ALGORITMOS E ESTRUTURAS DE DADOS I – T3

Giovanna Borba e Arthur Mazot

INTRODUÇÃO

Atráves do estudo de Algoritmos e Estruturas de dados entramos em contato com o conceito de Árvores de pesquisa, que nada mais são que estruturas organizadas hierarquicamente que fácilitam a adição, remoção e busca de seus elementos. O modelo de ordenamento implementado nessas estruturas se baseia na categorização dos elementos de menor valor (esquerda) e maior valor (direita) a medida que são adicionados na árvore. Apesar do seu processamento eficaz, a complexidade de uma Árvore de Pesquisa em seu pior caso será igual a sua altura, onde pode ser O(N), na qual N é a quantidade de nós existentes na árvore. Essa eficiência, porém, pode ser ampliada se houver um balanceamento interno dos nós existentes, com pior caso de complexidade O(Log N).

Árvores Balanceadas (AVL) são um tipo especifico de árvore de pesquisa binária balanceada, onde a diferença da altura das sub-árvores da esquerda e direita não pode exceder uma unidade, esse cálculo recebe o nome de Fator de Balanceamento. O objetivo dessa estrutura é, por meio da distribuição equilibrada de nós, otimizar a quantidade de operações de consultas, afim de expandir a capacidade e eficiência do programa.

Para montar uma AVL válida, é necessário a implementação de métodos de balanceamento que sejam inseridos nos processamentos padrões de uma árvore de pesquisa, como a inserção e remoção. Esses métodos utilizam na lógica de rotação de árvores que ocorre quando o FB de um nodo não está de acordo com as normas, logo, é superior a 1 ou inferior a -1.

Existem quatro possíveis rotações dentro de uma AVL:

Rotações Simples:

- Esquerda (FB > +1)
- Direita (FB < -1)

Rotações Duplas:

- Esquerda (Direita-Esquerda)
- Direita (Esquerda-Direita)

O processo ocorre da seguinte forma: para toda adição ou remoção de um nodo da árvore, todos os fatores de balanceamento são atualizados de baixo para cima (Nodos Externos -> Nodos internos), e a partir desses valores se da a verificação dos possíveis desbalanceamentos na estrutura. As rotações são feitas no nodo desbalanceado e dependem dos sinais dos fatores de balanceamento para escolher qual rotação é mais apropriada. Rotações simples ocorrem quando os nodos-filhos do nó desbalanceado compartilham o mesmo sinal de FB que seu nodo-pai, já Rotações duplas ocorrem quando o inverso acontece e os sinais de FB são opostos um do outro.

ROTAÇÕES E BALANCEAMENTO

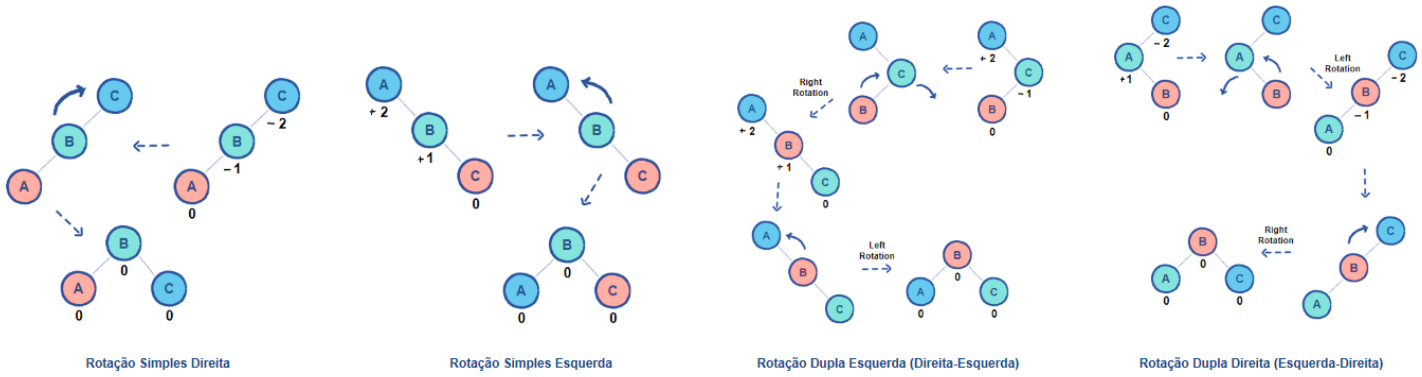

Figura 1 // https://www.educative.io/answers/common-avl-rotation-techniques [ADAPTADA]

