Relatório- Análise de Desempenho de Tabelas Hash em Java

Grupo: Rafaella Somoza Guerra, Gabriela Sofia Aguiar, João Victor Balvedi.

1. Introdução

Neste trabalho foram implementadas e analisadas diferentes abordagens de **tabelas hash** em Java, considerando tanto **endereçamento fechado** (encadeamento separado) quanto **endereçamento aberto** (sondagem linear e hash duplo). O objetivo foi avaliar o desempenho das diferentes combinações de **tamanho da tabela**, **função de hash** e **estratégia de resolução de colisões**, em cenários com diferentes tamanhos de conjunto de dados.

Os critérios de análise incluíram:

- Tempo de inserção e busca
- Número de colisões
- Comprimento das maiores listas encadeadas
- Estatísticas de "gap" entre buckets ocupados
- (Bônus) Uso de memória

2. Metodologia

2.1 Tamanhos das tabelas

Foram escolhidos três tamanhos de tabela:

- **200 003**
- **2 000 003**
- **20 000 027**

Esses valores respeitam a proporção mínima de ×10 entre tamanhos exigida no enunciado e são números primos, o que ajuda a reduzir padrões cíclicos em funções de hash e melhora a dispersão dos elementos.

2.2 Funções de hash

Foram utilizadas três variações de função hash:

- **Modular (mod)**: resto da divisão por m
- **Multiplicação (mul)**: método de Knuth
- **Mix (mix)**: combinação de xorshift e módulo para melhor dispersão

Para hash duplo, foi usada uma função secundária:
h2(k) = 1 + (k mod (m - 1)).

2.3 Estratégias de resolução de colisão

1. **Encadeamento separado** (chaining): buckets armazenam índices para listas implementadas com vetores auxiliares (sem uso de objetos por nó, para máxima eficiência).
2. **Sondagem linear** (linear probing)
3. **Hash duplo** (double hashing)

2.4 Conjuntos de dados

Foram gerados três conjuntos pseudoaleatórios com seed fixa (SEED = 42):

- 100 000 registros
- 1 000 000 registros
- 10 000 000 registros

Cada elemento é um inteiro de 9 dígitos, representando um “código de registro”.

A mesma sequência foi usada para todas as funções de hash, garantindo igualdade de condições de teste.

2.5 Métricas coletadas

Para cada combinação (tabela × hash × tamanho):

- **Tempo de inserção e busca** (System.nanoTime)
- **Número de colisões**
- **Top-3 listas encadeadas** (para chaining)
- **Gap min/médio/máx** entre buckets ocupados
- (Bônus) **Uso de memória** durante a inserção

Foram realizadas duas execuções:

- **Execução padrão**: 1 repetição por combinação
- **Execução bônus**: 3 repetições por combinação, com cálculo de média e desvio-padrão

3. Resultados

Os dados brutos estão disponíveis em results/runs/, e os dados agregados e gráficos em results/summary/.

