Centro Educacional Tecnológico Celso Sudckow da Fonseca CEFET/RJ
Engenharia de computação
AEDS II

Relatório sobre trabalho de AVL

Aluno: Ernst Franz Karl Zeidler Neto
Professor orientador: Laura Assis

Petrópolis
18 de maio de 2023

Centro Educacional Tecnológico Celso Sudckow da Fonseca CEFET/RJ
Engenharia de computação
AEDS II

Relatório

Primeiro Relatório de trabalho sobre árvore AVL apresentado na turma de AEDS II do Curso Engenharia de computação da Universidade CEFET/RJ como requisito parcial para o trabalho 1.

Aluno: Ernst Zeidler Neto

Professora orientadora: Laura

18 de maio de 2023

Conteúdo

1 Resumo

O trabalho proposto sobre a arvore AVL (arvore binária) é constituído por vários nos conectados entre a folha raiz (origem) folhas nós (posição atual) e nós folhas (nós filhos). Quando a arvore é criada, podemos notar que um nó armazena: altura, chave, filho esquerda, filho direita. Cada nó possuem seus atribuídos de valores próprios e dois ponteiros de endereço que são direcionados para os respectivos nós: esquerda e direita.

A construção da arvore vai crescendo de acordo com o percusso sendo guiado pelo algoritmo e assim como inserir novo nó folha. Contudo, é necessário conferir se pode haver conflito ou duplicidade de dados iguais e principalmente a arvore nao cresce de forma desordenada ou não balanceada. Com isso é imprescindível o fator de valor altura para cada no, se houver desbalanço, é necessário ordenar-la e rebalancear, de acordo com as regras e princípios a serem atingidos ao decorrer do relatórios.

2 Introdução

O conceito abordado sobre a estrutura de dados ideais, tem o seu objetivo de estudar as propriedades de um arvore binária AVL, que foi introduzido ou criado em 1962 por Adelson-Velskii e Landis. Uma árvore AVL é uma árvore de busca binária que é balanceada com respeito a altura do no em mediante ao calculo da altura das suas sub-folha ou altura das subárvores. Ela possui a característica que sua busca é realizada em relação ao tempo de resposta na equação(1) abaixo. se a árvore possui N nós. Logo, se formos fazer uma busca torna-se interessante a sua implementação em um sistema de controle de armazenamento de dados.

$$O(log_2 N) \tag{1}$$

As árvores de busca binária são projetadas para fornecer acesso rápido às informações. O ideal é que a árvore esteja equilibrada e mostre sua altura (quando a base estiver completa). É possível descrever que a altura de uma arvore AVL pode ser descrita como.

$$h = O(log_2 N) + 1 \tag{2}$$

Com o crescimento de nos na arvore, é necessário sempre ao inserir ou remover um nó, é necessário balancear a arvore com a sua medida de peso determinado ou tolerado, afim de permanecer em equilíbrio, seja sempre percorrer nós N, verifica se o peso esta dentro do tolerável sendo nos intervalos.

$$aceito : n- > altura >= -1 \quad (3)$$

$$aceito : n- > altura <= +1 \quad (4)$$

Quando verificamos a variável altura, caso o valor for aceito, podemos continuar com a verificação dos altura entre as suas subárvores esquerda e direita. Caso a altura esteja fora do aceitável (altura maior ou igual a +2 ou peso menor ou igual a -2), vai ocorrer o processo de balanceamento e escolher o tipo adequado para organizar a arvore a ponto de retornar o seu estado equilibrado, em relação aos fatores serão abordados no próximo tópico.

3 Desenvolvimento

3.1 Construção da struct

O desenvolvimento da arvore AVL foi desenvolvida na linguagem C, é imprescindível a construção de struct, pois em cada nó vai conter mais do que uma informação a ser processada. Por determinação da regra da arvore AVL, é preciso ter: uma variável inteiro altura, variável char chave, struct de lista_palavras, duas variáveis de ponteiro de endereço apontando com exclusividade para o(s) nó(s) filho(s) da esquerda e direita e se houver filho(s), o(s) filho(s) da esquerda e/ou direita.

