1. Introdução

Esta tarefa teve como objetivo o desenvolvimento de uma **Tabela Hash com encadeamento exterior** para eliminar duplicatas de registros contidos em um arquivo `.CSV`, simulando uma etapa comum do tratamento de dados em Ciência de Dados. A proposta consiste em aplicar conceitos de estruturas de dados para melhorar a eficiência da deduplicação de registros, reduzindo a complexidade de O(n log n), presente em métodos baseados em ordenação, para O(n), ao utilizar hashing.

2. Estruturas de Dados Utilizadas

- **Classe `Node`**: Representa um nó de uma lista encadeada em cada posição da tabela hash. Cada nó armazena uma **chave** (como um CPF ou ID), um **valor** (como um nome) e um ponteiro para o próximo nó.
- **Classe `HashTable`**: Implementa a estrutura principal da tabela hash. Internamente, utiliza um vetor de ponteiros (`vector<Node*>`) para armazenar as listas encadeadas, aplicando **encadeamento exterior** para resolução de colisões.
- **Arquivo CSV**: Utilizado como dataset de entrada. Cada linha é lida e inserida na tabela, sendo que apenas chaves não duplicadas são armazenadas.

3. Organização do Código

O código foi organizado em um único arquivo `hash_table.cpp`, contendo:

- A definição da classe `Node` para representar os elementos encadeados.
- A implementação da classe `HashTable`, com as operações fundamentais:
  - Inserção (`insert`)
  - Busca (`search`)
  - Remoção (`remove`)
  - Impressão (`print`)
- A função de hashing (`hashDivisao`) baseada no método da divisão.
- A função `eliminateDuplicate`, que realiza a leitura do arquivo e popula a tabela hash.
- A função `main`, que executa a deduplicação e imprime os registros únicos.

4. Funções e Métodos

Método `insert(key, value)`

- Insere um par (chave, valor) na tabela, se a chave ainda não estiver presente.
- Aplica a função de dispersão para determinar o índice.
- Utiliza encadeamento exterior para inserir em caso de colisão.

Método `search(key)`

- Busca a chave na posição apropriada da tabela.
- Retorna o valor associado ou string vazia caso não encontre.

Método `remove(key)`

- Remove a chave e o valor da tabela se encontrados.
- Realiza ajuste dos ponteiros da lista encadeada, liberando memória.

Método `print()`

- Percorre toda a tabela e imprime os pares (chave, valor) únicos, sem duplicatas.

Função `hashDivisao(string key)`

- Converte a chave para `unsigned long long` e aplica o operador módulo com o tamanho da tabela.
- Método simples, porém eficiente quando a chave é numérica.

Função `eliminateDuplicate()`

- Lê o arquivo CSV linha a linha.
- Insere cada linha na tabela hash utilizando a chave como indexador.
- Garante que duplicatas não sejam inseridas.

5. Complexidade de Tempo e Espaço

Operação	Complexidade Média	Complexidade Pior Caso
Inserção	O(1)	O(n) (colisões extremas)

Busca	O(1)	O(n)
Remoção	O(1)	O(n)
Leitura CSV	O(n)	O(n)
Impressão final	O(n)	O(n)

A complexidade geral do processo de deduplicação foi O(n), muito mais eficiente do que métodos tradicionais baseados em ordenação.

6. Problemas e Observações

- A função `hashDivisao` assume que a chave é numérica. Isso limita o código a datasets com chaves numéricas (como CPF, ID). Para maior generalidade, seria ideal adaptar a função de dispersão para strings genéricas.
- O método `insert` evita duplicatas ao verificar previamente se a chave já está presente usando o método `search`. Isso assegura a consistência dos dados armazenados.
- A estrutura com encadeamento externo funcionou bem mesmo para colisões múltiplas. O uso de listas evitou necessidade de rehashing imediato.
- O programa espera que o CSV tenha duas colunas por linha. Para datasets com mais colunas ou formatos diferentes, seria necessária uma adaptação do parser.

7. Conclusão

O projeto atendeu plenamente aos objetivos propostos. Foi possível realizar a deduplicação de registros de forma eficiente utilizando uma **tabela hash com encadeamento exterior**, mantendo desempenho O(n) mesmo em arquivos com diversas entradas.

A estrutura orientada a objetos adotada conferiu modularidade e clareza ao código, sendo facilmente adaptável para diferentes funções de dispersão e formatos de dados. Com pequenas melhorias, como suporte a chaves compostas e tratamento mais robusto de entrada, a solução pode ser aplicada em sistemas reais de pré-processamento de dados.