Trabalho 2: Árvore AVL
Caio Cesar Aguiar Alarcon n° 7241109
24 de outubro de 2018

1. Introdução

O trabalho atual tem como propósito a criação de uma árvore de busca, preferencialmente tipo AVL e verificar suas implicações.

2. Procedimento

-O código

Primeiramente o tipo abstrato de dado foi definido como sendo um struct portando cinco registros: ch, valor, prof, es, di sendo, respectivamente: a chave, o valor, a profundidade, referência para o filho esquerdo e direito. A profundidade foi definida como sendo menor para as folhas e maior para a raíz da árvore. O tipo valor, no caso, é inteiro. Porém para aplicações práticas é recomendado que ele fosse de tipo definível ou tipo ponteiro void para que possa receber qualquer tipo de dado. Além disso o tipo nó foi definido como sendo tipo ponteiro para o struct pois isso facilita na hora de lidar com ponteiros ou ponteiros para ponteiros em diversos momentos onde é necessário trocar os nós de lugar e mudar a referência da raíz da árvore.

A função ‘insere()’ recebe como parâmetros um ponteiro para tipo no, a chave e o valor, ambos inteiros. Sendo assim, a sua chamada deve receber o endereço de um tipo nó para que seja possível modificar o ponteiro que contém a árvore fazendo modificações úteis, como mudando o nó raiz por exemplo. Ela é uma função recursiva onde ao final chama a função de balanceamento ‘balancear()’ .

A função ‘balancear()’ é responsável por analisar os casos e fazer as rotações necessárias de modo que o balanceamento ocorra. Ela chama as funções ‘giraE()’ e ‘giraD’ que rotacionam para a esquerda ou direita, respectivamente. Além disso usa algumas funçoes auxiliares como ‘fatorB()’, responsável por fornecer o fator de balanceamento e ‘ActProf()’, responsável por atualizar a profundidade de um determinado nó baseando-se na maior profundidade dos seus filhos.

A função ‘rem()’ é responsável por remover um valor especificado. Ela faz isso identificando cada caso e agindo de acordo. Ela usa a função ‘free()’ para liberar da memória o nó que não será mais usado. Usa também a função ‘meNor()’ que é responsável por fornecer o menor nó de uma sub árvore. Isso é útil para caso seja necessário tirar um nó que possui dois filhos, o substituto é o menor nó da direita. Poderia ser o maior da esquerda também. Usa

ainda, assim como a função ‘insere()’, a função de balanceamento para que quando um nó seja removido a árvore seja balanceada também.

A função ‘busca()’ recebe um tipo no apenas, pois não precisa fazer qualquer alteração nele e também é recursiva e tem como objetivo encontrar uma chave determinada. Ela retorna o endereço do nó para que seja possível buscar tanto o valor quanto a profundidade onde o nó se encontra. Ela faz uma busca binária recursiva. Caso a árvore recebida esteja vazia ou a chave não for encontrada, retorna NULL, indicando a falha.

A função ‘preO()’ recebe tipo n pois também não necessita fazer qualquer alteração na estrutura da árvore, é recursiva e imprime os valores referentes à chave, valor e profundidade de cada nó.

A função ‘crlf()’ tem por objetivo facilitar a impressão de uma quebra de linha na tela, apenas devido ao fato de ser complicado fazer essa impressão utilizando a função recursiva de impressão

No programa principal há um loop que exibe o menu mostrando as opções solicitadas pela descrição do projeto, porém existem outras funcionalidades com o propósito de análise. Há uma função de impressão da árvore em pré ordem, uma opção de exibição da profundidade atual da árvore e, ainda, uma função que salva em um arquivo de texto chamado ‘saida.log’, para cada operação executada, uma linha contendo a profundidade dela e a quantidade de elementos inseridos nela, ambos separados por uma tabulação.

-O programa auxiliar

Além do programa principal, um programa auxiliar foi criado para facilitar a criação de árvores com centenas de elementos de modo que ficasse fácil de verificar erros e sua profundidade, algo crucial para verificar se a árvore está mesmo balanceada ou não.

O programa cria arquivos com comandos sequenciais e, ao enviar a saída desse programa para a entrada do programa principal, é possível observar se o resultado é o desejado ou não. São seis arquivos criados, o primeiro grupo de três insere, remove e busca valores gerados aleatoriamente. O segundo grupo efetua as mesmas operações, porém com um conjunto de valores sequenciais.

-A realização dos testes

Os testes consistem em analisar o log gerado para cada uma das entradas de teste. Conforme os arquivos ‘saida.log’ eram gerados, um novo nome a eles era dado para referenciar o teste que os gerou. Além disso, um teste adicional foi realizado desativando o balanceamento da árvore e observando o resultado do arquivo log para a entrada aleatória apenas, uma vez que o resultado para entradas sequenciais é conhecido.

O arquivo log para a função de busca foi desprezado uma vez que essa operação apenas analisa a árvore sem remover ou inserir qualquer elemento. Analisou-se apenas a saída impressa na tela com a intenção de comprovar sua integridade. A eficiência da função de

busca não foi levada em consideração pois é esperado que a altura da árvore revele essa informação com maior acurácia.

3. Resultados

Ao comparar o log da inserção aleatória e sequencial repara-se, primeiramente, que o número de elementos inseridos é diferente nos dois casos. Isso acontece pois, como a entrada utilizada é aleatória, a ocorrência de elementos repetidos é esperada. Além disso notou-se que a inserção aleatória resultou em uma árvore com doze camadas, enquanto que a inserção sequencial resultou em uma árvore com apenas onze, ambas com aproximadamente mil elementos. Como a árvore sequencial possuía apenas um elemento na camada onze, isso significa que a diferença de elementos entre as duas árvores era pelo menos a camada onze inteira. Já que a camada onze consegue comportar dois mil e quarenta e oito elementos, não é uma diferença desprezível.

Já considerando os testes com balanceamento desativado, a árvore alcançou vinte e uma camadas na inserção de valores aleatórios, um valor realmente alto considerando a quantidade de elementos que uma árvore dessa altura poderia de fato comportar. Apesar disso não se compara com os mil e vinte e quatro níveis que a árvore alcançaria caso uma entrada sequencial fosse nela inserida também com o balanceamento desativado.

Em relação ao teste de busca, a observação da saída mostrada na tela revelou que a função de busca está funcionando como o esperado.

4. Conclusão

Apesar de ficar claro que o balanceamento oferecido pela AVL não otimiza totalmente a altura da árvore, por ser um algoritmo simples e eficiente existe uma grande vantagem em utilizar esse tipo de balanceamento quando se trata de armazenar dados em uma base que ofereça conferência otimizada para busca. Inserção e remoção aparentam ser muito custosas, então se a otimização dessas operações for crucial, o ideal é buscar outro tipo de estrutura.