RA3 - Trabalho Tabela Hash

Marcelo Wzorek Filho Xuekai Qian Raphael de Oliveira Soares
Rafael Leal Machado

December 10, 2023

Abstract

Neste trabalho, implementamos e analisamos o desempenho de diferentes tabelas hash em Java, explorando a influência de diversos fatores, como o tamanho da tabela, a função de hash e o método de resolução de colisões. Realizamos experimentos abrangentes e apresentamos resultados que demonstram as vantagens e desvantagens de cada abordagem.

1 Introdução

As tabelas hash desempenham um papel importante na ciência da computação, oferecendo uma estrutura de dados eficaz para a pesquisa, recuperação e armazenamento de informações. Elas são amplamente utilizadas para otimizar operações de busca em conjuntos de dados, reduzindo o tempo necessário para localizar informações específicas. Este trabalho se concentra na exploração e análise de tabelas hash em Java, com o objetivo de investigar sua implementação e desempenho.

A importância das tabelas hash está na capacidade de acesso rápido aos dados, minimizando o esforço de busca. Neste contexto, nosso objetivo é pesquisar e avaliar a implementação de tabelas hash baseadas em listas encadeadas e analisar o desempenho dessas estruturas.

Além disso, exploramos a influência do tamanho do vetor da tabela hash nas operações de inserção e busca, permitindo-nos avaliar o impacto dessa variável no desempenho da tabela hash, A implementação do algoritmo utilizado está disponível no Git Hub.

2 Conceitos do Algoritmo

Nossa implementação das tabelas hash se baseia nas classes ‘Node‘, ‘Lista Encadeada‘, ‘Registro‘ e ‘TabelaHash‘ em Java. Isso nos permite explorar as estruturas de dados e compreender o funcionamento interno de uma tabela hash.

Dentro da classe ‘TabelaHash‘, implementamos diferentes métodos de cálculo de índice, cada um associado a uma variação da função de hash. Vamos explorar esses métodos em subseções subsequentes e discutir seu propósito.

2.1 Métodos de Cálculo de Índice

Dentro da classe ‘TabelaHash‘, temos três métodos de cálculo de índice: ‘calcularIndiceDobramento‘, ‘calcularIndiceMultiplicacao‘ e ‘calcularIndiceResto‘. Cada um desses métodos é responsável por calcular o índice no vetor da tabela hash com base na função de hash correspondente.

- ‘calcularIndiceDobramento‘: Este método utiliza a técnica de dobramento para calcular o índice. Exploraremos como essa técnica funciona e como ela se relaciona com a função de hash.

- ‘calcularIndiceMultiplicacao‘: O método ‘calcularIndiceMultiplicacao‘ implementa a técnica de multiplicação para determinar o índice. Investigaremos os detalhes dessa abordagem e seu impacto no desempenho.

- ‘calcularIndiceResto‘: O terceiro método, ‘calcularIndiceResto‘, emprega a operação de resto da divisão para calcular o índice. Discutiremos essa estratégia e suas características distintivas.

A compreensão desses métodos é fundamental para a análise do desempenho da tabela hash e a escolha da função de hash adequada.

3 Metodologia e Experimentos

Nossa metodologia de experimentação envolve a realização de testes rigorosos para avaliar o desempenho das tabelas hash e seus métodos de cálculo de índice. No decorrer dos experimentos, empregamos uma série de etapas bem definidas para garantir resultados confiáveis.

3.1 Geração de Dados de Teste

Para conduzir nossos experimentos, geramos aleatoriamente cinco conjuntos de dados, cada um contendo 20.000, 100.000, 500.000, 1.000.000 e 5.000.000 de elementos. Cada elemento é representado por um objeto da classe ‘Registro‘, que contém um código de registro de nove dígitos. Esses conjuntos de dados variados permitem que avaliemos o desempenho das tabelas hash em diferentes cenários, refletindo situações reais de uso.