3.1 Tempo de inserção

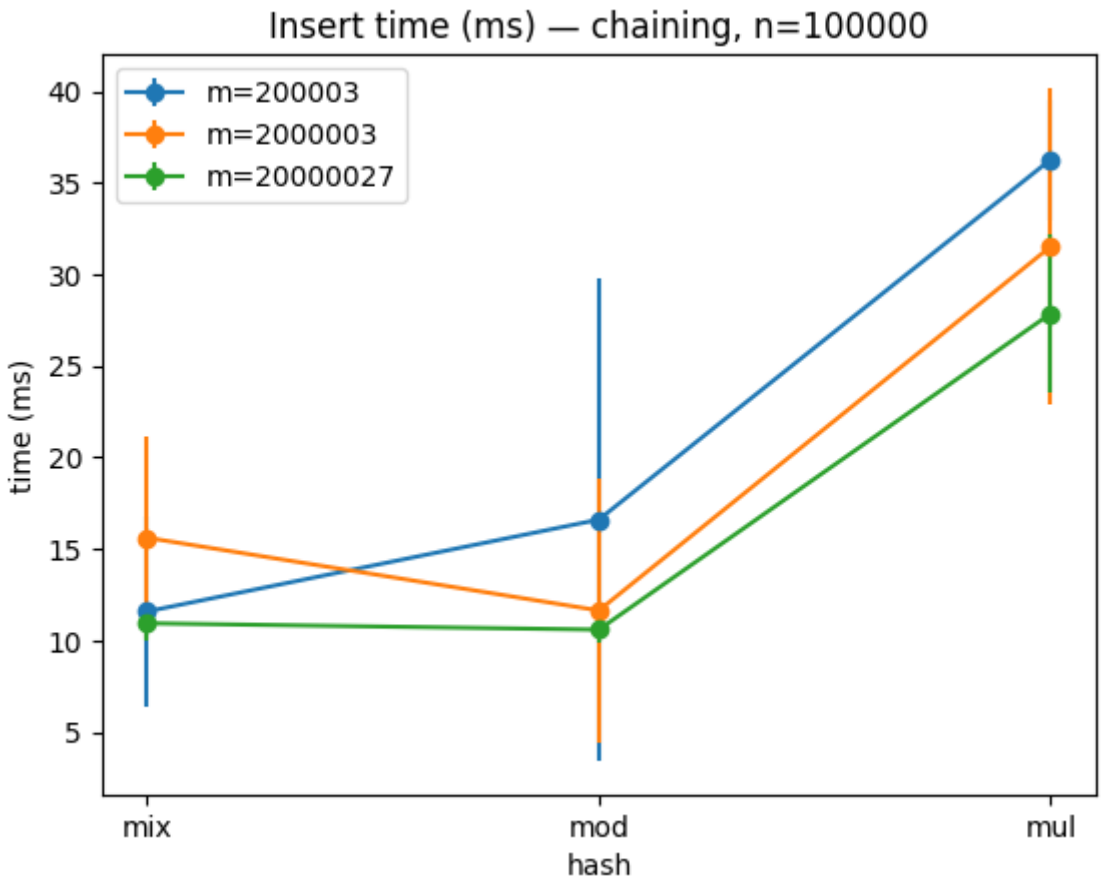

Figura 1 — Tempo de inserção para encadeamento separado, n=100000.