Como dito anteriormente, as rotações dependem exclusivamente dos fatores de balanceamento dos nodos e os sinais atrelados a eles. A nossa implementação manuseia essa ferramenta da seguinte forma:

```
void ArvoreBinariadeBusca::add(int e){
    root = add(root, e, nullptr);
    count++;
    if(!cop)
        AVL(root);}

Nodo *ArvoreBinariadeBusca::add(Nodo *n, int e, Nodo *f){
    if(n == nullptr){
        Nodo *aux = new Nodo(e);
        aux->father = f;
        return aux;}
    if(e > n->element){
        n->right = add(n->right, e, n);}
    else{
        n->left = add(n->left, e, n);}
    return n;}
```

Ao adicionar um elemento na árvore todo novo nodo é sujeito a chamada do método “AVL”. O método trabalha em partes, nas quais invoca recursivamente a própria função para as sub-árvores de ambos os lados, define o Fator de Balanceamento do nodo apontado, verifica a necessidade de rotações simples e então rotações duplas, até que todos os nodos sejam verificados.

```
void ArvoreBinariadeBusca::AVL(Nodo *n){
    if(n == nullptr)
        return;
    AVL(n->left);
    AVL(n->right);
    int FB = height(n->right) - height(n->left);
```

Antecedendo as verificações, a função se chama recursivamente para percorrer as subárvores e atingir as folhas (Nodos externos). Isso é essencial para evitar erros na atualização do fator de balanceamento.

Ademais, existem quatro condicionais usadas para definir as rotações na árvore. O código inicia com a verificação do fator de balanceamento (FB) do nó atual (n). Se o FB for maior ou igual a 2, indica a necessidade de uma rotação simples para a esquerda. Dentro dessa condição, há uma segunda verificação. Se o filho direito (n->right) existe e o fator de balanceamento desse filho for menor ou igual a -1, isso indica uma situação que exige uma rotação dupla.

```
//1ª Verificação (Rotação Simples Esquerda)
if(FB >= 2){
Nodo *aux;
    //2ª Verificação (Rotação Dupla Esquerda)
    if(n->right != nullptr)
        if(height(n->right->right) - height(n->right->left) <= -1){

            Nodo *aux150 = n->right->left;  //Rotação S.D
            Nodo *aux200 = n->right;
            aux150->father = n;
            aux150->right = aux200;
            aux200->father = aux150;
            aux200->left = nullptr;
            n->right = aux150;}

    Nodo *aux150 = n->right; //Rotação S.E
    aux = n->right->left;
    aux150->father = n->father;
    aux150->left = n;
    n->father = aux150;
    n->right = aux;
    if(n == root)
        root = aux150;
    else aux150->father->right = aux150;}
```

Caso a segunda verificação seja falsa, ou seja, não é necessária uma rotação dupla, então a primeira verificação fica livre para executar uma rotação simples para a esquerda. É feita também modificações nos ponteiros para que os relacionamentos entre os nodos não sejam prejudicados durante a rotação.

/ A segunda parte do método AVL acompanha o raciocinio das duas primeiras verificações, porém com rotações opostas para cobrir os casos restantes (Rotação simples direita / Rotação dupla direita). */*

TESTAGEM:

Para nosso cenário de teste, instanciamos uma árvore (chamada árvore) e adicionamos os elementos na seguinte ordem: [10, 12, 11, 13, 14, 8, 9, 7, 6].

