Relatório de AED-II

Júlia Chaparro

[1]Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
Av. Gen. Rodrigo Octávio 6200, Coroado I
69080-900 – Manaus – AM

julia.ferreira@icomp.ufam.edu.br

1. Árvore AVL

A árvore AVL é uma estrutura de dados que permite acesso eficiente aos dados, pois mantém a altura da árvore balanceada, garantindo um tempo de busca de O(logn) quando bem balanceada. Em uma árvore AVL, cada nó possui um fator de balanceamento, que é a diferença entre as alturas de sua subárvore direita e esquerda. Para manter o equilíbrio, esse fator deve ser igual a 0, -1 ou +1.

Quando o fator de balanceamento de um nó atinge +2 ou -2, uma rotação é realizada para corrigir o desequilíbrio. A inserção de um elemento segue as mesmas etapas de uma árvore binária de busca, mas com a diferença de que, após a inserção, os fatores de balanceamento são atualizados. Caso algum nó se torne desequilibrado, uma rotação é aplicada na raiz da subárvore afetada, restaurando o balanceamento da árvore.

2. Tabela Hash

Na tabela Hash são usados registros armazenados em uma tabela, endereçados a partir de uma transformação aritmética sobre a chave de pesquisa. O principal objetivo é ter eficiência O(1) nas operações de acesso, inserção e remoção.

3. Descrição do código AVL

- **ADD:** A função vai adicionar os elementos na AVL já utilizando as funções utilizadas na sala, a função vai receber como parâmetro todos os elementos dos sensores como Pump ID ,Class ID , Temperature , Vibration, Pressure, Flow Rate, RPM, Operational Hours, Maintenance Flag, a função vai percorrer a AVL e vai achar a chave ou vai criar uma novo no.
- **SEARCH:** Vai buscar na AVL e vai printar os itens buscados no arquivo de saída
- **REMOVER:** Vai percorrer a AVL e vai buscar o item que foi procurado, removendo a lista simplesmente encadeada(LSE) removendo uma por uma e quando chegar no final da lista quando o tamanho da LSE for 0 então podemos destruir a LSE e vai printar no arquivo de saída os itens que foram removidos.
- **REPORT MEAN:** Vai percorrer a AVL ate encontrar a LSE e vai executar a media dos Temperature , Vibration, Pressure, dos sensores que estão na LSE.
- **REPORT MAX:** Vai percorrer a AVL e ate encontrar a LSE e vai encontrar o maior dos Temperature , Vibration, Pressure, dos sensores que estão na LSE.
- **REPORT MIN:** Vai percorrer a AVL e ate encontrar a LSE e vai encontrar o menor dos Temperature , Vibration, Pressure, dos sensores que estão na LSE.

4. Descrição do código da TABELA HASH

- **ADD:** A função adicionará os elementos na tabela hash, onde cada entrada da tabela representa uma LSE (lista encadeada) para lidar com colisões. A função receberá todos os dados dos sensores como parâmetros: Pump ID, Class ID, Temperature, Vibration, Pressure, Flow Rate, RPM, Operational Hours, Maintenance Flag. O valor do Pump ID (ou outra chave identificadora) será usado para calcular o índice na tabela hash. Se a chave já existir na lista encadeada associada ao índice, o item será atualizado; caso contrário, um novo nó será adicionado.
- **SEARCH:** A função buscará o item desejado na tabela hash. Utilizando o Pump ID para calcular o índice, a função percorrerá a lista encadeada associada a esse índice. Se o item for encontrado, ele será exibido no arquivo de saída.
- **REMOVER:** A função removerá o item desejado da tabela hash. Utilizando o índice calculado pelo Pump ID, a função percorrerá a lista encadeada na posição correspondente e removerá o nó que contém o item buscado. Se a lista encadeada se tornar vazia após a remoção do item, a entrada da tabela hash será marcada como vazia. Todos os itens removidos serão registrados no arquivo de saída.
- **REPORT MEAN:** A função acessará o item na chave especificada e percorrerá a lista encadeada neste ítem. A função calculará a média dos valores de Temperature, Vibration e Pressure dos sensores armazenados nessa lista.
- **REPORT MAX:** Similar ao REPORT MEAN, a função percorrerá cada entrada da tabela hash e suas listas encadeadas. Para cada lista, a função encontrará os valores máximos de Temperature, Vibration e Pressure entre os sensores armazenados para determinar o valor máximo do item que fui procurado.
- **REPORT MIN:** A função seguirá o mesmo processo do REPORT MAX, mas, neste caso, buscará os valores mínimos de Temperature, Vibration e Pressure dos sensores nas listas encadeadas da tabela hash