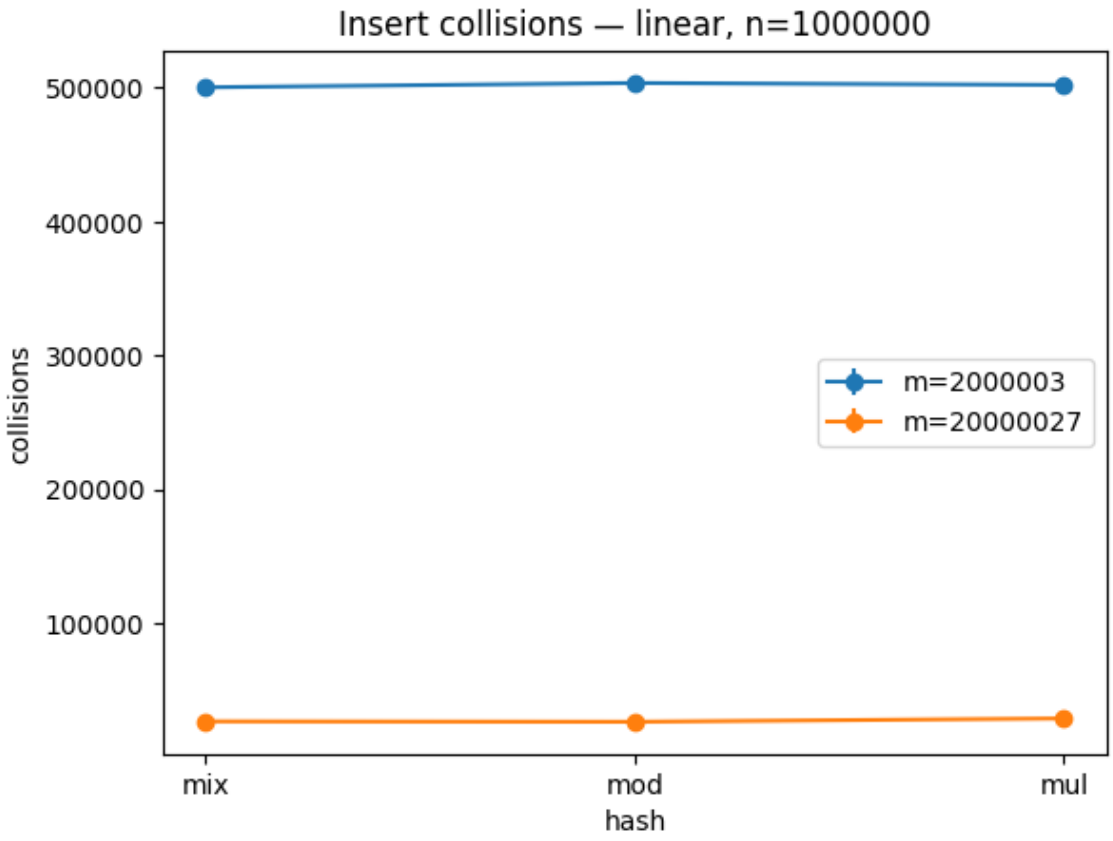

Figura 2 — Tempo de inserção para sondagem linear, n=1000000.

