Relatório – Análise de Tabelas Hash em Java

1. Introdução

Este trabalho tem como objetivo implementar e analisar o desempenho de diferentes técnicas de **tabelas hash** em Java. As tabelas hash são estruturas de dados fundamentais para busca rápida, e o tratamento de colisões é crucial para manter a eficiência. Neste projeto, foram implementadas três abordagens distintas:

1. **Hash Linear** – rehashing linear.
2. **Hash Duplo** – hash duplo como forma de reduzir colisões.
3. **Encadeamento Separado** – cada posição do vetor armazena uma lista encadeada.

O projeto segue rigorosamente os requisitos fornecidos, incluindo geração de dados com **seed fixa**, análise de métricas como **tempo de inserção**, **colisões**, **gaps**, **maiores listas encadeadas**, e comparação do desempenho entre diferentes funções e tamanhos de tabela.

2. Estrutura do Código

O projeto foi estruturado em pacotes e classes para modularidade e clareza:

- **hash/Registro.java**: Classe que representa um registro com código numérico de 9 dígitos. Inclui métodos `equals` e `hashCode`.
- **hash/HashTable.java**: Interface que define métodos essenciais para qualquer tabela hash: `inserir`, `buscar` e `getNumeroColisoes`.
- **hash/HashTableLinear.java**: Implementa tabela hash com **linear probing**.
- **hash/HashTableDuplo.java**: Implementa tabela hash com **double hashing**.
- **hash/HashTableEncadeamento.java**: Implementa tabela hash com **encadeamento separado**, utilizando listas encadeadas.
- **hash/ListaEncadeada.java**: Estrutura de apoio para listas encadeadas no encadeamento.
- **hash/Main.java**: Classe principal responsável por gerar dados, inserir, buscar, medir métricas e exportar resultados para análise gráfica.

3. Escolha dos Tamanhos da Tabela

Foram escolhidos três tamanhos de vetor para analisar o impacto no desempenho e nas colisões:

Tipo	Tamanho
Pequena	150.000
Média	1.500.000
Grande	15.000.000

A variação de **x10** entre cada tamanho permite observar claramente como o desempenho escala com o aumento da tabela.

4. Conjuntos de Dados

Três conjuntos de dados foram gerados aleatoriamente com **seed fixa** para garantir que os mesmos elementos fossem usados em todas as funções hash:

Conjunto	Nº de registros
1	100.000
2	1.000.000
3	10.000.000

Cada registro possui **9 dígitos**, garantindo uniformidade para validação e comparabilidade.

5. Funções Hash Implementadas

5.1 Hash Linear

- Função: `h(x) = x % tamanhoVetor`
- Colisões resolvidas por **linear probing**, ou seja, deslocamento unitário até encontrar espaço vazio.

5.2 Hash Duplo

- Funções:
  - `h1(x) = x % tamanhoVetor`
  - `h2(x) = 1 + (x / tamanhoVetor) % (tamanhoVetor - 1)`
- Rehashing: `pos = h1 + i * h2`
- Reduz clustering comparado à abordagem linear.

5.3 Encadeamento Separado

- Função: `h(x) = x % tamanhoVetor`
- Colisões são armazenadas em **listas encadeadas**.
- Permite múltiplos elementos na mesma posição sem precisar de deslocamento.

💡 Todas as funções foram pesquisadas e implementadas sem usar funções prontas ou slides do curso.

6. Métricas Avaliadas

Para cada combinação de tabela, tamanho e conjunto de dados, foram medidas:

1. **Tempo de inserção** (ms)
2. **Número de colisões**
3. **Tempo de busca** (ms)
4. **Gaps entre elementos** – menor, maior e médio
5. **Maiores listas encadeadas** (apenas para encadeamento)

As métricas foram **exportadas para CSV** para facilitar a análise gráfica.

7. Resultados

Exemplo de resultados para tabela grande (15.000.000) com 10.000.000 de registros:

Tabela	Inserção (ms)	Colisões	Busca (ms)	Gaps / Maiores listas

Linear	635	10.231.319	659	1 / 23 / 1
Duplo	427	6.535.216	414	1 / 17 / 1
Encadeamento	4.358	2.700.700	923	9 / 8 / 8

Observações:

- **Hash Duplo** foi mais eficiente na maioria das métricas.
- **Encadeamento** apresentou menor número de colisões, mas tempo de inserção maior.
- **Hash Linear** simples, mas sofre clustering em tabelas grandes.

8. Análise Comparativa

- **Eficiência de inserção**: Hash Duplo > Linear > Encadeamento
- **Número de colisões**: Encadeamento < Duplo < Linear
- **Tempo de busca**: Duplo > Linear > Encadeamento
- **Escalabilidade**: Hash Duplo mantém melhor desempenho em tabelas grandes.

9. Conclusão

- A escolha da função hash e do tamanho da tabela impacta diretamente o desempenho.
- **Hash Duplo** é a melhor opção em grandes volumes, equilibrando colisões e tempo de busca.
- **Encadeamento Separado** é mais seguro contra colisões, mas menos eficiente em tempo de inserção.
- **Hash Linear** é simples, mas não escala tão bem para grandes volumes.
- O uso de **seed fixa** garante resultados replicáveis, essencial para validação.

10. Bônus (Opcional) – Análise de Memória

- **Linear e Duplo**: ocupam memória proporcional ao tamanho do vetor.

- **Encadeamento**: overhead adicional devido às listas encadeadas, maior uso de memória em grandes tabelas.
- **Recomendação**: em tabelas muito grandes, Hash Duplo oferece melhor custo-benefício entre tempo e memória.

11. Como Executar

```
git clone <link-do-repo>
javac hash/*.java
java hash.Main
```

- Resultados no console e exportados para `metricas.csv`.