5. Comparaçao de desempenho da arvore AVL e tabela Hash

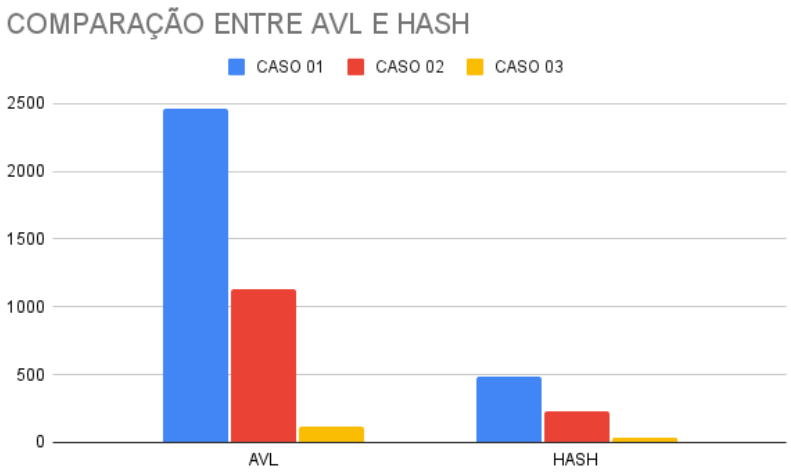

Figura 1. Números de comparações entre a hash e AVL

Para comparar o desempenho da Tabela Hash e da Árvore AVL, foram utilizados três arquivos de dados, cada um com um número específico de linhas, para observar as

diferenças no comportamento de cada estrutura em termos de colisões, comparações e rotações.

Caso 1: Ao utilizar um arquivo com 857 linhas, a Árvore AVL registrou 2.466 comparações e realizou 33 rotações. Em contraste, a Tabela Hash teve apenas 484 comparações e 20 colisões. Cada comando no arquivo foi projetado para forçar comparações nas duas estruturas, revelando uma clara diferença de desempenho em favor da Tabela Hash em termos de menor número de comparações e colisões.

Caso 2: Com um arquivo de 539 linhas, a Árvore AVL registrou 1.132 comparações e 28 rotações. A Tabela Hash, por sua vez, teve 234 comparações e 13 colisões. Esse caso também destacou a eficiência da Tabela Hash em comparação com a Árvore AVL ao lidar com um volume intermediário de dados.

Caso 3: No teste com um arquivo de 271 linhas, a Árvore AVL teve 119 comparações e realizou 14 rotações, enquanto a Tabela Hash registrou 34 comparações e apenas 6 colisões. Este último caso, com menor quantidade de dados, ainda demonstrou que a Tabela Hash apresentou menor número de comparações e colisões.

Esses testes evidenciam que, com o aumento do número de entradas, a Tabela Hash mantém uma vantagem em eficiência sobre a Árvore AVL, exigindo menos rotações e comparações, especialmente à medida que o volume de dados crescer

A análise demonstra que, com o aumento do número de dados (como observado do Caso 3 para o Caso 1), a AVL realiza cada vez mais rotações e comparações para manter-se balanceada. Isso sugere que, para grandes volumes de dados, a Tabela Hash tende a ser mais eficiente na maioria das operações básicas (inserção e acesso).

6. Adequaçao para Processamento em Batch

As arvores AVL, com sua altura minimizada, podem oferecer melhor desempenho para cargas de trabalho pesadas e processamento em batch, onde muitas operações de busca são necessárias. Por outro lado, a Tabela Hash é mais eficiente em cenários com muitas inserções e remoções devido ao menor numero de colisões.

7. Discussão sobre desafios encontrados e como foram abordados

Implementar corretamente as rotações para garantir que a arvore AVL permaneça balanceada, além de manter os fatores de balanceamento atualizados apos inserções e remoções, foi uma tarefa complicada.

Implementar corretamente a Rehashing para garantir o funcionamento adequado da hash para a alocação de memoria para um novo espaço, entender a funcionalidade de algumas funções e como utilizar corretamente, foram tarefas desafiadoras de se implementar.