A primeira rotação que ocorre é após a inserção do elemento [11], causando uma **rotação dupla esquerda** no nodo [10]:

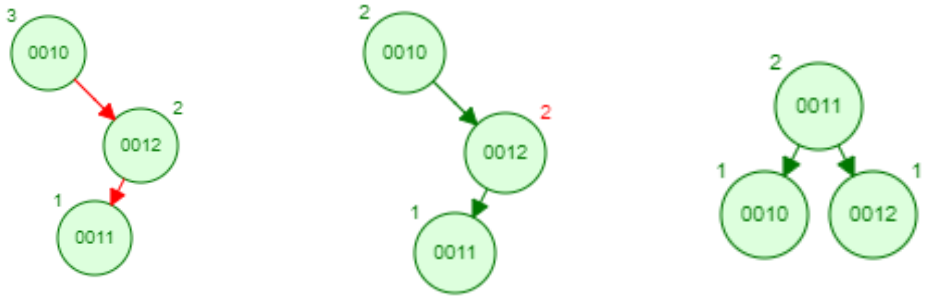

Figura 2 // https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

A segunda rotação que ocorre é após a inserção do elemento [14], causando uma **rotação simples esquerda** no nodo [12]:

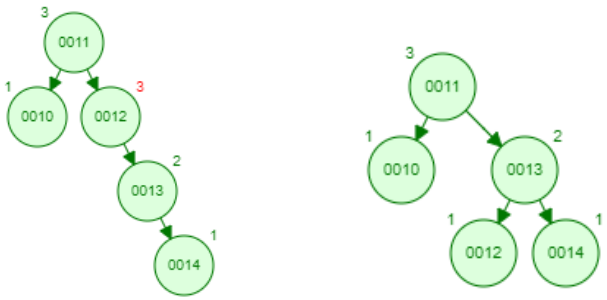

Figura 3 // https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

A terceira rotação que ocorre é após a inserção do elemento [9], causando uma **rotação dupla direita** no nodo [10]:

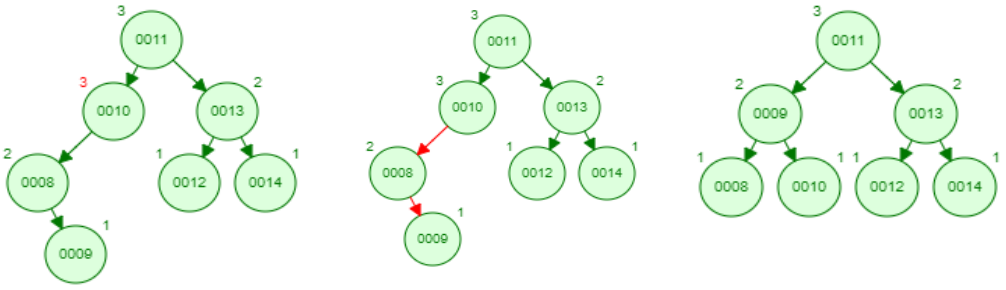

Figura 4 // https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

A quarta – e última - rotação que ocorre é após a inserção do elemento [6], causando uma **rotação simples direita** no nodo [8]:

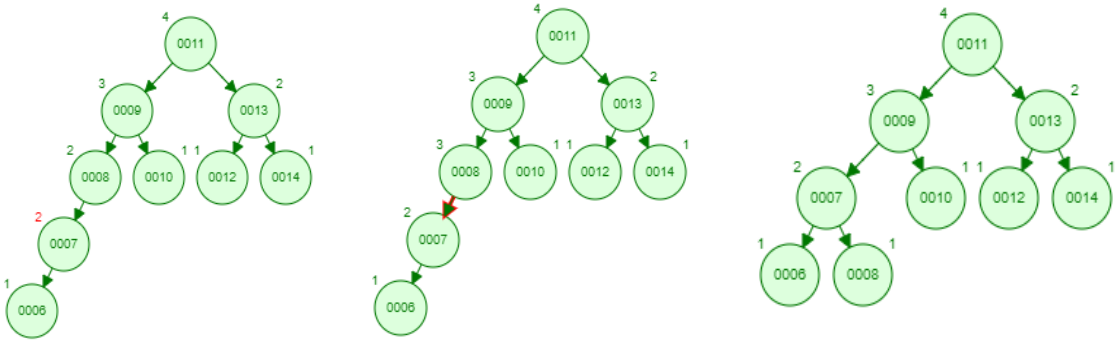

Figura 5 // https://www.cs.usfca.edu/~galles/visualization/AVLtree.html