Pontifícia Universidade Católica do Paraná (PUCPR)
Bacharelado em Sistemas de Informação

Disciplina: Resolução de Problemas Estruturados em Computação
Alunas: Júlia Tatim, Laura, Maria Eduarda Melo
Professora: Marina de Lara
Local e Data: Curitiba, 27 de Maio de 2025

Implementação de Tabelas Hash com Diferentes Funções Hash e Análise Comparativa de Eficiência

1. Introdução

O presente trabalho tem como objetivo central a implementação e avaliação de duas tabelas hash distintas, utilizando a linguagem Java e conceitos de Programação Orientada a Objetos. O estudo visa comparar a eficiência de duas funções hash diferentes – FNV-1a e uma função Polinomial para strings – em termos de três métricas principais: o número total de colisões durante a inserção de dados, o tempo despendido nas operações de inserção e busca, e a uniformidade da distribuição das chaves nas tabelas.

Tabelas hash são estruturas de dados fundamentais em ciência da computação, amplamente utilizadas devido à sua capacidade de fornecer acesso rápido a dados. A escolha de uma boa função hash e de um método eficaz para tratamento de colisões é crucial para o desempenho dessas estruturas. Este relatório detalha a metodologia empregada na construção das tabelas, apresenta os resultados experimentais obtidos a partir da inserção de 5001 nomes e, por fim, discute as observações e conclusões retiradas da análise comparativa.

---

2. Metodologia

O desenvolvimento do projeto seguiu as diretrizes propostas, utilizando a linguagem Java e aplicando princípios de Programação Orientada a Objetos.

- **Ambiente e Dados:**
  - O programa foi desenvolvido para ler um conjunto de 5001 nomes do arquivo `female_names.txt`.
  - Ambas as tabelas hash foram inicializadas com uma capacidade fixa de 32 posições.
- **Funções Hash Implementadas:**

    - **Tabela Hash 1 (FNV-1a):** Utilizou-se a função de hash FNV-1a (Fowler-Noll-Vo), uma função não criptográfica conhecida por sua boa dispersão e velocidade. O cálculo envolve operações XOR e multiplicação por um número primo específico (FNV_PRIME) sobre os bytes da chave de entrada.
    - **Tabela Hash 2 (Polinomial):** Implementou-se uma função de hash polinomial para strings. Nesta abordagem, cada caractere da string é multiplicado por uma potência de um número primo (neste caso, 31), e os resultados são somados.
- **Tratamento de Colisões:**
    - Para ambas as tabelas, o método de tratamento de colisões adotado foi o **Encadeamento Separado**. Cada posição na tabela hash armazena uma referência para uma lista encadeada (`ListaEncadeada`), onde os elementos que colidem (ou seja, que são mapeados para o mesmo índice pela função hash) são armazenados.
- **Testes de Eficiência:**
    - **Número de Colisões:** Contabilizado sempre que uma inserção ocorria em uma posição da tabela que já continha um ou mais elementos (ou seja, a lista encadeada naquele índice não estava vazia).
    - **Tempo de Inserção e Busca:** Medido utilizando `System.nanoTime()` antes e depois do loop de inserção de todos os 5001 nomes, e similarmente para o loop de busca de todos esses nomes. Os resultados foram convertidos para milissegundos (ms).
    - **Distribuição das Chaves:** Verificada pela contagem do número de chaves armazenadas em cada uma das 32 posições da tabela ao final das inserções.

Nota: A saída do console indicou que a Tabela 2 teve "Total de colisões na Hash Table 1: 4968". Assume-se que isso seja um erro de digitação no rótulo da saída do programa e que o valor se refere à Tabela 2.

---

3. Resultados Obtidos

Após a execução do programa Java com a entrada de 5001 nomes, foram coletados os seguintes dados:

- **Dados Gerais:**
    - Número total de nomes lidos do arquivo: 5001
- **Tabela Hash 1 (FNV-1a):**
    - Tempo total de inserção: **4.9809 ms**
    - Tempo total de busca: **1.3357 ms**

    - Tempo total de execução (Inserção + Busca): **6.3166 ms**
    - Número total de colisões: **4968**
    - **Distribuição das Chaves (Número de chaves por posição):**
        - Posição 00: 168 | Posição 01: 154 | Posição 02: 148 | Posição 03: 133
        - Posição 04: 157 | Posição 05: 159 | Posição 06: 156 | Posição 07: 158
        - Posição 08: 156 | Posição 09: 167 | Posição 10: 136 | Posição 11: 158
        - Posição 12: 144 | Posição 13: 143 | Posição 14: 140 | Posição 15: 152
        - Posição 16: 165 | Posição 17: 183 | Posição 18: 131 | Posição 19: 149
        - Posição 20: 151 | Posição 21: 177 | Posição 22: 158 | Posição 23: 168
        - Posição 24: 165 | Posição 25: 151 | Posição 26: 149 | Posição 27: 156
        - Posição 28: 164 | Posição 29: 177 | Posição 30: 133 | Posição 31: 162
- **Tabela Hash 2 (Polinomial para Strings):**
    - Tempo total de inserção: **4.9704 ms**
    - Tempo total de busca: **0.825 ms**
    - Tempo total de execução (Inserção + Busca): **5.7954 ms**
    - Número total de colisões: **4968** (conforme discutido na Metodologia)
    - **Distribuição das Chaves (Número de chaves por posição):**
        - Posição 00: 160 | Posição 01: 164 | Posição 02: 163 | Posição 03: 155
        - Posição 04: 158 | Posição 05: 165 | Posição 06: 153 | Posição 07: 143
        - Posição 08: 170 | Posição 09: 143 | Posição 10: 162 | Posição 11: 140
        - Posição 12: 177 | Posição 13: 145 | Posição 14: 153 | Posição 15: 146
        - Posição 16: 139 | Posição 17: 138 | Posição 18: 162 | Posição 19: 139
        - Posição 20: 143 | Posição 21: 158 | Posição 22: 175 | Posição 23: 147
        - Posição 24: 150 | Posição 25: 168 | Posição 26: 148 | Posição 27: 164
        - Posição 28: 158 | Posição 29: 159 | Posição 30: 176 | Posição 31: 147

