Implementação da Árvore AVL

Trabalho 1 - Algoritmos e Estruturas de Dados III

João Armênio Silveira e Fábio Naconeczny da Silva

Uma árvore AVL é uma árvore binária de busca balanceada, ou seja, o fator de balanceamento de um nó, pertencente a essa árvore, não pode sair do intervalo entre +1 e -1. Devido a isso, após cada inclusão ou remoção de um nó, é necessário fazer um processo de balanceamento, que consiste de rotações entre a raiz e seus filhos.

Acerca da estrutura de uma árvore AVL, existem duas diferenças em relação a BST padrão, é a adição das propriedades novas, a altura, utilizada no cálculo do fator de balanceamento (altura da subárvore da esquerda - altura da subárvore da direita) e o nível, utilizado no cálculo da profundidade de uma árvore.

```c
typedef struct No {
    int chave;
    int altura;
    int nivel;
    struct No *pai;
    struct No *esq;
    struct No *dir;
} No_t;
```

```c
// Calcula o Fator de Balanceamento de um no qualquer da arvore
int calculaFb(No_t *no)
{
    if (no->esq == NULL && no->dir == NULL)
        return 0;
```

Foi criada então uma função que calcula a altura de um nó, **alturaNo( ),** e uma que ao ser chamada atualiza a altura de todos nós presentes nas subárvores da esquerda e da direita do nó chamado, **atualizaAlt( )**, que é chamada toda vez que há uma movimentação na montagem da árvore. Ambas são executadas de modo recursivo.

Quando ocorre uma inclusão, após um novo nó com a chave ser incluído em sua posição favorável na árvore, há a atualização da altura de todos os nó, em seguida a verificação do fator de balanceamento, com a chamada da função **balanceiaAvl( )**, essa função examina se há a necessidade de uma rotação, sendo ela simples ou dupa.

As rotações implementadas foram a rotação para esquerda **rotDir( )** e rotação para esquerda **rotEsq( )**. Suas derivadas, a rotação dupla para esquerda e dupla para direita são executadas diretamente na função **balanceiaAvl( ).**

```c
No_t *balanceiaAvl(No_t* raiz, int chave)
{
    No_t* noBal;

    if(calculaFb(raiz) > 1){
        if (calculaFb(raiz->esq) < 0)
            raiz->esq = rotEsq(raiz->esq);
        noBal = rotDir(raiz);
    }
```

```c
    else if (calculaFb(raiz) < -1){
        if(calculaFb(raiz->dir) > 0)
            raiz->dir = rotDir(raiz->dir);
        noBal = rotEsq(raiz);
    }

    atualizaAlt(raiz);

    return noBal;
}
```

```c
// Realiza a rotação para esquerda e retorna a nova raiz
No_t *rotEsq(No_t *x)
{
    No_t *y = x->dir;
    No_t* T2 = y->esq;

    // Perform rotation
    y->esq = x;
    x->dir = T2;

    //  Update heights
    atualizaAlt(x);
    atualizaAlt(y);

    // Return new root
    return y;
}
```

```c
// Realiza a rotação para direita e retorna a nova raiz
No_t *rotDir(No_t *y)
{
    No_t *x = y->esq;
    No_t *T2 = x->dir;

    // Executa rotação
    x->dir = y;
    y->esq = T2;

    // Atualiza altura da arvore
    atualizaAlt(y);
    atualizaAlt(x);

    return x;
}
```

A função de exclusão, **remocao( )**, primeiro busca qual é o nó que contém a chave desejada. Ao encontrar, olhando para o número de filhos que o nodo tem, ele realiza o ajuste de ponteiros. Após isso ocorre novamente a atualização da altura de todos nós da árvore, e em seguida o balanceamento da árvore, caso necessário.

Para que haja a impressão como requisitada, é necessário calcular o nível/profundidade de um nodo. Essa função é feita usando uma leitura pré-ordem, onde o primeiro elemento lido é a matriz, que recebe zero, e em seguida, a cada recursão, o contador de profundidade vai aumentando.

```c
void profundidadeNo(No_t *no, int nivel)
{
    if (no == NULL)
        return;

    printf("%d, altura:%d nivel:%d\n", no->chave, alturaNo(no), nivel);
    nivel++;
    profundidadeNo(no->esq, nivel);
    profundidadeNo(no->dir, nivel);
}
```

Por fim há a função **destroiArvore( )**, que libera a memória alocada em todo processo.

```c
// Libera a memoria de toda arvore
void destroiArvore(No_t *raiz)
{
    if (raiz == NULL)
        return;
    destroiArvore(raiz->esq);
    destroiArvore(raiz->dir);
    free(raiz);
}
```