Contando que não existe apenas uma struct No, existe dentro da struct No em cada nó existente uma struct Palavra que é a variável lista_palavras. Ela define uma lista encadeada simples ordenada de palavras que contém a primeira letra como a referencia da chave do nó, por exemplo: palavra: ABACAXI com a chave: A do nó existente. Enquanto isso, na operação de busca_palavra, inserção_palavra e remoção_palavra, é necessário o percusso da lista encadeada de que possa ser alocada dinamicamente e ser ordenada. Podemos entender que precisamos criar uma segunda struct e nela possa ser introduzida dentro da struct No.

Percebendo que existe dentro da struct no, temos uma struct lista palavras, refere-se a outra struct de uma lista encadeada simples ordenada para um dicionario de palavras, tanto o no e lista de palavras são alocados dinamicamente. Na lista tem uma struct de endereço apontando para o próximo ate chegar no final da lista ou encontrar NULL.

3.2 Construção do nó

Para uma inserção primeiro nó de arvore vazia ou a arvore existir, caso a letra chave ou primeira letra da palavra não contem um nó existente. Se for

para o caso de inserção de palavras, temos uma função de construir no e a sua função tem o retorno de struct No.

3.3 Balanceamento

Para cada operação de inserção ou remoção de um nó da arvore, é necessário recalcular a altura da arvore e caso se ocorrer desbalanceamento da arvore, processos de rotações dos nós adequados serão acionados e executados para restabelecer o balanceamento. Caso precisar fazer a operação de rotação dos nós, na árvore AVL existe 4 tipos de rotações: LL(esquerda), RR(direita), LR(Esquerda e direita), RL(direita e esquerda).

Através da função balancear, utilizamos a função fator de balanceamento, caso o valor fb (altura) do nó atual estiver fora do intervalo de

$$-1 < fb < +1 \tag{5}$$

Caso um das condições for verdade, função adequada de rotação será executada para rotacionar os nós e restabelecer o equilíbrio da altura permitida.

Mediante na função fatorDeBalanceamento, usamos a função de alturaDoNo para retornar o valor da altura do nó referente.

3.3.1 Rotação LL

Rotação LL ou somente para esquerda, quando a altura do no da subarvore da direita tem valor de 2 ou mais de altura de diferença comparando com a altura da subarvore esquerda. Ocasionando o valor da altura do no desbalanceado para a esquerda e seu no filho da direita ocupa o seu lugar.

3.3.2 Rotação RR

Semelhante a rotação LL, ocorre que a rotação se desloca para a direita, pois a altura da subárvore da esquerda é igual ou maior que 2 em comparação a subárvore da direta.

3.3.3 Rotação LR

Ocorre duas rotações, nesse caso faz a primeira a rotação da esquerda(LL) e depois a rotação da direita(RR).

3.3.4 Rotação RL

Ocorre duas rotações, nesse caso faz a primeira a rotação da direita(RR) e depois a rotação da esquerda(LL).

3.3.5 Maior

Nas funções de rotações LL e RR, é necessário chamar a função de altura na qual retornar o valor do qual nó altura esquerda ou direita é maior.

3.4 Buscar

Enquanto desejamos buscar uma palavra e/ou chave de um nó da arvore, foi necessário criar duas funções de busca: buscar_no, buscar_palavra. Na main, temos a opção de buscar somente o carácter chave para encontrar todas as palavras que iniciem com a primeira letra da palavra, por exemplo: A. Se existir um nó 'A' vai imprimir todas as palavras que iniciem com a letra 'A' e vai informal em qual nível o nó está localizado na arvore, caso contrário irá imprimir nó inexistente!.

