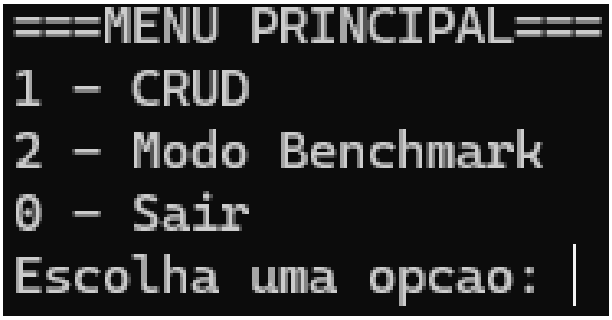

Figura 1 – Menu principal do sistema.

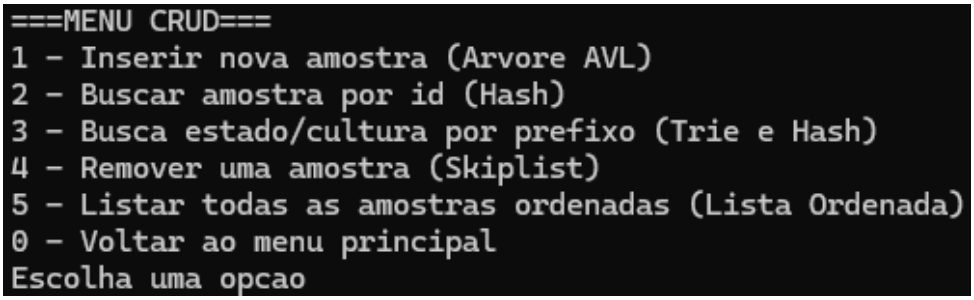

Figura 2 – Menu CRUD: opções para manipulação das amostras.

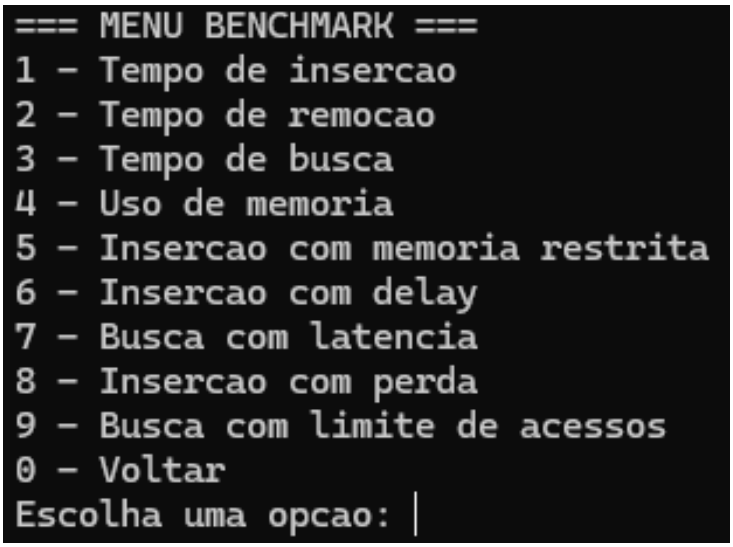

Figura 3 – Menu Benchmark: opções de testes de desempenho.

5.4 Busca por Prefixo com Trie e Hash

A busca por prefixo no sistema é realizada por meio da combinação das estruturas Trie e Hash, aplicado para os campos de estado e cultura das amostras agrícolas.

Para realizar a busca por prefixo, o sistema percorre a Trie seguindo os nós correspondentes aos caracteres do prefixo informado pelo usuário. Ao encontrar o nó referente ao último caractere do prefixo, todas as palavras completas derivadas desse caminho são listadas por meio de uma função recursiva, que percorre os ramos descendentes e imprime todos os termos que compartilham o prefixo buscado.

Após identificar os estados ou culturas correspondentes ao prefixo na Trie, o sistema utiliza a tabela Hash para recuperar e exibir os dados completos das amostras. A Hash armazena todas as amostras agrícolas e permite acesso eficiente por ID ou por filtros. O sistema percorre a tabela Hash e, para cada amostra, verifica se o campo de estado ou cultura corresponde a algum dos termos encontrados na Trie..