---

4. Análise dos Resultados

A análise comparativa dos resultados obtidos para as duas funções hash revela os seguintes pontos:

- **Número de Colisões:**
  - Ambas as funções hash, FNV-1a e Polinomial, resultaram em um número idêntico de colisões: **4968**. Com 5001 itens inseridos em 32 posições, o fator de carga é extremamente alto (aproximadamente 156.28 itens por posição, em média). Dado que uma colisão é contada quando um item é inserido em um slot já ocupado, e há 32 slots, o número máximo de colisões esperadas seria 5001 - 32 = 4969 (assumindo que cada slot é preenchido pelo menos uma vez). O valor de 4968 colisões é consistente com esse cenário de alta ocupação, indicando que ambas as funções distribuíram os itens de forma que quase todas as inserções, após as primeiras 32 (ou um número próximo), resultaram em colisões.
- **Tempo de Inserção:**
  - A Tabela Hash 2 (Polinomial) apresentou um tempo de inserção marginalmente menor (**4.9704 ms**) em comparação com a Tabela Hash 1 (FNV-1a, **4.9809 ms**). A diferença é mínima (aproximadamente 0.0105 ms), sugerindo que, para este volume de dados e complexidade das strings, ambas as funções têm um custo computacional muito similar durante a fase de cálculo do hash e inserção na lista encadeada.
- **Tempo de Busca:**
  - Aqui se observa uma diferença mais notável. A Tabela Hash 2 (Polinomial) foi consideravelmente mais rápida no tempo total de busca (**0.825 ms**) do que a Tabela Hash 1 (FNV-1a, **1.3357 ms**). A Tabela 2 levou aproximadamente 62% do tempo da Tabela 1 para realizar todas as buscas. Isso sugere que, embora o número total de colisões seja o mesmo, a forma como os elementos estão distribuídos dentro das listas encadeadas ou a eficiência da busca nessas listas pode ter sido discretamente favorecida pela distribuição da função Polinomial.
- **Tempo Total de Execução:**
  - Consequentemente, a Tabela Hash 2 (Polinomial) teve um tempo total de execução menor (**5.7954 ms**) em comparação com a Tabela Hash 1 (FNV-1a, **6.3166 ms**).

- **Distribuição das Chaves (Clusterização):**
    - **Tabela 1 (FNV-1a):** Mínimo de 131 chaves/posição, Máximo de 183 chaves/posição. Desvio Padrão: aprox. 13.97.
    - **Tabela 2 (Polinomial):** Mínimo de 138 chaves/posição, Máximo de 177 chaves/posição. Desvio Padrão: aprox. 11.77.
    - Ambas as tabelas mostram que todas as 32 posições foram intensamente utilizadas, o que é esperado pelo alto fator de carga. A Tabela 2 (Polinomial) apresentou uma distribuição ligeiramente mais uniforme, como indicado por um menor desvio padrão e uma menor amplitude entre o número mínimo e máximo de chaves por posição (Range: 39 para Tabela 2 vs 52 para Tabela 1). Uma distribuição mais uniforme pode levar a listas encadeadas um pouco menores em média (ou com comprimentos menos variáveis), o que pode explicar parcialmente o melhor tempo de busca da Tabela 2, mesmo com o mesmo número total de colisões.

---

5. Conclusão

Este trabalho demonstrou a implementação e comparação de duas tabelas hash com diferentes funções (FNV-1a e Polinomial para strings) sob um cenário de alto fator de carga. Ambas as funções resultaram no mesmo número total de colisões (4968), indicando uma saturação similar das tabelas.

No entanto, a **função hash Polinomial (Tabela 2) apresentou um desempenho superior em termos de tempo de busca e, consequentemente, tempo total de execução**. Além disso, exibiu uma **distribuição de chaves ligeiramente mais uniforme** entre as posições da tabela, evidenciada por um menor desvio padrão e menor variação entre o mínimo e máximo de itens por slot. Embora a diferença no tempo de inserção tenha sido insignificante, a maior eficiência na busca torna a função Polinomial, neste cenário específico, marginalmente mais vantajosa.

O estudo reforça a importância da escolha da função hash, mesmo quando o número de colisões totais é similar, pois a qualidade da distribuição pode impactar o desempenho das operações subsequentes, como a busca. A atividade proporcionou uma valiosa experiência prática na implementação de estruturas de dados essenciais e na análise crítica de seu desempenho.

---

6. Referências (Opcional)

- (Se você usou fontes externas, cite-as aqui. Ex: Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Java*. Wiley.)
- Documentação do Projeto TDE fornecida pela disciplina.