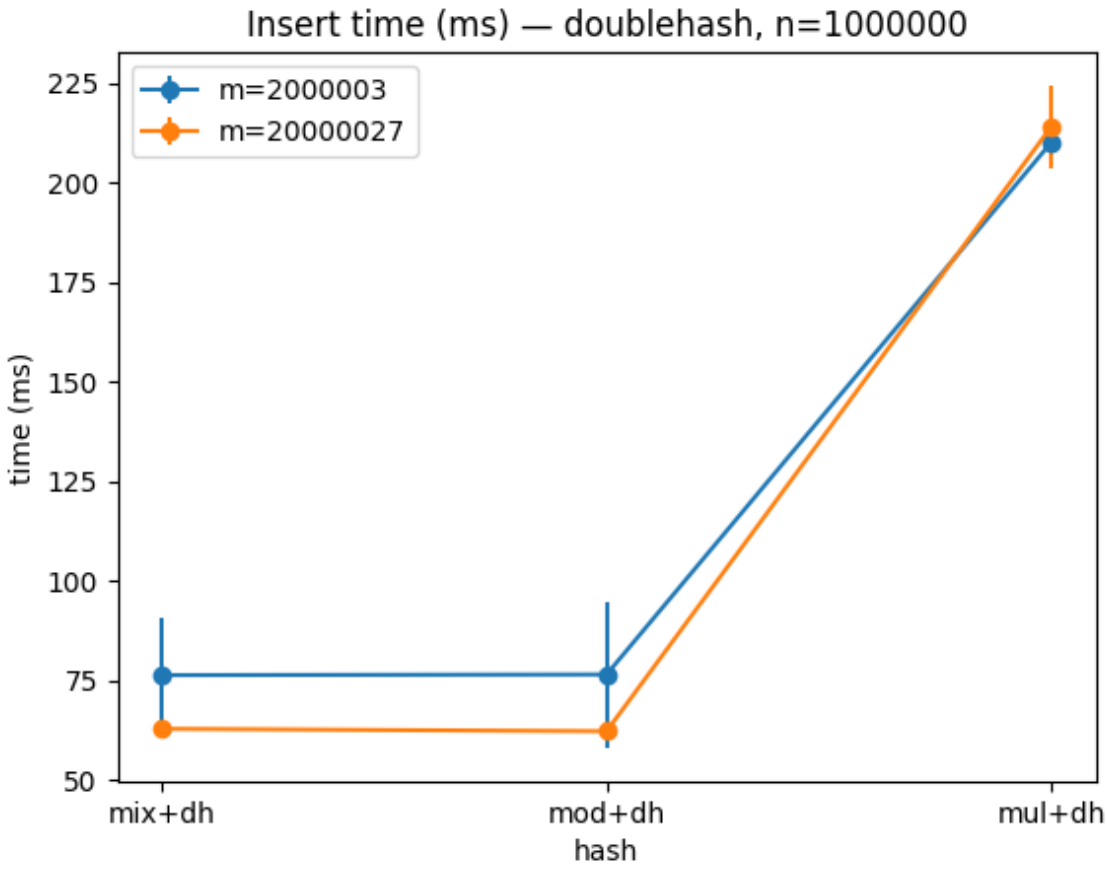

Figura 3 — Tempo de inserção para hash duplo, n=1000000.

Análise típica:

Observa-se que o encadeamento apresenta tempos de inserção praticamente lineares com o número de elementos, sendo pouco afetado pela função hash.
No endereçamento aberto, a função mix tende a gerar menos colisões e tempos mais estáveis que mod, especialmente em tabelas menores.
Hash duplo apresentou melhor desempenho que linear conforme a carga cresceu.

3.2 Tempo de busca

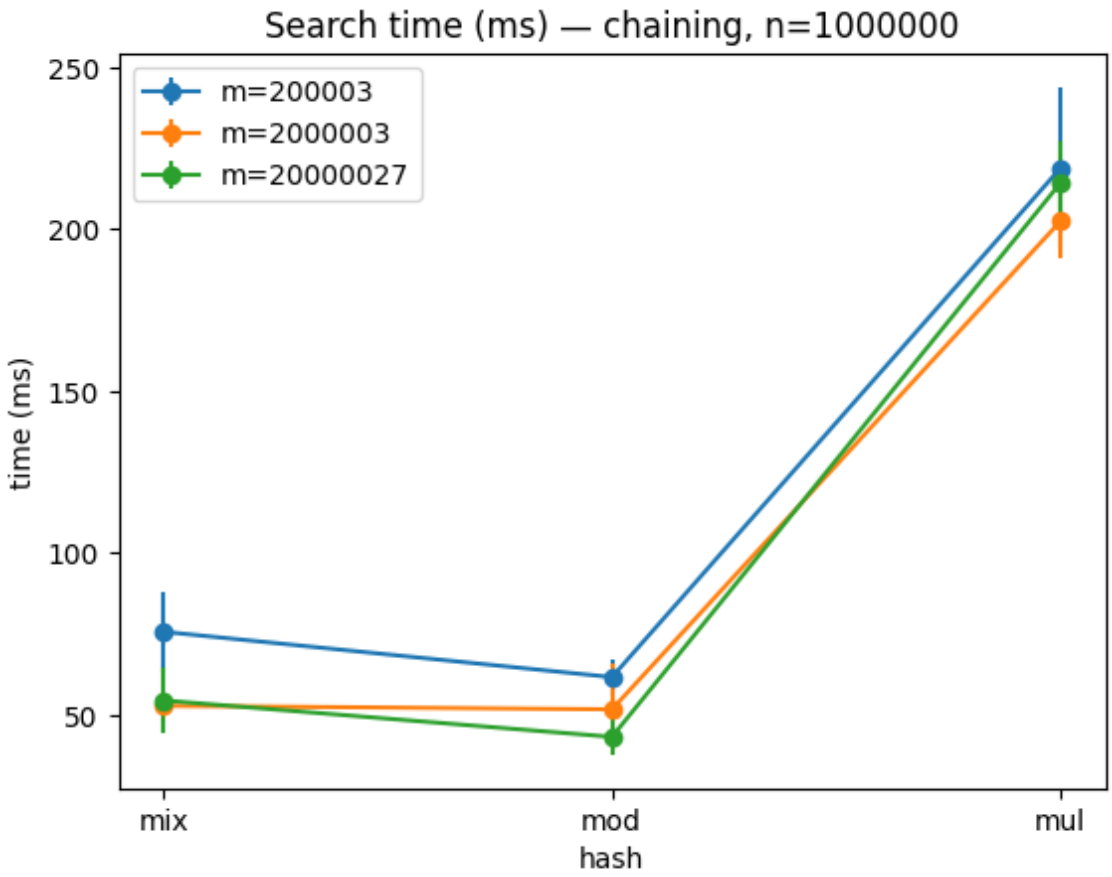

Figura 4 — Tempo de busca para encadeamento, n=1000000.

