RELATÓRIO TÉCNICO: IMPLEMENTAÇÃO E ANÁLISE DE TABELAS HASH

1. Introdução

Este relatório apresenta a análise comparativa de duas implementações de Tabelas Hash, utilizando duas funções de dispersão distintas: **Soma Simples (ASCII)** e **Polinomial (Base 31)**.

O objetivo do experimento, realizado com um conjunto de 5001 nomes e uma tabela de tamanho fixo (32 posições), é avaliar o impacto da função hash nas métricas de desempenho, como tempo de execução, número de colisões e distribuição das chaves.

Contexto do Experimento:

- **Tamanho da Tabela (M):** 32
- **Método de Tratamento de Colisão:** Encadeamento Separado (implícito nas métricas de "Maior cadeia" e contagem de elementos por posição).
- **Número de Elementos Inseridos (N):** 5000 (dos 5001 nomes lidos).
- **Fator de Carga (**alpha = N/M**):** 5000 / 32 = 156,25. Este alto fator de carga, muito superior a 1, garante que as tabelas estarão sob estresse, maximizando a ocorrência de colisões e a diferença de desempenho entre as funções hash.

2. Metodologia: Funções Hash em Detalhe

A função hash é o coração da Tabela Hash, responsável por transformar uma chave (string, neste caso) em um índice válido dentro do *array* da tabela. A qualidade de uma função é medida por sua capacidade de distribuir uniformemente as chaves, minimizando colisões.

Tabela 1: Soma Simples (ASCII)

Esta é a função hash mais elementar.

Aspecto	Descrição
Cálculo da Função Hash	Soma o valor ASCII de cada caractere da chave.
Fórmula	$h(chave) = \left(\sum_{i=0}^{L-1} \text{ASCII}(\text{chave}[i])\right) \pmod{M}$
Benefício	**Velocidade:** É extremamente rápida de ser calculada, pois envolve apenas adições.
Drawback	**Alta Colisão (Clusterização):** É altamente propensa a colisões, especialmente com chaves de mesmo comprimento ou com as mesmas letras em ordem diferente (anagramas). Muitos nomes

	de pessoas tendem a ter uma distribuição de valores ASCII semelhante, gerando um resultado de soma previsível.

Trecho de Código (funçãoHash):

```
@Override
protected int funcaoHash(String chave) {
    int soma = 0;
    for (int i = 0; i < chave.length(); i++) {
        soma += chave.charAt(i);
    }
    return Math.abs(soma % tamanho);
}
```

Tabela 2: Polinomial (Base 31)

Conhecida como **Hashing Polinomial** ou *Rolling Hash*, esta é uma técnica mais sofisticada.

Aspecto	Descrição
Cálculo da Função Hash	Trata a chave como um número na base 31 (um número primo). Cada caractere tem seu valor ASCII multiplicado pela base elevada à sua posição.
Fórmula	$h(chave) = \left(\sum_{i=0}^{L-1} \text{ASCII}(\text{chave}[i]) \cdot b^{L-1-i}\right) \pmod M$, onde $b = 31$
Benefício	**Baixa Colisão e Melhor Distribuição:** A base (geralmente um número primo como 31) assegura que a posição de cada caractere na string impacte o resultado de forma distinta, dispersando as chaves de maneira mais uniforme.
Drawback	**Velocidade:** É marginalmente mais lenta, pois exige operações de multiplicação e potências a cada iteração, tornando o cálculo mais custoso.

Trecho de Código (funcaoHash):

```
@Override
protected int funcaoHash(String chave) {
    long hashValor = 0;
    int base = 31; // Número primo como base

    for (int i = 0; i < chave.length(); i++) {
        // Cálculo polinomial (rolling hash)
        hashValor = (hashValor * base + chave.charAt(i));
    }

    return (int) (Math.abs(hashValor) % tamanho);
}
```

3. Resultados e Análise Comparativa

O alto fator de carga (156) resultou em muitas colisões em ambas as implementações, o que era esperado. A tabela a seguir resume as métricas de desempenho:

3.1. Comparação de Métricas de Desempenho

Métrica	Tabela 1: Soma Simples (ASCII)	Tabela 2: Polinomial (Base 31)	Observações
Elementos Inseridos	5000	5000	-
Colisões Totais	**4969**	**4969**	Empate. Colisões inevitáveis devido ao alto fator de carga.
Tempo de Inserção	**10,964 ms**	13,287 ms	A Tabela 1 é mais rápida devido à simplicidade de sua função hash.
Tempo de Busca	**1,712 ms**	1,750 ms	A Tabela 1 é marginalmente mais rápida na busca.
Maior Cadeia	182	**178**	**Vantagem Tabela 2.** Indica que a distribuição da Polinomial é melhor, pois a "pior" cadeia é menor.