6 Resultados

6.1 Apresentação dos Resultados

Os resultados obtidos a partir dos benchmarks foram organizados de modo a permitir a comparação direta entre as seis estruturas de dados implementadas: lista encadeada, lista encadeada ordenada, árvore AVL, tabela hash, skip list e trie.

6.2 Análise dos Resultados: Tempo de Inserção

A tabela a seguir apresenta a média dos tempos de inserção para cada estrutura, considerando os três tamanhos de amostra testados (1.000, 5.000 e 10.000):

Tabela 1 – Tempo de inserção (em segundos) para diferentes estruturas de dados

Estrutura	1000 amostras	5000 amostras	10000 amostras
Lista Encadeada	0.00042183	0.00247840	0.00527183
Lista Encadeada Ordenada	0.00425323	0.06455213	0.28869330
Árvore AVL	0.00180313	0.01014943	0.01996580
Tabela Hash	0.00172187	0.00946230	0.01823053
Skip List	0.00184137	0.00952200	0.01974720

A análise dos resultados indica que a **Lista Encadeada** apresenta o menor tempo médio de inserção, devido à simplicidade da operação de inserção no início da lista, que não requer busca ou ordenação. Por outro lado, a **Lista Ordenada** apresenta o maior tempo médio, refletindo o custo adicional de manter a lista ordenada durante a inserção.

As estruturas **Árvore AVL**, **Hash** e **Skip List** apresentam desempenho semelhante, com tempos de inserção maiores que a lista encadeada simples, mas muito inferiores à lista ordenada.

Em termos de escalabilidade, observa-se que a Lista Ordenada tem crescimento acentuado no tempo de inserção, enquanto as demais estruturas crescem de forma mais controlada: a Lista Encadeada cresce linearmente, e as demais mantêm desempenho próximo ao logarítmico, sendo mais indicadas para sistemas que exigem eficiência com grandes quantidades de dados.

6.3 Análise dos Resultados: Tempo de Remoção

A tabela a seguir apresenta a média dos tempos de remoção para cada estrutura, considerando os três tamanhos de amostra testados (1.000, 5.000 e 10.000):

Tabela 2 – Tempo de remoção (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	0.000645	0.023536	0.152400
Lista Ordenada	0.000834	0.026594	0.152299
Árvore AVL	0.000326	0.001686	0.003684
Hash	0.000030	0.000145	0.000329
Skip List	0.000180	0.001108	0.002520

A análise dos resultados mostra que a **tabela Hash** apresenta o menor tempo médio de remoção entre todas as estruturas testadas. Isso se deve ao acesso direto proporcionado pela função de dispersão, permitindo localizar e remover elementos de maneira extremamente eficiente, mesmo com o aumento do volume de dados.

As estruturas **Árvore AVL** e **Skip List** também apresentam bom desempenho, mantendo tempos baixos e crescimento controlado conforme o número de amostras aumenta, o que evidencia sua boa escalabilidade para operações de remoção em grandes conjuntos de dados. Por outro lado, a **Lista Encadeada** e a **Lista Ordenada** apresentam tempos de remoção significativamente maiores, especialmente para volumes elevados, pois a remoção exige percorrer a lista até encontrar o elemento desejado. Esse comportamento limita a escalabilidade dessas estruturas para operações de remoção em grandes datasets.

6.4 Análise dos Resultados: Tempo de Busca por ID

A tabela a seguir apresenta a média dos tempos de busca por ID para cada estrutura, considerando os três tamanhos de amostra testados (1.000, 5.000 e 10.000):

Tabela 3 – Tempo de busca por ID (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	0.001218	0.040784	0.255299
Lista Ordenada	0.027968	0.135814	0.273515
Árvore AVL	0.000086	0.000555	0.001491
Hash	0.000017	0.000072	0.000162
Skip List	0.000138	0.001013	0.002333

A análise dos resultados mostra que a **tabela Hash** apresenta o menor tempo médio de busca por ID em todos os cenários testados. Esse desempenho é resultado do acesso direto proporcionado pela função de dispersão, permitindo localizar rapidamente qualquer elemento, independentemente do tamanho do conjunto de dados.