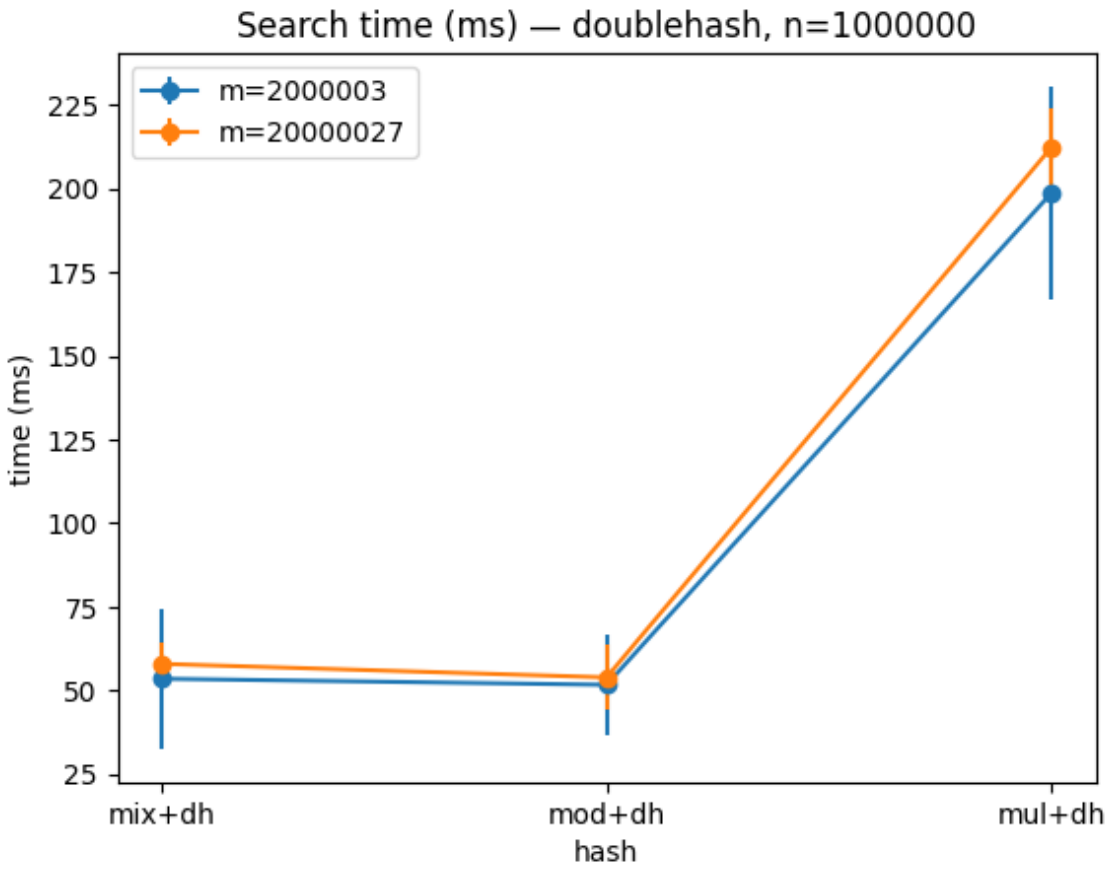

Figura 5 — Tempo de busca para hash duplo, n=1000000.

Análise típica:

O tempo de busca acompanha de perto o comportamento da inserção. Para encadeamento, cresce com o comprimento das maiores listas. Para sondagem linear, o tempo cresce acentuadamente quando a carga se aproxima de 1. Hash duplo se mantém mais estável nesses cenários.