3.2. Distribuição das Chaves e Clusterização

Métrica	Tabela 1 (Soma Simples)	Tabela 2 (Polinomial)
Maior Cadeia (Posição 14)	182 elementos	154 elementos
Menor Cadeia (Posição 16)	139 elementos	140 elementos
Desvio Padrão (aproximado)	Maior	Menor

A análise da distribuição por posição é crucial para entender a **clusterização** (agrupamento indesejado de chaves). Uma distribuição ideal deve ter valores próximos à média (5000 / 32 = 156,25).

- **Tabela 1 (Soma Simples):** Apresenta uma variação maior, com um pico de 182 elementos na Posição 14, indicando um alto grau de clusterização nessa posição específica.
- **Tabela 2 (Polinomial):** O valor máximo é 178 (Posição 12), sendo que o maior desvio em relação à média (156,25) é menor do que na Tabela 1. O número menor na "Maior cadeia" (178 vs 182) confirma que a função Polinomial **distribuiu melhor as chaves**.

3.3. Análise de Distribuição e Colisões

Tabela 1: Distribuição das Chaves por Posição (Quantidade)

Esta tabela mostra a quantidade de chaves alocadas em cada posição (0 a 31) para as duas estruturas de dados (Tabela 1 e Tabela 2).

Posição	Tabela 1 (Qtd)	Tabela 2 (Qtd)
0	160	161
1	171	166
2	161	164
3	146	156
4	150	159
5	149	166
6	167	154
7	159	144
8	164	171
9	146	144
10	155	163
11	149	140
12	176	178
13	150	146
14	182	154
15	148	146
16	139	140
17	146	140
18	162	163
19	160	140
20	148	144
21	152	160
22	158	176
23	159	148
24	147	151
25	148	169
26	157	149
27	147	165
28	160	159
29	159	160
30	177	177
31	148	147

Tabela 2: Colisões por Posição (Clusterização)

Esta tabela mostra o número de colisões (indicando o grau de clusterização) em cada posição para as duas estruturas de dados.

Posição	Tabela 1 (Colisões)	Tabela 2 (Colisões)
0	159	160
1	170	165
2	160	163
3	145	155
4	149	158
5	148	165
6	166	153
7	158	143
8	163	170
9	145	143
10	154	162
11	148	139
12	175	177
13	149	145
14	181	153
15	147	145
16	138	139
17	145	139
18	161	162
19	159	140
20	147	143
21	151	159
22	157	175
23	158	147
24	146	150
25	148	168
26	156	148
27	146	164
28	159	158
29	158	159
30	176	176
31	147	146

4. Conclusão

Os resultados confirmam as características teóricas das funções hash:

1. **Velocidade de Cálculo vs. Desempenho na Colisão:**
    - A **Tabela 1 (Soma Simples)** foi a mais rápida nos testes de inserção e busca devido à simplicidade da função.
    - A **Tabela 2 (Polinomial)**, apesar de ter um tempo de cálculo da função hash maior, demonstrou uma **melhor qualidade de distribuição de chaves**, evidenciada pela sua **menor cadeia máxima (178 vs 182)**.
2. **Impacto do Alto Fator de Carga:**
    - O altíssimo fator de carga fez com que ambas as tabelas tivessem um número idêntico de colisões (4969).
    - No entanto, a métrica de **Maior cadeia** é a mais relevante neste cenário de alta colisão, pois ela dita o tempo de busca no "pior caso". Como a Tabela 2 gerou uma cadeia máxima menor, ela é tecnicamente **mais robusta** e garantiria um desempenho mais previsível e estável em situações extremas.

Recomendação:

Para aplicações onde a **qualidade da distribuição** e a garantia de um **pior caso de busca menos custoso** são prioritárias (situações com risco de alto fator de carga ou chaves problemáticas), a **Função Hash Polinomial (Base 31)** é a mais indicada.

No entanto, para aplicações onde a tabela não atinge um fator de carga tão elevado e a **velocidade pura** da função hash é crítica, a **Soma Simples (ASCII)** pode ser a melhor escolha.