A **Árvore AVL** também apresenta excelente desempenho, com tempos de busca baixos e crescimento controlado mesmo com o aumento do número de amostras, evidenciando sua eficiência para operações de busca em grandes volumes de dados. A **Skip List** mantém desempenho satisfatório, com tempos superiores à AVL e à Hash, mas ainda assim

adequados para buscas rápidas. Por outro lado, a **Lista Encadeada** e a **Lista Ordenada** apresentam tempos de busca significativamente maiores, especialmente em grandes volumes de dados, pois a busca exige percorrer a lista até encontrar o elemento desejado. Esse comportamento limita a escalabilidade dessas estruturas para operações de busca em grandes datasets.

6.5 Análise dos Resultados: Uso de Memória

A tabela a seguir apresenta o uso de memória (em bytes) para cada estrutura de dados, considerando os três tamanhos de amostra testados (1.000, 5.000 e 10.000):

Tabela 4 – Uso de memória (em bytes) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	136,000	680,000	1,360,000
Lista Ordenada	985,152	985,152	985,152
'Arvore AVL	112,000	560,000	1,120,000
Hash	96,000	480,000	960,000
Skip List	216,000	1,080,000	2,160,000

A **Lista Ordenada** apresenta uso constante de memória, independentemente do número de amostras, indicando provável pré-alocação de espaço ou limitação estrutural. Já as demais estruturas apresentam crescimento linear do uso de memória conforme o volume de dados aumenta, o que é esperado dado que cada novo elemento demanda espaço adicional.

A **Hash** apresenta o menor consumo de memória entre as estruturas dinâmicas, sendo eficiente mesmo para grandes volumes de dados. A **Árvore AVL** e a **Lista Encadeada** também mostram bom comportamento, com crescimento proporcional ao número de amostras. Por outro lado, a **Skip List** apresenta o maior consumo de memória, reflexo da necessidade de múltiplos ponteiros por elemento para garantir a eficiência das operações de busca e inserção.

6.6 Análise dos Resultados: Inserção com Memória Restrita

A tabela a seguir apresenta o tempo médio de inserção com memória restrita (em segundos) para cada estrutura de dados, considerando três tamanhos de amostra (1.000, 5.000 e 10.000) e limites de memória proporcionais ao maior consumo observado em cada cenário:

Além dos tempos, o benchmark também registra a quantidade de elementos inseridos até o limite de memória ser atingido. Observa-se que, com limites de memória mais restritos, algumas estruturas não conseguem inserir todos os elementos previstos:

Lista Encadeada e **Lista Ordenada**: atingem o limite de memória em 7.710 elementos, independentemente do número de amostras.

Tabela 5 – Tempo de inserção com memória restrita (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	0.00413	0.00443	0.00545
Lista Ordenada	0.30405	0.30583	0.31591
Árvore AVL	0.01852	0.01964	0.02090
Hash	0.01919	0.01848	0.01919
Skip List	0.00952	0.01003	0.01952

Árvore AVL e **Hash**: atingem o limite em 9.362 elementos para 1.000 e 5.000 amostras, e inserem todos os 10.000 elementos no último cenário.

Skip List: é a estrutura que atinge o limite mais cedo, com apenas 4.853 elementos nas duas primeiras situações e 9.707 elementos no último caso.

A análise dos resultados mostra que, sob restrição de memória, a **Skip List** é a mais penalizada, pois consome mais memória por elemento devido à sua estrutura interna com múltiplos ponteiros. Já a **Árvore AVL** e a **Hash** apresentam melhor aproveitamento do espaço, conseguindo inserir mais elementos antes de atingir o limite. A **Lista Ordenada** mantém o maior tempo de inserção, mesmo com memória restrita, devido à necessidade de busca sequencial para cada inserção.