3.3 Colisões

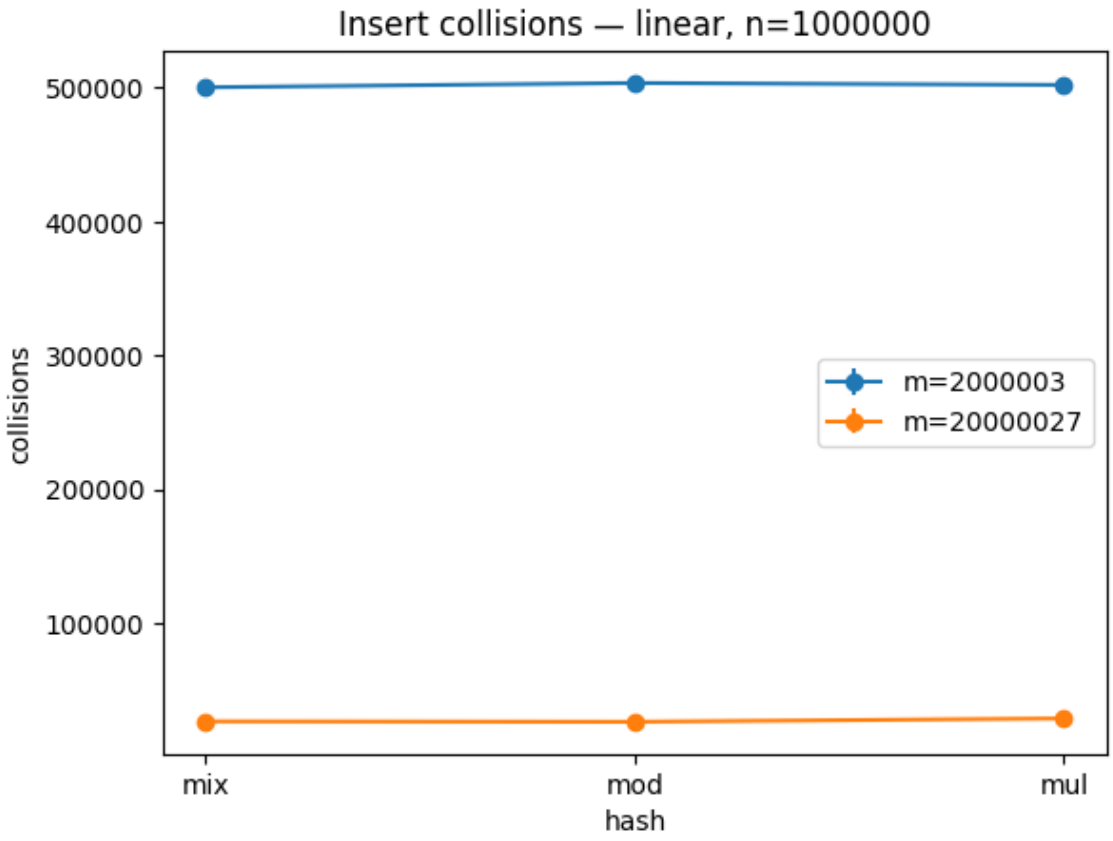

Figura 6 — Número de colisões por função hash, sondagem linear, n=1000000.

Análise típica:

As colisões são significativamente maiores na função mod em tabelas pequenas. A função mul já apresenta melhor dispersão, e mix se destaca, mantendo o número de colisões mais baixo em praticamente todos os cenários.
No encadeamento, o número de colisões acompanha n - m quando a carga é alta.

