Relatório de Projeto e Análise de Algoritmos

Marcos Paulo de Carvalho Lopes

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octavio Jordão Ramos, 6200 - Coroado I, CEP:69.080-900
Manaus – AM – Brasil

marcos.lopes@icomp.ufam.edu.br

1. Introdução

Em Projeto e Análise de Algoritmos (PAA), é de suma importância saber a natureza e limites de vários algoritmos e linguagens de programação para então escolher o que melhor se encaixa na resolução de um determinado problema. Neste relatório, será feito a análise de duas estruturas de dados, Tabela Hash e Árvore Rubro-Negra, em duas linguagens de programação diferentes, Java e C++, com objetivo de verificar o consumo de tempo e memória que cada estrutura e linguagem utilizam ao fazer uso dos recursos de cada uma. No, fim serão feitas considerações em relação aos resultados adquiridos e observados ao longo do estudo.

2. Teoria

Tabela Hash é uma estrutura de dados bastante utilizada na Ciência da Computação. Ela, na maioria das vezes, é usada para armazenar dados de grande volume, como arquivos de dados, além de ser bastante implementada para vetores associativos, caches e conjuntos. A tabela Hash é uma estrutura muito eficiente, pois a posição de onde o registro será posicionado é facilmente calculada com uma função hash, usada para fazer a dispersão dos dados. Com complexidade *O(1)* para suas funções básicas se a implementação for correta (baixo nível de colisões).

Uma árvore rubro-negra é uma árvore binária de pesquisa balanceada, uma estrutura de dados bastante utilizada na Ciência da Computação, usada geralmente para implementar vetores associativos. Ela torna-se complexa devido aos seus casos de pivotação para deixar sua estrutura balanceada, o que garante um bom pior-caso de tempo de execução para suas operações sendo bastante eficiente na prática: buscar, inserir, e remover são *O(log n)*, sendo n o número total de elementos da árvore.

3. Experimento

Para fazer o experimento, foi preciso baixar todas as dependências da linguagem C++ e Java para execução de um código, além disso foi utilizado os containers da própria linguagem que estão disponíveis para uso, como o *unordered map* e *map* no C++ e *Hashtable* e *TreeMap* em Java.

O ambiente de trabalho usado no experimento é um computador *desktop* Linux (Ubuntu), processador i3 e 8GB de memória RAM. Essas especificações são bastante relevantes nos alcances do limite superior de número de inserções proposto no trabalho, já que cada máquina tem um poder de processamento diferente, assim como um S.O pode tratar os seus processos de forma alternativa, o que pode gerar resultados divergentes.

No experimento, foram feitas 8 iterações em C++ variando a quantidade de entradas da tabela Hash e Árvore Rubro-Negra de 10 até 100000000 pares, aumentando sempre em 10 vezes. A cada iteração verifica-se o tempo de execução da inserção de n pares da iteração atual em uma estrutura e damos um pause na execução para fazer a análise da memória através do monitor do sistema para aquela determinada estrutura. Os resultados obtidos são anotados e então limpamos a estrutura de dados para fazer o mesmo procedimento para a próxima estrutura de dados. Feito todas as iterações e anotado os resultados, foi construído o gráfico proposto a seguir.