6.7 Análise dos Resultados: Tempo de Inserção com Delay

A tabela a seguir apresenta o tempo médio de inserção (em segundos) para cada estrutura de dados, considerando a execução do benchmark com delay fixo de 5 ms por inserção e três tamanhos de amostra (1.000, 5.000 e 10.000):

Tabela 6 – Tempo de inserção com delay (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	11.08	53.43	105.18
Lista Ordenada	10.81	52.91	103.92
Árvore AVL	11.11	51.99	113.12
Hash	11.03	52.03	113.21
Skip List	10.60	52.42	112.44

A análise dos resultados mostra que, com a introdução de um delay fixo de 5 ms por inserção, os tempos médios de todas as estruturas tornam-se bastante próximos, independentemente do tipo de estrutura utilizada. Isso ocorre porque o tempo de espera artificial passa a dominar o tempo total do benchmark, tornando irrelevantes as diferenças de desempenho intrínsecas entre as estruturas para a operação de inserção. Pequenas variações entre as estruturas podem ser atribuídas ao overhead específico de cada implementação, mas não alteram o padrão geral.

Nesses casos, a eficiência algorítmica das estruturas é mascarada pelo tempo de espera imposto, e a diferença de desempenho só volta a ser significativa quando o delay artificial é removido ou reduzido.

6.8 Análise dos Resultados: Tempo de Busca com Latência

A tabela a seguir apresenta o tempo médio de busca com latência para cada estrutura, considerando os três tamanhos de amostra testados (1.000, 5.000 e 10.000):

Tabela 7 – Tempo de busca com latência (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	10.54	53.24	103.94
Lista Ordenada	10.43	52.54	110.23
Árvore AVL	10.52	51.85	117.93
Hash	10.63	51.95	114.47
Skip List	10.50	53.92	110.14

A análise dos resultados mostra que, sob a condição de busca com latência, todas as estruturas apresentam tempos elevados e bastante próximos, independentemente do tipo de estrutura utilizada.Mesmo assim, observa-se que para 1.000 e 5.000 amostras, as estruturas Árvore AVL e Hash tendem a apresentar levemente menores tempos médios em relação às listas, mas a diferença é pequena. Para 10.000 amostras, as variações entre as estruturas se mantêm dentro de uma faixa semelhante, reforçando que, sob alta latência, a escolha da estrutura de dados tem impacto reduzido no tempo total de operação.

6.9 Análise dos Resultados: Tempo de Inserção com Perda

A tabela a seguir apresenta o tempo médio de inserção com perda (em segundos) para cada estrutura de dados, considerando três tamanhos de amostra (1.000, 5.000 e 10.000):

Tabela 8 – Tempo de inserção com perda (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	0.00061	0.00285	0.00510
Lista Ordenada	0.00390	0.06531	0.21104
Árvore AVL	0.00189	0.01049	0.01773
Hash	0.00188	0.00989	0.01564
Skip List	0.00188	0.01041	0.02359

A análise dos resultados demonstra que a **Lista Encadeada** mantém o menor tempo médio de inserção com perda, devido à simplicidade da operação de inserção no início da

lista. As estruturas **Hash**, **Árvore AVL** e **Skip List** também apresentam tempos baixos e crescimento controlado, evidenciando boa eficiência mesmo quando parte dos elementos é descartada durante o processo de inserção.

Por outro lado, a **Lista Ordenada** se destaca pelo tempo de inserção significativamente maior, principalmente para volumes maiores de dados. Isso ocorre porque, mesmo com perda, é necessário buscar a posição correta para cada elemento a ser inserido, resultando em complexidade O(n) para cada operação.

6.10 Análise dos Resultados: Tempo de Busca com Limite de Acessos

A tabela a seguir apresenta o tempo médio de busca com limite de acessos (em segundos) para cada estrutura de dados, considerando três tamanhos de amostra (1.000, 5.000 e 10.000) e limite de 3 acessos por busca:

Tabela 9 – Tempo de busca com limite de acessos (em segundos) para diferentes estruturas de dados

Estrutura	**1000 amostras**	**5000 amostras**	**10000 amostras**
Lista Encadeada	0.000058	0.000063	0.000127
Lista Ordenada	0.000013	0.000069	0.000136
Árvore AVL	0.000034	0.000175	0.000351
Hash	0.000052	0.000220	0.000436
Skip List	0.000017	0.000147	0.000308

A análise dos resultados evidencia que, ao impor um limite de acessos por busca, o tempo de execução se mantém extremamente baixo para todas as estruturas, independentemente do tamanho do conjunto de dados. Isso ocorre porque a busca é interrompida rapidamente, após apenas três acessos, reduzindo drasticamente o tempo total da operação em relação à busca tradicional.