3.2 Configuração dos Tamanhos de Vetor

Além disso, adotamos diferentes tamanhos de vetor para cada teste, contemplando valores de 10, 100, 1.000, 10.000 e 100.000. Essa variação nos tamanhos de vetor permitirá que exploremos o impacto desse parâmetro no desempenho da tabela hash. A escolha cuidadosa dos tamanhos de vetor contribui para uma análise abrangente e precisa.

3.3 Coleta de Dados de Desempenho

Utilizamos a ferramenta VisualVM para monitorar o consumo de CPU e o tempo de execução durante a realização dos experimentos. Registramos o número de colisões ocorridas durante o processo de inserção e medimos o tempo de busca, bem como o número de comparações, em um conjunto de pelo menos cinco buscas em cada cenário.

Essa metodologia rigorosa nos permitiu coletar dados confiáveis e detalhados sobre o desempenho das tabelas hash e seus métodos de cálculo de índice. Na seção de Resultados e Discussão, apresentaremos as descobertas obtidas a partir desses experimentos.

4 Resutados

Resto do Índice

Table 1: Tabela de tamanho 10

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	392175	0
DADOS 100.000	Médio/Alto	13507593	3
DADOS 500.000	Médio/Alto	56288947	122
DADOS 1.000.000	Médio/Alto	23153462	465
DADOS 5.000.000	Alto	155874321	12442

Table 2: Tabela de tamanho 100

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	47058	0
DADOS 100.000	Médio/Alto	1346383	3
DADOS 500.000	Médio/Alto	6725976	122
DADOS 1.000.000	Médio/Alto	30377614	465
DADOS 5.000.000	Alto	116835720	12442

Table 3: Tabela de tamanho 1.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	19070	0
DADOS 100.000	Médio/Alto	161463	3
DADOS 500.000	Médio/Alto	805573	122
DADOS 1.000.000	Médio/Alto	3208104	465
DADOS 5.000.000	Alto	15932678	12442

Table 4: Tabela de tamanho 10.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	15766	0
DADOS 100.000	Médio/Alto	60640	3
DADOS 500.000	Médio/Alto	302978	122
DADOS 1.000.000	Médio/Alto	1375421	465
DADOS 5.000.000	Alto	9589078	12442

Table 5: Tabela de tamanho 100.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	14815	0
DADOS 100.000	Médio/Alto	48107	3
DADOS 500.000	Médio/Alto	241234	122
DADOS 1.000.000	Médio/Alto	1090805	465
DADOS 5.000.000	Alto	5985210	12442

Dobramento do Índice

Table 6: Tabela de tamanho 10

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	399417	0
DADOS 100.000	Médio/Alto	13846199	3
DADOS 500.000	Médio/Alto	69304976	122
DADOS 1.000.000	Médio/Alto	31572387	465
DADOS 5.000.000	Alto	131544145	12442

Table 7: Tabela de tamanho 100

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	52159	0
DADOS 100.000	Médio/Alto	1435777	3
DADOS 500.000	Médio/Alto	7167368	122
DADOS 1.000.000	Médio/Alto	32632726	465
DADOS 5.000.000	Alto	111589120	12442

Table 8: Tabela de tamanho 1.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	22505	0
DADOS 100.000	Médio/Alto	184595	3
DADOS 500.000	Médio/Alto	918475	122
DADOS 1.000.000	Médio/Alto	4152345	465
DADOS 5.000.000	Alto	18015536	12442

Table 9: Tabela de tamanho 10.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	18385	0
DADOS 100.000	Médio/Alto	72145	3
DADOS 500.000	Médio/Alto	360723	122
DADOS 1.000.000	Médio/Alto	1632398	465
DADOS 5.000.000	Alto	8793947	12442

Table 10: Tabela de tamanho 100.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	18096	0
DADOS 100.000	Médio/Alto	57469	3
DADOS 500.000	Médio/Alto	287345	122
DADOS 1.000.000	Médio/Alto	1293373	465
DADOS 5.000.000	Alto	5392210	12442