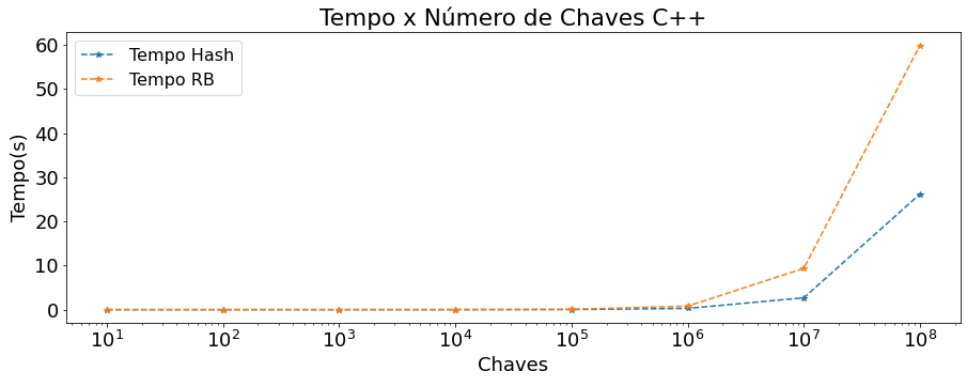

Figura 1. Gráfico do Tempo de Execução x Número de Chaves

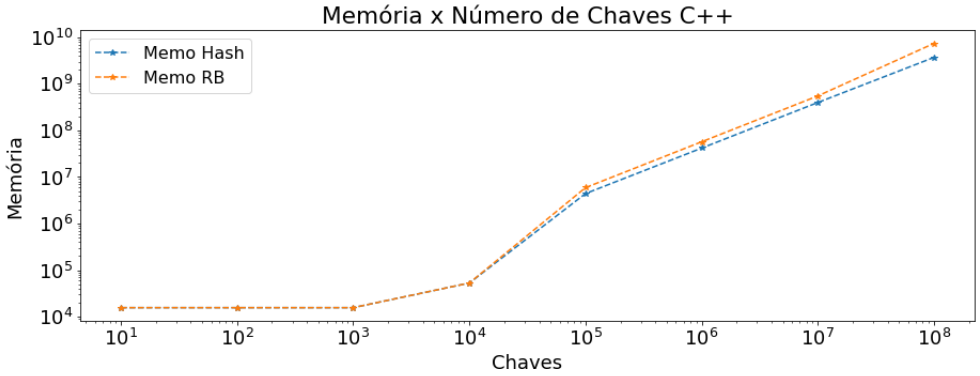

Figura 2. Gráfico do Consumo de Memória x Número de Chaves

Na linguagem Java, foram feitas 7 iterações, uma a menos que os testes na linguagem C++, variando a quantidade de entradas da tabela Hash e Árvore Rubro-Negra de 10 até 10000000 pares, aumentando em 10 vezes. A cada iteração verifica-se o tempo de execução da inserção de n pares da iteração atual em uma estrutura e damos um pause na execução para fazer a análise da memória através do monitor do sistema para aquela determinada estrutura. Os resultados obtidos são anotados e então limpamos a estrutura de dados para fazer o mesmo procedimento para a próxima estrutura de dados. Feito todas as iterações e anotado os resultados, foi construído o gráfico proposto a seguir.