As diferenças entre as estruturas tornam-se praticamente irrelevantes nesse cenário. Pequenas variações nos tempos médios podem ser atribuídas ao overhead de controle de cada implementação, mas o padrão geral é de desempenho nivelado.

6.11 Análise Geral dos Resultados

De modo geral, as estruturas que oferecem acesso direto ou operações com complexidade logarítmica se destacam em eficiência, enquanto as listas encadeadas apresentam limitações de escalabilidade para grandes volumes de dados.

A tabela hash mostrou-se consistentemente a estrutura mais eficiente para remoção e busca por ID, graças ao acesso direto proporcionado pela função de dispersão, mantendo

tempos praticamente constantes mesmo com o aumento do número de amostras. A árvore AVL também apresentou excelente desempenho, com tempos baixos e crescimento controlado, evidenciando sua eficiência para operações que exigem ordenação e acesso rápido. A skip list manteve desempenho satisfatório, situando-se entre a AVL e as listas, e pode ser uma alternativa interessante em cenários que exigem simplicidade de implementação e bom desempenho.

Por outro lado, as listas encadeada simples e ordenada apresentaram crescimento acentuado nos tempos de inserção, remoção e busca, tornando-se pouco indicadas para aplicações com grandes volumes de dados ou que exigem alta performance nessas operações. A lista ordenada, em particular, teve desempenho inferior devido ao custo adicional de manter a ordenação durante as inserções e buscas.

Nos cenários de restrição, como inserção com memória limitada, delay ou latência artificial, observou-se que a eficiência algorítmica das estruturas se torna menos relevante quando o fator limitante é externo (por exemplo, tempo de espera imposto ou limite de acessos). Nesses casos, todas as estruturas apresentaram tempos elevados e muito próximos, indicando que, sob restrições severas, a escolha da estrutura de dados tem impacto reduzido no tempo total de operação.

Por fim, o uso de memória variou conforme a estrutura, com a hash apresentando o menor consumo entre as dinâmicas e a skip list exigindo mais espaço devido à sua estrutura interna. O comportamento linear de uso de memória foi observado na maioria das estruturas, exceto na lista ordenada, que apresentou uso constante devido à provável pré-alocação.

Em resumo, para sistemas que exigem operações rápidas de busca, inserção e remoção em grandes volumes de dados, a tabela hash e a árvore AVL são as escolhas mais indicadas. Em cenários com restrições externas severas, a diferença entre as estruturas diminui, e outros fatores, como simplicidade de implementação e consumo de memória, podem se tornar decisivos.

6.12 Conclusão

O trabalho atingiu seus objetivos ao implementar e comparar seis estruturas de dados aplicadas ao gerenciamento de dados agrícolas em C, avaliando eficiência em diferentes cenários. Os resultados dos benchmarks mostraram que a tabela hash é a estrutura mais eficiente para operações rápidas de busca e remoção por ID, enquanto a árvore AVL também apresentou ótimo desempenho, especialmente quando a ordenação é importante. A skip list mostrou-se uma alternativa eficiente e simples de implementar.

Por outro lado, as listas encadeada simples e ordenada apresentaram desempenho inferior em grandes volumes de dados, principalmente devido ao crescimento do tempo de operação. Em cenários com restrições externas, como delay ou latência, as diferenças

entre as estruturas diminuem, pois o tempo total passa a ser dominado pela restrição imposta.

Em resumo, a escolha da estrutura de dados deve considerar o tipo de operação mais frequente, o volume de dados e possíveis restrições do ambiente. A análise realizada fornece um guia prático para selecionar a estrutura mais adequada em sistemas embarcados e aplicações que exigem manipulação eficiente de grandes conjuntos de dados.