Multiplicação do Indice

Table 11: Tabela de tamanho 10

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	395866	0
DADOS 100.000	Médio/Alto	13631605	3
DADOS 500.000	Médio/Alto	70007884	122
DADOS 1.000.000	Médio/Alto	32256398	465
DADOS 5.000.000	Alto	134123452	12442

Table 12: Tabela de tamanho 100

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	47185	0
DADOS 100.000	Médio/Alto	1442531	3
DADOS 500.000	Médio/Alto	7156113	122
DADOS 1.000.000	Médio/Alto	32697019	465
DADOS 5.000.000	Alto	135784943	12442

Table 13: Tabela de tamanho 1.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Baixo	19735	0
DADOS 100.000	Médio/Alto	161589	3
DADOS 500.000	Médio/Alto	803802	122
DADOS 1.000.000	Médio/Alto	3279663	465
DADOS 5.000.000	Alto	13825447	12442

Table 14: Tabela de tamanho 10.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Médio/Alto	18366	0
DADOS 100.000	Médio/Alto	71054	3
DADOS 500.000	Médio/Alto	356896	122
DADOS 1.000.000	Médio/Alto	1606857	465
DADOS 5.000.000	Alto	6778669	12442

Table 15: Tabela de tamanho 100.000

	CPU	Tempo (μs)	Colisões
DADOS 20.000	Médio/Alto	17961	0
DADOS 100.000	Médio/Alto	56819	3
DADOS 500.000	Médio/Alto	284101	122
DADOS 1.000.000	Médio/Alto	1279613	465
DADOS 5.000.000	Alto	5327005	12442

5 Conclusão e Considerações Finais

Neste estudo, exploramos o desempenho de três funções de hash diferentes (Resto do Índice, Dobramento do Índice e Multiplicação do Índice) em uma variedade de tamanhos de tabela hash para cinco conjuntos de dados de diferentes tamanhos (20.000, 100.000, 500.000, 1.000.000 e 5.000.000 elementos). Nosso objetivo foi avaliar quais funções de hash são mais eficientes em termos de uso da CPU, tempo de execução e minimização de colisões em diferentes cenários.

Primeiramente, observamos que a função de hash "Resto do Índice" mostrou ser eficaz em termos de uso da CPU e tempo de execução, especialmente para tamanhos menores de tabela hash. No entanto, à medida que o tamanho da tabela aumentou, o número de colisões também aumentou, tornando-a menos eficiente para conjuntos de dados maiores.

A função de hash "Dobramento do Índice" também apresentou um desempenho aceitável, mas teve um aumento mais acentuado no número de colisões à medida que a tabela cresceu. Isso a torna adequada para tamanhos de tabela intermediários, mas menos eficiente para tamanhos muito grandes.

Por outro lado, a função de hash "Multiplicação do Índice" mostrou ser eficaz em manter o número de colisões relativamente baixo, mesmo para tamanhos maiores de tabela hash. No entanto, ela é mais intensiva em termos de CPU e tempo de execução. Portanto, para conjuntos de dados maiores e quando a minimização de colisões é crítica, a função de hash de multiplicação pode ser a escolha mais apropriada.

Em resumo, a escolha da função de hash depende do cenário específico e das prioridades do aplicativo. Se o tempo de execução e o uso da CPU forem uma preocupação maior do que as colisões, a função de hash "Resto do Índice" pode ser preferível em tamanhos menores de tabela hash. Para tamanhos intermediários, a função de hash "Dobramento do Índice" pode ser uma escolha razoável. No entanto, se a minimização de colisões for essencial, especialmente para conjuntos de dados maiores, a função de hash "Multiplicação do Índice" deve ser considerada.

É importante ressaltar que os resultados podem variar com base na implementação, nos dados específicos e nos requisitos do aplicativo. Recomenda-se a realização de testes práticos em cenários reais para escolher a função de hash mais apropriada.

Em trabalhos futuros, pode ser interessante explorar outras funções de hash, otimizações e estratégias de tratamento de colisões para aprimorar ainda mais o desempenho das tabelas hash em diferentes contextos.