A outra opção de busca é a busca pela palavra, caso deseje procurar uma palavra "ABACAXI", é preciso pegar a primeira letra como referencia de chave para o nó da arvore para a a busca de no. Se existir o no chave 'A', vai entrar retornar o nó com esta chave e vai entrar em uma outra função de buscar palavra em que a variável (Palavra* lista_palavras) é uma struct que se destaca como uma lista encadeada simples e ordenada. Através dela, percorremos nesta variável ate encontrar a palavra desejada. Caso encontrar a palavra na lista vai imprimir a palavra que existe na lista, caso contrario vai imprimir palavra inexistente!. Caso não encontrar o nó com a busca de chave, vai imprimir nó inexistente!.

3.5 Inserir

Enquanto desejamos inserir palavras dentro de uma das opções da main, podemos inserir sequencias de palavras e sempre conferir se a entrada de dados de uma palavras desde seja diferente de "0". Se encontrar a palavra "0"na entrada de palavras, a condição de inserção é abortada e retorna ao menu principal. Enquanto, inserimos as palavras, é preciso estar atento para evitar a inserção de palavras iguais em uma lista de palavras. Para isso, precisamos usar uma função citada anteriormente (buscar_no). Em cada inserção de palavra, a primeira letra da palavra sera a referencia de chave para a busca do nó, caso não encontrar o nó será necessário criar um novo nó com a sua primeira palavra e fazer o balanceamento apos a inserção do nó na arvore.

Caso existir um nó com a chave correspondido, apenas utilizar a função inserir_palavra e confere se existe a palavra na lista antes de fazer a inserção de palavra. Se existir a palavra, não ocorre a inserção de palavra. Caso

contrário, vai alocar o es paco para a nova palavra na lista de palavra do nó de forma ordenada, nesse caso, o auxilio de variava veis de anterior e depois são utilizados para redirecionar os ponteiros de endereço da lista encadeada.

3.6 Remoção

Em uma das opções da main, ocorre uma entrada de palavra e pega a primeira letra como referencia de chave e novamente utilizar a função buscar_no. Se existir o no, entrar na lista_palavra e fazer a percusso para encontrar a palavra como a mesma ideia de busca de palavra numa lista encadeada, caso encontrar a palavra, sempre conferir se a lista encadeada se a palavra é a única palavra existente ou possuem mais palavras. Caso existir somente esta palavra na lista encadeada, libera o espaço alocado nesta lista encadeada e também liberar o espaço alocado para este nó e recalcular a altura e balanceamento. Caso existir outras palavras na lista encadeada, usar as variáveis auxiliares para poder modificar os ponteiros de endereço corretamente e liberar o espaço alocado para esta palavra.

3.7 Imprimir

Há duas opções de imprimir na main, imprimir somente o nó desejado ou imprimir a arvore. imprimir um nó basicamente faz a entrada de um char de chave e utilizar a função buscar_no para encontrar se o no existe, caso se existir, imprimir a sua chave e seu numero nível de percusso da arvore, desde a raiz até o seu nó atual e imprimir todas as palavras existentes neste nó.

A outra opção de imprimir toda a arvore, é preciso imprimir a arvore de forma in-order, ou seja, imprimir primeiro o nó da esquerda, segundo a origem (meio) e por último da direita. in-order: (esquerda, meio, direita).

4 testes

Ao efetuar os testes do algoritmo implementado, podemos utilizar o casos de teste disponível. Usando o arquivo disponibilizado o arquivo palavraordem.txt, podemos fazer os fatores de teste de inserção de palavras de quase 30.000 palavras. Lembrando que a função de inserção, remoção e buscar de uma arvore AVL é definida na equação(1).

