

Nome: Juan Rodrigues dos Santos Servelo e João Victor Monteiro Tancon

## RELATÓRIO TABELA HASH

### Análise dos Resultados

O objetivo deste experimento foi comparar o impacto de duas **funções hash distintas** no desempenho de uma **tabela hash com encadeamento exterior** como método de tratamento de colisões. A estrutura de dados utilizada foi mantida constante, alterando apenas o método de geração do índice.

Dados de saída do console:

Tabela com Hash (soma de chars):	Tabela com Hash (hashCode):
Colisões: 15214	Colisões: 5292
Tempo de inserção: 9.2234 ms	Tempo de inserção: 6.1596 ms
Tempo de busca: 1.6962 ms	Tempo de busca: 0.5578 ms
Distribuição de chaves:	Distribuição de chaves:
Índice 0: 0 entradas	Índice 0: 1 entradas
Índice 1: 0 entradas	Índice 1: 0 entradas
Índice 2: 0 entradas	Índice 2: 0 entradas
Índice 3: 0 entradas	Índice 3: 0 entradas
Índice 4: 0 entradas	Índice 4: 1 entradas
Índice 5: 0 entradas	Índice 5: 0 entradas
Índice 6: 0 entradas	Índice 6: 2 entradas
Índice 7: 0 entradas	Índice 7: 1 entradas
Índice 8: 0 entradas	Índice 8: 0 entradas
Índice 9: 0 entradas	Índice 9: 0 entradas
Índice 10: 0 entradas	Índice 10: 2 entradas
Índice 11: 0 entradas	Índice 11: 0 entradas
Índice 12: 0 entradas	Índice 12: 0 entradas
Índice 13: 0 entradas	Índice 13: 1 entradas
Índice 14: 0 entradas	Índice 14: 0 entradas
Índice 15: 0 entradas	Índice 15: 0 entradas
Índice 16: 0 entradas	Índice 16: 1 entradas
Índice 17: 0 entradas	Índice 17: 0 entradas
Índice 18: 0 entradas	Índice 18: 0 entradas

A comparação foi feita com base nos seguintes critérios:


## Número de colisões para cada função hash

- **Hash pela soma dos caracteres:**

Essa abordagem resultou em **15.214 colisões** durante a inserção. O valor elevado se explica pela simplicidade da função — strings com os mesmos caracteres em ordens diferentes ou com somas similares acabam colidindo com facilidade, gerando **baixa dispersão** dos índices na tabela.

- **Hash com hashCode():**

A função hashCode() padrão da linguagem produziu **apenas 5.292 colisões**. Isso representa uma melhoria significativa na distribuição dos dados. A implementação da função hash garante uma dispersão mais uniforme, minimizando as colisões entre diferentes strings.

 **Conclusão:** Apesar de ambas as tabelas usarem o mesmo método de colisão (encadeamento exterior), a **qualidade da função hash é determinante** para a eficiência da estrutura. O hashCode() mostra-se muito mais robusto e eficaz na prevenção de colisões.

## ✳ Clusterização — Distribuição por posição

### Hash pela soma dos caracteres:

A tabela apresentou **alta concentração de entradas em determinadas faixas** de índices, enquanto outras permanecem totalmente vazias. Exemplo:

Índice 715: 18 entradas

Índice 716: 26 entradas

Índice 717: 16 entradas

Índice 718: 28 entradas

Índice 719: 22 entradas

🔍 Observa-se um padrão de clusterização local: as colisões se concentram em regiões específicas, indicando má dispersão da função hash.

### Hash com hashCode():

Apresenta uma **distribuição muito mais uniforme**. Entradas espalhadas desde o índice 0 até os índices finais da tabela:

Índice 0: 1 entrada

Índice 6: 2 entradas

Índice 10: 2 entradas

...

Índice 9318: 3 entradas

Índice 9320: 2 entradas



Poucas posições têm mais de 2 ou 3 entradas, e uma grande quantidade de índices contém 0 ou 1 entrada — sinal de **ótima dispersão**.

### 🧠 Conclusão:

A clusterização é um reflexo direto da função hash. A função hashCode() apresentou **menor aglomeração de chaves**, enquanto a função por soma de caracteres gerou **áreas de concentração de colisões**, o que pode prejudicar a performance em buscas.



## Tempo de inserção e busca

### Hash pela soma dos caracteres:


-  **Inserção: 9,22 ms**
-  **Busca: 1,69 ms**

O tempo de inserção é mais alto devido ao grande número de colisões e consequente necessidade de gerenciamento de listas encadeadas extensas. A busca também é mais custosa, pois muitas vezes é necessário percorrer várias entradas dentro da mesma posição.

### Hash com hashCode():

-  **Inserção: 6,16 ms**
-  **Busca: 0,56 ms**

A performance é superior em ambos os casos. A menor quantidade de colisões contribui para inserções mais diretas e listas encadeadas menores, facilitando buscas rápidas.

 Conclusão: Mesmo com o mesmo tipo de colisão (encadeamento exterior), **a eficiência da função hash afeta diretamente os tempos** de execução. Uma função melhor distribuída reduz a complexidade das operações.

## Considerações Finais

- A **função de hash baseada na soma dos caracteres** demonstrou ser **ineficiente em termos de dispersão**: além de provocar um número elevado de colisões (15.214), concentrou dados em clusters, prejudicando a velocidade das operações.
- Por outro lado, a **função hash hashCode()** apresentou **ótimos resultados**, com **baixa taxa de colisões (5.292)**, **distribuição equilibrada das chaves** e **menores tempos de execução**.

## Conclusão geral:

Mesmo quando se mantém o tratamento de colisões por **encadeamento exterior**, a **qualidade da função hash** utilizada tem impacto direto e significativo no desempenho da tabela. Uma função mal planejada pode gerar clusterizações intensas e colisões em massa, tornando a estrutura ineficiente.