Kenji Henrique Ueyama Yashinishi - 2524627
Luiz Felipe Fuzeto - 2524635

Desempenho médio de cada estrutura (Calculado a partir de 10 testes cada)

Árvore AVL:

in1.txt - 2.058 ms., 4560 comparações

in2.txt - 2.010 ms., 4606 comparações

in3.txt - 2.005 ms., 4950 comparações

Tabela Hash Redimensionável com endereçamento aberto:

in1.txt - 1.396 ms., 500 comparações

in2.txt - 1.283 ms., 500 comparações

in3.txt - 5.065 ms., 37425 comparações

Tabela Hash de Tamanho Fixo com Encadeamento:

in1.txt - 1.390 ms., 2679 comparações

in2.txt - 1.418 ms., 2898 comparações

in3.txt - 2.283 ms., 27850 comparações

1. Qual arquivo representa o pior e o melhor caso para a tabela hash? Justifique sua resposta.

Pior: in3.txt, pois muitos valores causavam colisões, sendo necessário buscas lineares a partir do índice em que foi inserido por conta da sondagem linear, aumentando consequentemente o tempo de busca.

Melhor: in1.txt e in2.txt, além de apresentarem tempos de execução similares, seus valores não apresentaram colisões, possibilitando acesso direto à chave ao calcularmos o índice de busca dele.

2. Qual arquivo representa o pior e o melhor caso para a árvore AVL? Justifique sua resposta.

Pior: O arquivo in3.txt apresenta o pior desempenho devido à busca constante por chaves grandes. As entradas variam consideravelmente, resultando na necessidade da raiz da árvore percorrer longos caminhos durante a busca.

Melhor: Os arquivos in1.txt e in2.txt não requerem descida constante até o final da árvore, pois suas entradas não variam tanto entre eles mesmos. Eles conseguem localizar suas buscas de forma eficiente.

A velocidade de execução dos códigos não varia significativamente na árvore AVL, pois a complexidade tanto na implementação quanto na busca é, em média, O(log N). Nos piores casos, a complexidade também permanece como O(log N).

3. Se houver diferença de desempenho entre as estruturas de dados, explique o motivo dessa diferença?

O principal motivo que causa esta diferença está na forma em que é acessado o nodo em cada estrutura. Na árvore AVL, é necessário fazer a busca binária por meio de atualização de ponteiros, sendo obrigatoriamente necessário percorrer por diversos valores, a não ser que o valor sendo pesquisado seja a raiz em si, enquanto na tabela hash, ao encontrarmos o índice de posição da chave, calculado por meio de uma função (chave%capacidade neste caso), é possível acessar diretamente esta posição no vetor. Porém, caso haja muitas colisões em uma tabela hash, ao utilizarmos o método da sondagem linear para lidarmos com eles, há o risco de uma chave ocupar o índice de uma chave futura, causando a necessidade de uma busca linear, pois o índice calculado pode estar no meio de uma série de colisões anteriores, problema não presente na árvore AVL, pois ela sempre estará balanceada, sem a possibilidade de um desequilíbrio que possa causar buscas com custo acima da média.

Já a tabela hash com encadeamento separado, apresenta a vantagem de colisões não afetarem índices de chaves futuras, mantendo o problema da busca linear para apenas o índice que a chave pertencer.

Estrutura que apresentou melhor desempenho para cada entrada

in1.txt - Tabela Hash Redimensionável, pois não houve colisões, permitindo acesso instantâneo à chave por meio do índice, além do tamanho ser dobrado caso seu fator de carga atingisse 0.70.

in2.txt - Tabela Hash Redimensionável, mesmo motivo que a entrada in1.txt.

in3.txt - Árvore AVL, pois a mesma lida muito bem com colisões quando comparada com a tabela hash.