**Relatório — Trabalho de Estrutura de Dados: Tabela Hash**

**Alunos:** Stephanny e Matheus

**Introdução**

O objetivo deste trabalho foi desenvolver uma tabela hash em Java, utilizando listas encadeadas para tratamento de colisões. O principal foco foi entender como diferentes funções hash impactam no desempenho da tabela, especialmente na distribuição dos dados e na quantidade de colisões.

**Desenvolvimento**

**Funcionamento da Tabela Hash**

A tabela hash é uma estrutura que associa chaves (neste caso, nomes) a índices numéricos, através de uma função hash. Cada chave é armazenada no índice calculado. Quando ocorre de duas ou mais chaves gerarem o mesmo índice (chamado de colisão), os dados são armazenados em uma lista encadeada dentro desse índice.

**Implementação**

Foram criadas duas tabelas hash, cada uma utilizando uma função hash diferente. Ambas possuem 32 posições (índices) e usam listas encadeadas para armazenar múltiplos elementos em caso de colisões.

**Estruturas utilizadas:**

* **Classe MyArrayList:** Simula uma lista dinâmica.
* **Classe MyLinkedList:** Implementa a lista encadeada para tratar colisões.
* **Classe Node:** Define os nós da lista encadeada.
* **Classe HashTable:** Classe abstrata que define as operações principais.
* **Classe HashTableHash1:** Implementa a primeira função hash (soma dos caracteres).
* **Classe HashTableHash2:** Implementa a segunda função hash (multiplicação com peso).
* **Classe Main:** Responsável pela leitura do arquivo e execução dos testes.

**Explicação das Funções Hash**

**🔸 Hash 1 — Soma dos caracteres**

* **Lógica:** Soma os valores numéricos (ASCII) de cada caractere da palavra e aplica o módulo com o tamanho da tabela.
* **Problema:** Não leva em consideração a ordem dos caracteres. Palavras como "LIA" e "IAL" geram o mesmo índice, causando colisões.
* **Ponto positivo:** É extremamente rápida no cálculo.
* **Ponto negativo:** Apresenta má distribuição dos dados e alto número de colisões.

**🔹 Hash 2 — Multiplicação com peso (técnica mais robusta)**

* **Lógica:** Começa com um valor inicial e, para cada caractere, multiplica o valor atual por 31 (número primo) e soma o código do caractere. Por fim, aplica o módulo com o tamanho da tabela.
* **Motivação:** O uso de 31, que é um número primo, ajuda a distribuir melhor os dados, reduzindo colisões.
* **Ponto positivo:** A distribuição dos dados é muito mais uniforme, com baixíssimo número de colisões.
* **Ponto negativo:** O cálculo é um pouco mais complexo, mas isso não afeta significativamente o desempenho geral.

**Análise dos Resultados**

Ao executar o programa com um arquivo contendo uma lista de nomes, os seguintes comportamentos foram observados:

* A tabela com **Hash 1** apresentou uma grande concentração de dados em poucos índices. Isso gerou muitas colisões, aumentando o tamanho das listas encadeadas e, consequentemente, o tempo de busca dentro delas.
* A tabela com **Hash 2** teve uma distribuição bem mais equilibrada. A maioria dos índices continha poucos elementos ou até apenas um, o que reduziu drasticamente o número de colisões.

**Por que uma é mais eficiente que a outra?**

A função **Hash 2** é mais eficiente justamente porque considera tanto o valor dos caracteres quanto a ordem em que eles aparecem. Isso faz com que nomes diferentes sejam convertidos em índices bem distribuídos pela tabela, evitando que muitos elementos fiquem acumulados no mesmo local.

Apesar de o cálculo da Hash 2 ser ligeiramente mais complexo, isso não interfere no desempenho final. Pelo contrário, a economia de tempo nas buscas compensa completamente o custo extra no cálculo do índice. Já a Hash 1, apesar de ser extremamente simples e rápida no cálculo, gera muitas colisões, o que prejudica muito o desempenho geral da tabela.

**Conclusão**

Este trabalho deixou claro a importância da escolha correta da função hash no desenvolvimento de algoritmos e estruturas de dados. Uma função hash bem projetada faz toda a diferença na eficiência da tabela, tanto na inserção quanto na busca dos dados.

Ficou evidente que, na prática, vale mais a pena optar por uma função hash com cálculo um pouco mais elaborado (como a Hash 2), pois ela garante uma distribuição muito melhor dos dados e um desempenho muito superior.

O aprendizado sobre colisões, tabelas hash e a influência das funções hash foi de extrema importância para nosso desenvolvimento no entendimento de estruturas de dados.