João Felipe Veras Dantas Rocha / DRE:117202753
Lucas dos Santos Melo / DRE:123312281
Rafael Augusto Santos / DRE:124155331
Rayan Lucas de Paula Carvalhal / DRE:124180742
Link para o GitHub dos arquivos com os códigos, assim como o arquivo de texto: https://github.com/jfvdrocha/BD-2025.1-tabela-hash-e-deduplicacao

# Tabela Hash e Deduplicação

Professor: Milton Ramos Ramirez

Rio de Janeiro

2025

Comentários sobre o Desenvolvimento da Tabela Hash e Deduplicação de Dados:

O desenvolvimento da solução para o problema de deduplicação com base em uma Tabela Hash teve como objetivo aplicar conceitos de estruturas de dados, hashing e programação orientada a objetos para reduzir a complexidade do tratamento de dados em CSVs, problema comum em aplicações de Ciência de Dados.

Estruturas de Dados Utilizadas

A estrutura central do projeto foi uma Tabela Hash com encadeamento externo, implementada por meio de uma lista principal (self.tabela) composta por listas internas (buckets) para armazenar colisões. Essa abordagem foi escolhida por sua simplicidade de implementação e boa eficiência média para inserção e busca. A classe Registro foi criada para representar cada linha do CSV como um objeto, encapsulando os dados e facilitando a comparação de duplicatas com base em uma chave única (como CPF ou DRE).

Divisão de Módulos e Funções

O projeto foi modularizado em três partes principais:

- Classe Registro: encapsula os dados de cada linha do CSV e permite comparação de duplicatas.
- Classe TabelaHash: implementa a estrutura da tabela com métodos de inserção, busca, remoção e exportação.
- Função remover_duplicatas_csv: orquestra a leitura do arquivo CSV, inserção dos dados na tabela e gravação do arquivo deduplicado.

A separação em classes e funções seguiu os princípios da coesão funcional, modularidade e clareza, facilitando manutenção e testes.

Complexidade de Tempo e Espaço

Tempo médio de inserção e busca: O(1), graças ao uso de hashing com encadeamento.

Complexidade total da deduplicação: O(n), onde n é o número de registros no CSV — consideravelmente melhor do que métodos baseados em ordenação (O(n log n)).

Espaço: O(n), já que cada elemento do dataset é carregado na memória.

Esse ganho de performance é fundamental quando se lida com grandes volumes de dados.

## Problemas e Observações

Durante o desenvolvimento, alguns pontos exigiram atenção:

A escolha da função de hashing influencia diretamente o desempenho. Neste projeto, optamos pela função baseada na soma dos códigos ASCII dos caracteres, com módulo do tamanho da tabela. Apesar de simples, ela pode gerar colisões em datasets mal distribuídos.

Gestão de colisões foi feita por listas (encadeamento externo), que embora robusta, pode se degradar em desempenho se muitas colisões forem concentradas em poucos índices.

Conversão e preservação dos tipos de dados (como números inteiros ou strings) exigiu cuidado na manipulação de dicionários no Python, especialmente ao reescrever o CSV final.

A estrutura é dependente da definição clara da chave de deduplicação (campo do CSV), que deve ser corretamente escolhida pelo usuário.

## Conclusão

A solução proposta demonstrou ser eficaz e eficiente para deduplicação de dados em arquivos CSV. A abordagem com tabela hash reduziu significativamente a complexidade computacional, garantindo resultados consistentes em tempo linear. Além disso, a organização orientada a objetos permitiu uma estrutura clara, reutilizável e expansível, adequada a aplicações reais em projetos de Ciência de Dados e Engenharia de Produção. O projeto reforça a importância de estruturas de dados clássicas na otimização de tarefas práticas do cotidiano computacional.