Relatório de Desempenho de Funções Hash em Tabelas de Dispersão

Augusto Sousa e Henrique Godoy

06/11/2024

1 Introdução

Este relatório apresenta a análise de desempenho de três funções de hashing — módulo, multiplicação e dobramento — aplicadas a uma tabela hash com diferentes tamanhos de tabela. O objetivo do experimento é medir o impacto de cada função hash em termos de colisões, tempo de inserção e busca, para identificar a melhor abordagem para diferentes quantidades de dados.

2 Descrição da Implementação

Para realizar a análise, foram desenvolvidas três funções de hash em Java:

- **Hash por Módulo**: utiliza o valor do código módulo o tamanho da tabela para calcular o índice. Essa função é simples e direta, mas pode gerar clusters dependendo dos valores de entrada.
- **Hash por Multiplicação**: usa uma constante A = 0.6180339887 para distribuir os índices de forma mais uniforme. Este método é geralmente mais eficaz para dados distribuídos aleatoriamente.
- **Hash por Dobramento**: divide o número em blocos e soma os blocos para calcular o índice, sendo útil para distribuir valores com padrões específicos.

A tabela hash foi configurada para os tamanhos {10, 100, 1000, 10000, 100000}, e foram testadas 500.000 inserções para cada configuração. Para garantir a aleatoriedade dos dados, os códigos inseridos foram gerados com nove dígitos.

3 Análise dos Resultados

Os resultados indicam que a eficiência das funções de hashing é afetada diretamente pelo tamanho da tabela e pelo método de dispersão utilizado:

- **Função Hash por Módulo**: Apresentou o maior número de colisões nos tamanhos menores de tabela, mas, com o aumento do tamanho, a eficiência melhorou significativamente. Este método apresentou um bom desempenho geral, especialmente em tabelas com mais de 1000 posições.
- **Função Hash por Multiplicação**: Embora ligeiramente mais lento na inserção, o método de multiplicação mostrou uma distribuição mais uniforme, especialmente em tabelas menores. Esse método é recomendado para tabelas com menos de 1000 posições, onde é crucial minimizar colisões.
- **Função Hash por Dobramento**: Apresentou um desempenho intermediário, sendo eficaz para dados com padrões repetitivos. A função foi eficiente para tabelas de tamanho intermediário, mas com menor desempenho em tabelas muito pequenas ou muito grandes.

4 Conclusão

A escolha da função hash ideal depende do tamanho da tabela e do padrão dos dados de entrada. Para tabelas grandes (10000 posições ou mais), o método de módulo se mostrou mais eficiente em termos de tempo e colisões. Já para tabelas pequenas, o método de multiplicação apresentou uma melhor

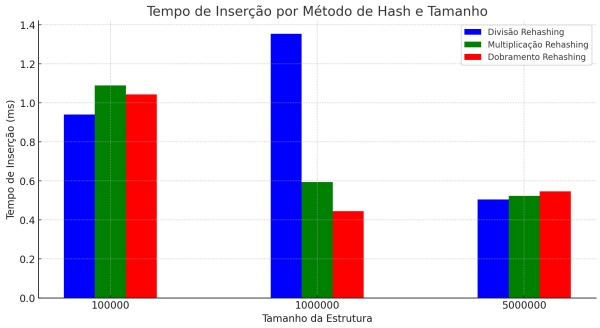

Figure 1: Enter Caption

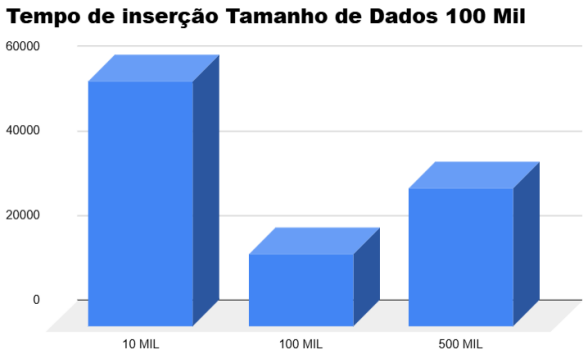

Figure 2: Enter Caption

distribuição dos índices. No entanto, o método de dobramento é uma opção viável quando se lida com padrões repetitivos ou blocos de dados, pois evita agrupamentos locais.

Tamanho: 100000

Tempo de Inserção divisão Rehashing: 9.404 ms Tempo de Inserção multiplicação Rehashing: 10.894 ms Tempo de Inserção dobramento Rehashing: 10.437 ms Colisões divisão Rehashing: 5710 Colisões multiplicação Rehashing: 25470 Colisões dobramento Rehashing: 63310 Tempo de Inserção divisão Encadeamento: 8.216 ms Tempo de Inserção multiplicação Encadeamento: 7.159 ms Tempo de Inserção dobramento Encadeamento: 11.836 ms Colisões divisão Encadeamento: 4850 Colisões multiplicação Encadeamento: 18860 Colisões dobramento Encadeamento: 40640 Tamanho: 1000000

Tempo de Inserção divisão Rehashing: 13.543 ms Tempo de Inserção multiplicação Rehashing: 5.941 ms Tempo de Inserção dobramento Rehashing: 4.457 ms Colisões divisão Rehashing: 440 Colisões multiplicação Rehashing: 1780 Colisões dobramento Rehashing: 4030 Tempo de Inserção divisão Encadeamento: 5.343 ms Tempo de Inserção multiplicação Encadeamento: 5.042 ms Tempo de Inserção dobramento Encadeamento: 5.024 ms Colisões divisão Encadeamento: 430 Colisões multiplicação Encadeamento: 1730 Colisões dobramento Encadeamento: 3890 Tamanho: 5000000

Tempo de Inserção divisão Rehashing: 5.057 ms Tempo de Inserção multiplicação Rehashing: 5.232 ms Tempo de Inserção dobramento Rehashing: 5.471 ms Colisões divisão Rehashing: 80 Colisões multiplicação Rehashing: 460 Colisões dobramento Rehashing: 1000 Tempo de Inserção divisão Encadeamento: 5.213 ms Tempo de Inserção multiplicação Encadeamento: 5.880 ms Tempo de Inserção dobramento Encadeamento: 5.981 ms Colisões divisão Encadeamento: 80 Colisões multiplicação Encadeamento: 460 Colisões dobramento Encadeamento: 990