AED2 - Algoritmos e Estr. de Dados II

Aula 4: Tabelas Hash e Funções de Dispersão

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

Sumário

1. O Problema da Busca

Imagine que você quer verificar se um CPF está em uma base de dados.

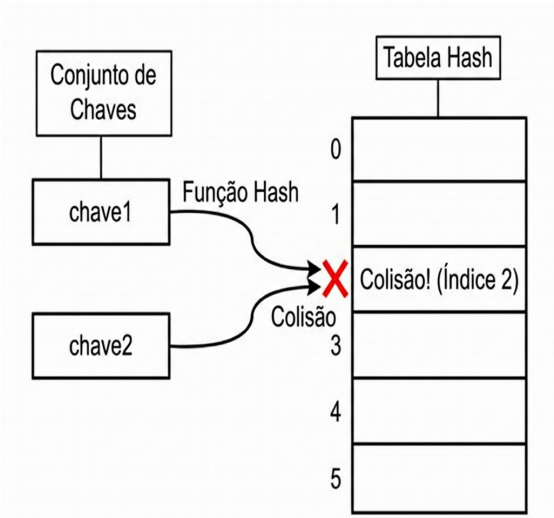
- Em um vetor desordenado: $O(N)$.
- Em um vetor ordenado (Busca Binária): $O(\log N)$.
- **Em uma Tabela Hash:** $O(1)$ (no caso médio).

As Tabelas Hash (ou de Dispersão) buscam o "Santo Graal" das estruturas de dados: operações de busca, inserção e remoção em tempo constante.

2. Conceito Fundamental

A ideia é usar uma **Função de Hash** ($h(x)$) que mapeia chave $\rightarrow$ índice $[0, M-1]$.

$$h(\text{chave}) \rightarrow \text{índice}$$



3. Tratamento de Colisões

Pelo **Princípio da Casa dos Pombos**, se temos mais chaves possíveis do que índices na tabela, colisões são inevitáveis. Ou seja, $h(k1) == h(k2)$ para $k1 \neq k2$.

3.1. Encadeamento Exterior (Separate Chaining)

Cada posição da tabela aponta para uma lista encadeada (ou vetor, ou árvore) contendo todos os elementos que colidiram ali.

- **Vantagem:** Simples de implementar; a tabela "nunca enche" (apenas as listas ficam longas).
- **Desvantagem:** Uso extra de memória (ponteiros); perde localidade de cache.
- **Java:** O `HashMap` do Java usa essa técnica. Se a lista ficar muito grande (limit ¿ 8), ele converte a lista em uma Árvore Rubro-Negra para garantir busca $O(\log N)$ no pior caso.

3.2. Endereçamento Aberto (Open Addressing)

Se ocorrer colisão, procuramos outro lugar *livre* dentro da própria tabela seguindo uma regra de sondagem.

4. Complexidade e Fator de Carga

O desempenho depende do **Fator de Carga** (α):

$$\alpha = \frac{N}{M}$$

Onde N é o número de elementos inseridos e M é o tamanho da tabela.

Operação	**Caso Médio**	**Pior Caso**
Busca / Inserção	$O(1)$	$O(N)$

Para garantir $O(1)$, as implementações dobram o tamanho da tabela (M) e remapeiam tudo (**Rehash**) quando α passa de um limiar (no Java, 0.75).

5. Prática em Java (‘HashSet‘ vs ‘HashMap‘)

Interfaces Importantes

- `Set` **(Conjunto):** Guarda apenas chaves únicas. Não permite duplicatas.
- `Map` **(Dicionário/Mapa):** Guarda pares **Chave** $\rightarrow$ **Valor**.

‘HashMap‘ / ‘HashSet‘

- **Ordem:** Não garante nenhuma ordem dos elementos.
- **Complexidade:** $O(1)$ para `put`, `get`, `contains`.
- **Requisito:** As chaves devem implementar `equals()` e `hashCode()` corretamente.

‘TreeMap‘ / ‘TreeSet‘

- **Ordem:** Mantém os elementos ordenados (naturalmente ou via `Comparator`).
- **Implementação:** Árvore Rubro-Negra (Red-Black Tree).
- **Complexidade:** $O(\log N)$.
- **Uso:** Quando você precisa processar dados em ordem ou buscar faixas de valores.

6. Exercício Resolvido: Two Sum

Problema: Dado um array de inteiros e um alvo `target`, retorne os índices de dois números que somam `target`. **Solução Ingênua** $O(N^2)$**:** Dois loops for. **Solução com Hash** $O(N)$**:**

```java
public int[] twoSum(int[] nums, int target) {
    // Mapa: Valor -> Indice
    Map<Integer, Integer> map = new HashMap<>();

    for (int i = 0; i < nums.length; i++) {
        int complemento = target - nums[i];

        if (map.containsKey(complemento)) {
            return new int[] { map.get(complemento), i };
        }

        map.put(nums[i], i);
    }
    return new int[]{}; // Não encontrou
}
```

7. Exercícios Sugeridos

- **LeetCode 1:** Two Sum.
- **LeetCode 49:** Group Anagrams (Use Hash de contagem ou String ordenada como chave).
- **Beecrowd 1256:** Tabelas Hash (Implementar o encadeamento exterior na mão).