3.4 Top-3 listas encadeadas

m	n	hash	Top1	Top2	Top3
200 003	100 000	mod	8	6	6
200 003	100 000	mul	6	6	6
200 003	100 000	mix	5	5	5
200 003	1 000 000	mod	19	19	17
200 003	1 000 000	mul	16	16	16
200 003	1 000 000	mix	17	17	17

m	n	hash	Top1	Top2	Top3
200 003	10 000 000	mod	84	83	83
200 003	10 000 000	mul	85	84	83
200 003	10 000 000	mix	89	87	84
2 000 003	10 000 000	mod	19	19	18
2 000 003	10 000 000	mul	19	19	19
2 000 003	10 000 000	mix	20	19	19
20 000 027	10 000 000	mod	7	7	7
20 000 027	10 000 000	mul	9	8	8
20 000 027	10 000 000	mix	8	7	7

Tabela 1 — Comprimento médio (Top1–Top3) das maiores listas encadeadas no método de encadeamento separado. Dados referentes a 3 repetições. Funções hash: mod (modular), mul (multiplicação), mix (xorshift + mod).

Análise típica:

À medida que n aumenta para além de m, as listas encadeadas crescem proporcionalmente, mas o top-3 se mantém relativamente pequeno devido à boa dispersão da função hash. Essa métrica é importante porque impacta diretamente o tempo médio de busca no encadeamento.

3.5 Gaps

m	n	hash	média	máx	mín
200 003	100 000	mod	1.55	22	0
200 003	1 000 000	mod	0.0068	2	0
2 000 003	100 000	mod	19.51	238	0
2 000 003	1 000 000	mod	1.54	26	0
20 000 027	100 000	mod	199.46	2469	0
20 000 027	1 000 000	mod	19.51	284	0
20 000 027	10 000 000	mod	1.55	31	0

Tabela 2 — Estatísticas de gaps (médio, máximo e mínimo) para encadeamento, função hash mod. A métrica de gap representa a distância entre buckets ocupados consecutivos.

Análise típica:

A Tabela 2 apresenta as estatísticas de gaps para a abordagem de encadeamento, considerando diferentes tamanhos de tabela e conjuntos de dados. Observa-se que:

- Para **tabelas pequenas (m = 200 003)**, o gap médio cai rapidamente para próximo de zero à medida que o número de elementos cresce, pois a maior parte dos buckets é ocupada.
- Para **tabelas médias e grandes** com **poucos elementos (n ≪ m)**, o gap médio é elevado (ex.: 199 em m = 20 milhões, n = 100 000), e o máximo pode chegar a milhares — indicando grandes intervalos vazios entre posições ocupadas. Isso é esperado, já que a tabela está quase vazia.
- À medida que a tabela é preenchida, os gaps médios diminuem drasticamente (ex.: 1.55 para m = 20 milhões e n = 10 milhões), refletindo uma ocupação mais homogênea.
- Essa métrica é particularmente relevante para métodos de endereçamento aberto, em que grandes gaps impactam diretamente o custo de sondagem. No encadeamento, o impacto é menor, mas os valores ainda ajudam a compreender a dispersão dos dados.

Resultados semelhantes foram observados para as funções mul e mix, com pequenas variações nos valores máximos. Optou-se por exibir apenas os dados da função mod para manter a tabela concisa e ilustrar claramente o comportamento típico.

