Análise do Projeto de Tabela Hash

Como Montamos o Projeto

Nós decidimos criar uma classe HashMap abstrata que cuida de quase todo o trabalho: inserir (put), buscar (get), redimensionar (rehash) e contar colisões.

Dessa forma, nossas classes HashMapFunction1 e HashMapFunction2 ficaram simples: elas só precisam dizer como calcular o hash, chamando os métodos que criamos na Contagem.java.

Para colisões, usamos o método de encadeamento separado. Cada bucket da nossa tabela é, na verdade, uma Lista Duplamente Encadeada.

O Ponto Principal: Rehashing

A parte mais importante que implementamos foi o rehashing dinâmico. A tabela começa com capacidade 32, mas definimos que, quando ela atingisse 75% da capacidade, nós dobraríamos o tamanho dela e redistribuiríamos todas as chaves. Isso foi crucial para os resultados.

O que os Testes Mostraram

Nós rodamos os testes usando o Main.java, que leu os 5000 nomes do arquivo e mediu o tempo de inserção e busca.

Resultados da Função 1 (baseada em produtoVogaisConsoantes):

- Tempo de inserção: 50 ms
- Tempo de busca: 5 ms
- Número de colisões: 1302

Resultados da Função 2 (baseada em somaPesosVogaisMaisConsoantes):

- Tempo de inserção: 13 ms
- Tempo de busca: 3 ms
- Número de colisões: 1284

---

1. Colisões (1302 vs 1284): A Função 2 ganhou por pouco, com 18 colisões a menos. Isso faz sentido, porque a lógica dela é um pouco mais inteligente: ela usa pesos diferentes para cada vogal e também leva em conta a posição da letra (i + 1) e a operação XOR para misturar melhor as chaves.
2. Busca (5ms vs 3ms): Aqui está a prova de que nosso rehashing funcionou. Os tempos de busca foram muito rápidos. Isso acontece porque, ao inserir 5000 nomes, a tabela cresceu. No final, tínhamos 5000 nomes espalhados em 8192 baldes. Como a tabela estava "vazia", a maioria das listas tinha tamanho 1 ou 2.
3. Inserção (50ms vs 13ms): A Função 2 foi muito mais rápida para inserir. Acreditamos que isso seja por causa do "aquecimento" da JVM. Como a Função 1 rodou primeiro no

Main.java, ela já otimizou o código. Quando a Função 2 rodou, o Java já estava preparado e executou tudo (incluindo os vários rehashing) muito mais rápido.

Conclusão

Ambas as funções de hash que fizemos são ótimas. A Função 2 é tecnicamente melhor (menos colisões), mas a maior vitória do projeto foi a implementação do rehashing dinâmico, que garantiu que nossa tabela hash fosse extremamente rápida na busca, que é o objetivo principal.