ATIVIDADE RA3 - TABELA HASH

Este projeto implementa e avalia o desempenho das funções hash "Multiplicação" "Resto" e "Dobramento" em tabelas com tratamento de colisões utilizando listas encadeadas. A implementação tem o objetivo de testar a eficiência de inserção e busca em conjuntos de dados grandes.

Estrutura do Projeto

Gerador De Dados

Gera uma lista de registros com códigos representados como strings de 9 dígitos.

- Método: gerar(int quantidade, long seed)
    - Parâmetros:
        - quantidade: Número de registros a serem gerados.
        - seed: Semente para o gerador. (garante resultados iguais em gerações seguintes).
    - Retorno: Lista de objetos Registro.

Lista Encadeada

Uma lista simples usada para armazenar registros em caso de colisões na tabela hash.

- Métodos principais:
    - inserir(Registro r): Insere um novo registro no início da lista.
    - buscar(String codigo): Busca um registro no código.
        - Conta o número de comparações feitas.
    - isEmpty(): Verifica se a lista está vazia.

Tabela Hash Encadeada

Implementa a tabela com um array de listas encadeadas para o tratar colisões.

- Métodos principais:
    - inserir(Registro r, BiFunction<String, Integer, Integer> hashFunc):
        - Insere um registro na tabela usando uma função hash personalizada.
        - Conta colisões.

    - buscar(String codigo, BiFunction<String, Integer, Integer> hashFunc):
        - Busca um registro pelo código.
        - Retorna o número de comparações realizadas.

Funções Hash

As funções hash estão localizadas no pacote HashFunções e incluem:

- Hash Resto:
    - Calcula o resto da divisão do código pelo tamanho da tabela.
- Hash Multiplicação:
    - Usa o método da multiplicação com um fator irracional para dispersão.
- Hash Dobramento:
    - Soma os dígitos do código e tira o módulo do tamanho da tabela.

Avaliador de Desempenho

Realiza testes de desempenho com inserções e buscas aleatórias. Mede:

- Tempo de inserção.
- Número de colisões.
- Tempo e número médio de comparações durante buscas.

Classe Principal

Executa os testes de desempenho para diferentes tamanhos de tabela e conjuntos de dados.

- Configurações:
    - Tamanhos da tabela: {1000, 10000, 100000}
    - Quantidade de registros: {1_000_000, 5_000_000, 20_000_000}
    - Sementes para geração de dados: {111, 222, 333}
- Testa cada função hash com cada combinação de configuração.

Como Executar

1. Certifique-se que todas as classes estão compiladas.
2. Execute a classe Main.

Resultados Esperados

- Saída formatada com:
    - Tempo de inserção (ms).
    - Número de colisões.
    - Tempo e número médio de comparações durante buscas.

Escolhas tomadas

As três variações de função hash mencionadas anteriormente foram escolhidas devido ao exemplo fornecido, mas também, após os testes de desempenho, foi observado que resultados tão diferentes como os apresentados serviram para analisarmos o quão ampla a diferença da eficiência entre métodos é quando contabilizamos um grande conjunto de dados.

Justificativa dos Resultados

Multiplicação

Pudemos observar que a função hash baseada em multiplicação apresentou maior tempo de inserção devido à complexidade das operações matemáticas envolvidas. E o uso de números de ponto flutuante que aumentam o uso da CPU especialmente em grandes volumes de dados.

Dobramento

Já a função de dobramento demonstrou maior número de colisões e comparações médias devido à má distribuição dos hashes gerados. Isso acontece pois ao somar os dígitos, ela gera muitos valores repetidos. Esse comportamento reduz significativamente o desempenho das operações.

Resto

A função baseada em resto apresentou desempenho consistente e eficiente em termos de tempo de inserção, número de colisões e comparações durante buscas. tornando-a uma boa escolha para tabelas hash de tamanhos variados.