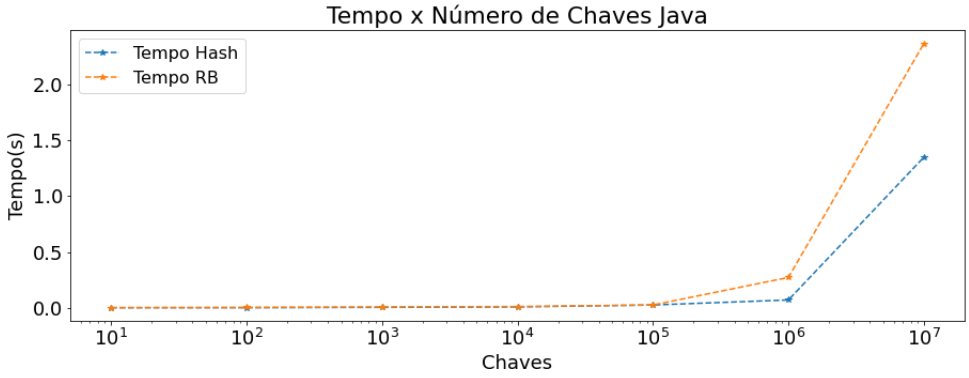

Figura 3. Gráfico do Tempo de Execução x Número de Chaves

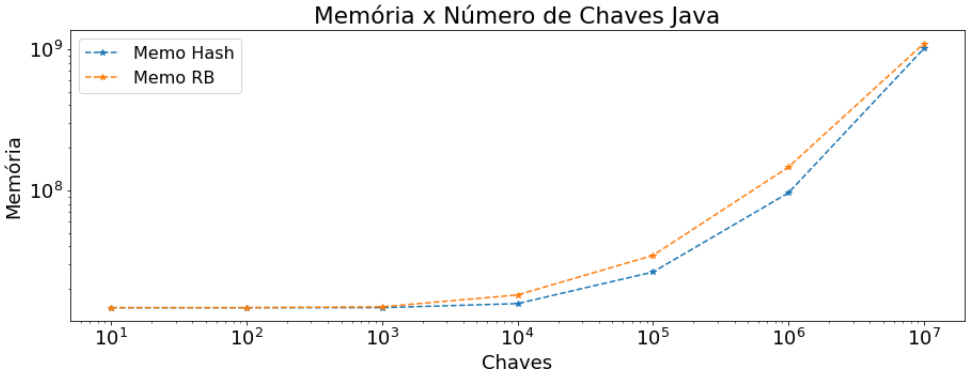

Figura 4. Gráfico do Consumo de Memória x Número de Chaves

Feito os gráficos, pode-se fazer algumas observações básicas sobre os primeiros resultados obtidos. Nos testes das duas estruturas, Java inicia o arquivo com mais ou menos de 15MB já alocados referente aos containers padrão, logo é mais rápido com alocações na hora da execução que C++, porém gera um problema de memória quando tentamos inserir grandes entradas nas estruturas, já que o computador em questão não conseguiu inserir mais de 10000000 elementos excedendo a memória do Heap, 10 vezes menos que no C++.

Cada par de chave e valor é implementado por dois inteiros, serão utilizados 8 bytes por cada elemento inserido. Além disso, um nó da árvore também tem um ponteiro na sua implementação, totalizando 24B para cada nó e 16B para cada campo hash. Com 100 milhões de entradas, são necessários 100000000 * 24B = 2.4GB para a árvore e 100000000 * 16B = 1.6GB para o Hash teoricamente, já que há mais armazenamento da própria linguagem devido a implementação e uso de containers. Esse comportamento, de acordo com os gráficos de memória, ficou mais organizado na linguagem C++, que seguiu o padrão teórico analisado desde o inicio da execução até o fim, com poucas variações que são explicadas por implementações da linguagem. Java, desde o início da execução, apresentou resultados divergentes aos teóricos para entradas pequenas e médias dado a alocação interna da própria linguagem, o que está explícito devido a diferença entre as escalas do eixo da memória das duas linguagens, C++ inicia com kbps enquanto Java inicia com mbps.

4. Conclusão

No caso da linguagem C++, os tempos de execução das duas estruturas ficaram bastante iguais com inserção de um número baixo de valores, até 1000000, mas com o avanço do número de elementos da inserção, a tabela Hash se mostrou mais promissora na hora de inserir, levando mais ou menos a metade do tempo de execução do limite superior proposto, 100000000. O que é esperado devido a complexidade das duas estruturas, já que Hash é *O(1)* enquanto Árvore RB é *O(log n)* para essa função. Em memória, a tabela Hash se mostrou levemente superior em todos os pontos na linguagem, o que pode ser explicado pela relação de bytes em cada estrutura, que é menor na implementação de Hash.

Na linguagem Java, os tempos de execução se mostraram muito iguais também até a inserção de 100000 elementos, ponto onde o Hash começa a ter um desempenho melhor em relação a Árvore Rubro-Negra. Também explicado pela complexidade de ambas. Em memória, as estruturas se mostraram bastante iguais até a inserção de 10000 elementos, onde Hash começa a se demonstrar melhor no uso de memória. O mesmo pode ser explicado por causa da implementação de bytes/estrutura também.

Em relação às duas linguagens, Java ganha em tempo de execução do que C++ na maioria dos casos, mas utiliza uma quantidade de memória maior em pequenos e médios casos e pode ser inviável para um grande número de entradas, e de acordo com os gráficos, ambas estruturas seguiram suas complexidades.