Plotando o gráfico em relação inserção de palavras decorrente ao tempo de execução:

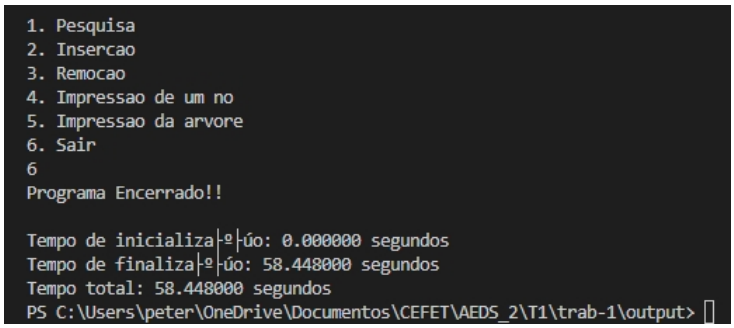

Figura 1: tempo gerado para inserção das palavras da palavraordem.txt

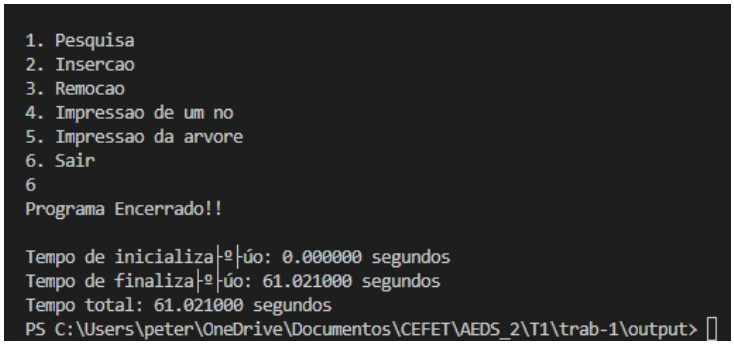

Figura 2: tempo gerado para inserção e depois liberação do espaço alocado da variável raiz das palavras da palavraordem.txt

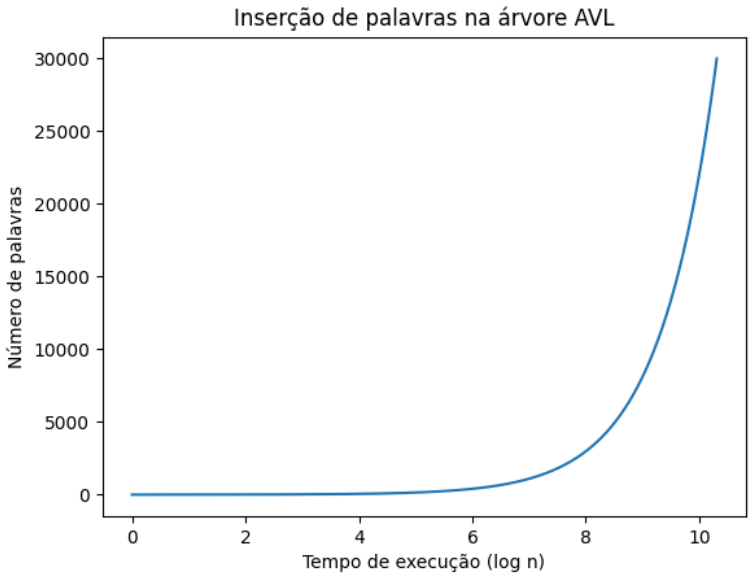

Figura 3: Figura do gráfico representa a inserção de 30.000 palavras na arvore AVL

5 Conclusões

Ao analisar as funções citadas anteriormente, podemos descrever ao inserirmos as palavras e percebendo que o gráfico vai aumentando de forma exponencial de acordo com o numero de nós criados e a quantidade de palavras a serem inseridas. Podemos dizer que o gráfico apresenta uma função conhecida no mundo acadêmico.

$$f(n) = \epsilon^n \tag{6}$$

De acordo com o aumento de numero de nós e palavras, lembrando que a inserção e remoção é necessário conferir e se for o caso de balancear a arvore para afim de permanecer a função da equação(1).

Bibliografia

ASSIS, Laura Aula1 AEDsII AVL Busca e Inserção. CEFET/RJ Uned: Petropolis, Rio de Janeiro, Brasil.
ASSIS, Laura Aula2 AEDsII AVL Remoção. CEFET/RJ Uned: Petropolis, Rio de Janeiro, Brasil.