Árvore AVL

Eduardo Stefanel Paludo (GRR20210581)
Fernando Gbur dos Santos (GRR20211761)
Universidade Federal do Paraná (UFPR)
Curitiba, Brasil

I. Introdução

O projeto desenvolvido consiste na implementação de uma árvore AVL na linguagem C. Para isso, foram escritos quatro arquivos: o programa principal (main.c), uma biblioteca com as principais funções da árvore (libavl.c), um arquivo de header (libavl.h) e um Makefile. O programa foi compilado através do comando *make*, com os parâmetros *-Wall* e *-std=c99*.

As funções implementadas foram: novo-nodo, deletar-arvore, emordem, nodo-max, max, altura, fator-balanceamento, rot-esq, rot-dir, inserir-nodo, remover-nodo. Após a conclusão do projeto, foram comparadas as saídas para cada caso de teste com a saída esperada do programa.

II. Programa principal

No arquivo principal (*main.c*), o ponteiro para a raiz da árvore foi inicializado como *NULL*. Em seguida, para ler o arquivo recebido pela linha de comando, foi utilizado um *while (!feof(stdin))*, em conjunto com um *scanf* que lê um caractere. Quando o caractere lido corresponde à instrução de inserir na árvore (i), um inteiro é lido e inserido na árvore. O mesmo ocorre com a instrução de remoção. Após finalizar a leitura do arquivo, é executada a impressão da chave e da altura (em relação à raiz) de cada nodo da árvore, pela travessia *emordem*. Por fim, a árvore é deletada e o programa encerra.

III. Estrutura

Para implementar a árvore, foi utilizada uma *struct* correspondente a um nodo, contendo a chave (*int*), altura (*int*) e ponteiros para os filhos da esquerda e direita, que também são nodos. A função novo-nodo aloca espaço para um novo nodo, inicializa sua chave com o valor passado como parâmetro, inicializa a altura como 0 (pois sempre é inserido na folha), inicializa os filhos da esquerda e da direita como *NULL* e retorna o ponteiro para esse nodo.

IV. Inserção

A inserção e a remoção na árvore são feitas de maneira recursiva. No caso da inserção, o programa cria um novo nodo e o retorna caso tenha chegado em uma folha da árvore (caso base). Caso contrário, se o valor a ser inserido for menor que a chave do nodo atual, a função inserir-nodo é chamada recursivamente para a subárvore da esquerda. Caso seja maior ou igual, é chamada para a subárvore da direita.

Em seguida, a altura dos nodos percorridos é atualizada, com base na maior altura entre a subárvore da esquerda e da direita. Logo após, é calculado o fator de balanceamento do nodo, ou seja, a diferença entre a altura da subárvore da esquerda e da direita. Se esse valor não estiver no intervalo permitido ($-1 \leq fb \leq 1$), é feito o balanceamento da árvore.

Existem quatro casos possíveis de desbalanceamento: *zig-zig* na esquerda/direita ou *zig-zag* na esquerda/direita. O programa encontra o caso certo e executa o procedimento adequado para cada um. Como a função retorna um ponteiro para nodo, os filhos dos nodos chamados recursivamente são atualizados após o balanceamento.

V. Remoção

A remoção ocorre de modo similar à inserção, mas possui 3 casos distintos quanto ao nodo a ser removido: ele pode ter 0, 1 ou 2 filhos. Caso não tenha filhos, é simplesmente executado um *free(nodo)*. Caso tenha 1 filho, o pai do nodo a ser removido recebe ou o filho da esquerda ou da direita do nodo. Caso tenha 2 filhos, é encontrado o antecessor do nodo a ser removido (o maior valor na árvore menor que ele). O valor do antecessor é copiado para o nodo, e a função é chamada recursivamente para remover a duplicata (antecessor).

Assim como na inserção, a altura dos nodos anteriores é atualizada, o fator de balanceamento é calculado e o método de balanceamento adequado para o caso é executado. No entanto, enquanto na inserção é utilizado o valor de um dos filhos como condição, na remoção é utilizado o fator de balanceamento de um dos filhos.

VI. Rotações

Para balancear a árvore depois de uma inserção ou remoção, são executadas rotações na árvore. Elas alteram a altura da árvore, mas mantêm a propriedade de BST (o filho da esquerda de qualquer nodo é sempre menor que ele, e o da direita é sempre maior ou igual).

Na rotação para a esquerda, o filho da direita (y) de um nodo x se torna seu pai; e o nodo da esquerda de y se torna o filho da direita de x. Em seguida, a altura de x e y é atualizada.

A rotação para a direita é similar, mas em vez de y ser o filho da direita de x, é o da esquerda; e x se torna o filho da direita de y.

VII. Conclusão

Após a conclusão dos algoritmos, foram executados testes conforme especificado, utilizando os comandos *./myavl* e *diff*. As saídas obtidas foram compatíveis com as saídas esperadas. Além disso, não foi observado nenhum erro de alocação de memória e inicialização de variáveis, através da execução do programa pelo *Valgrind*.