3.6 Uso de memória (bônus)

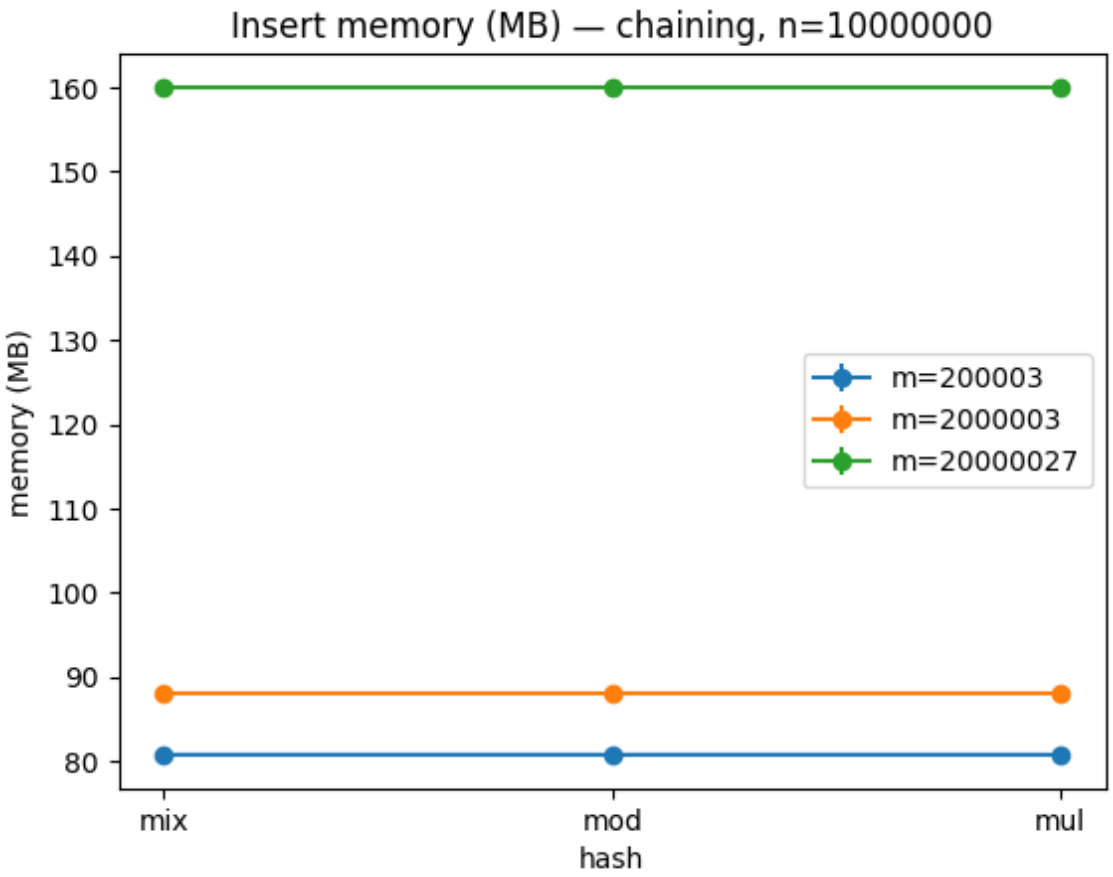

Figura 7 — Consumo de memória durante inserção para encadeamento separado.

Análise típica:

O encadeamento usou mais memória absoluta, pois mantém vetores auxiliares para head, next e keys. Contudo, por evitar objetos por elemento, manteve uso previsível mesmo com 10 milhões de elementos.
Linear e double usam apenas o array principal, então sua memória é proporcional a m.

4. Discussão

Os resultados confirmam os comportamentos clássicos das diferentes abordagens:

- **Encadeamento separado** é robusto quando n ≫ m, com tempo de inserção linear em n e colisões proporcionais ao excesso de elementos.
- **Sondagem linear** degrada fortemente à medida que o fator de carga se aproxima de 1.
- **Hash duplo** oferece melhor dispersão e tempos mais estáveis em altas cargas que linear.
- A escolha da **função hash** tem impacto direto na distribuição: funções multiplicativas ou baseadas em xorshift (mix) apresentaram melhor desempenho que o módulo simples.

- O uso de números primos como tamanho da tabela ajudou a reduzir padrões repetitivos.

5. Conclusão

Foi possível implementar e analisar três estratégias distintas de tabelas hash em Java de forma eficiente, validando tanto o comportamento esperado teórico quanto práticas de engenharia (uso de primos, funções hash adequadas e estruturas de dados leves).

Os experimentos mostraram que:

- **Encadeamento** é a abordagem mais flexível e robusta para grandes volumes de dados.
- **Hash duplo + função mix** apresentou os melhores resultados entre os métodos de endereçamento aberto.
- A função mod foi consistentemente inferior em distribuição.

Além disso, a execução com medições de memória e repetição demonstrou que implementações cuidadosas (com arrays em vez de objetos) podem atingir tempos muito inferiores aos observados em implementações “de slides”, mesmo com milhões de elementos.