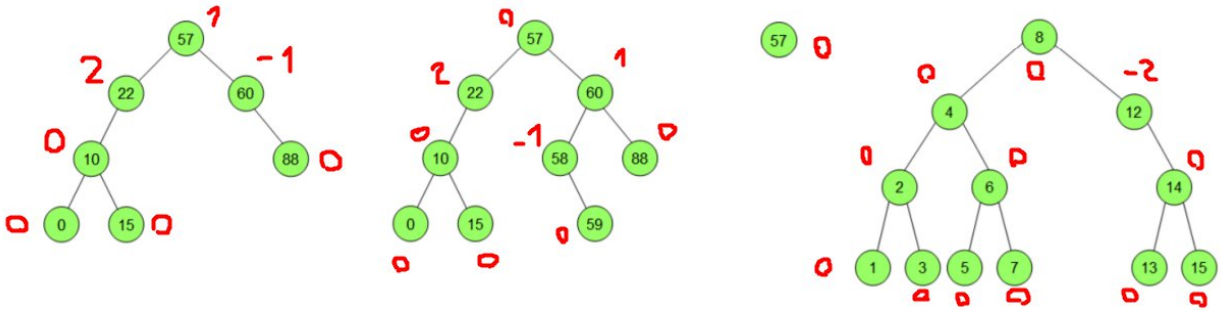
Lista de Árvores binárias

Ian Batista Fornaziero

Gabriel Augusto Dupim

1 – A primeira árvore não respeita o balanceamento pois tem um nó (22) que tem valor 2, o que não respeita a regra de (1,0,-1) do balanceamento, o mesmo ocorre na segunda árvore e inclusive com o mesmo nó, já a terceira é uma árvore AVL tendo em vista que ela está com valor 0 de balanceamento, e a última não é pois o nó 12 tem valor 2.



2 - A)

50 adicionado como raiz —> 30 adicionado a esq de 50 —> 20 adicionado a esq de 30 —> fazer balanceamento LL —> torna 30 raiz, 20 a esq de 30 e 50 a dir de 30 —> adiciona 70 a dir de 50, 40 a esq de 50 e 35 a esq de 40 —> faz balanceamento RL em 50,40,35 —> tornando 40 raiz de tudo, 30 a esq de 40, 35 a dir de 30, 50 a dir de 40 e 70 a dir de 50 —> adiciona 37 a dir de 35 e 38 a dir de 37 —> faz balanceamento RR entre 35,37,38 —> torna 37 raiz da sub árvore, 38 a dir de 37 e 35 a esq de 37, adiciona 10 a esq de 20 e 32 a esq de 35 —> faz balanceamento LR na sub árvore de 32 —> torna 37 raiz de tudo, 30 a esq de 37, 35 a dir de 30, 32 a esq de 35, 40 a dir de 37 e 38 a esq de 40 —> Adiciona 45 a esq de 50 e 42 a esq de 45 —> faz balanceamento RL na sub árvore 40 —> torna 45 raiz da sub árvore, 40 a esq de 45 e 42 a dir de 40 —> Adiciona 25 a dir de 20, 47 a esq de 50 e 36 a dir de 35 —> inserção finalizada

B)

Adiciona 100 como raiz, 80 a esq de 100 e 60 a esq de 80 —> fazer balanceamento LL na árvore —> torna 80 raiz da árvore, 100 a dir de 80 e 60 a esq de 80 —> adiciona 40 a esq de 60 e 20 a esq de 40 —> fazer balanceamento LL na sub árvore de 60 —> torna 40 raiz da sub árvore, 60 a dir de 40 e 20 a esq de 40 —> Adiciona 70 a dir de 60 —> faz balanceamento LR na árvore —> torna 60 raiz da árvore inteira, 80 a dir de 60 e 70 a esq de 80 —> Adiciona 30 a dir de 20 —> faz balanceamento LR na sub árvore de 40 —> torna 30 raiz da sub árvore, 20 a esq de 30 e 40 a dir de 30 —> adiciona 50 a dir de 40, 35 a esq de 40 e 45 a esq de 50 —> faz balanceamento na sub árvore 30 —> torna 40 a raiz da sub árvore, 30 a esq de 40 e 35 a dir de 30 —> adiciona 55 a dir de 50, 75 a dir de 70, 65 a esq de 70 e 73 a esq de 75 —> fazer balanceamento RL na sub árvore 80 —> torna 75 raiz da sub árvore, 80 a dir de 75, 70 a esq de 75 e 73 a dir de 70 —> adiciona 77 a esq de 80 —> Inserção finalizada

C)

Adiciona 41 como raiz, 38 a esq de 41, 31 a esq de 38 —> faz balanceamento LL na árvore —> tornando 38 raiz e 41 a dir de 38 —> Adiciona 12 a esq de 31 e 19 a dir de 12 —> faz balanceamento LR na sub árvore de 31 —> tornando 19 raiz da sub árvore, 12 a esq de 19 e 31 a dir de 19 —> Adiciona 8 a esq de 12 —> faz balanceamento LL na árvore — tornando 19 raiz da árvore inteira e 38 a dir de 19 —> adiciona 27 a esq de 31 e 49 a dir de 41 —> fim da inserção

D)

Adiciona 10 como raiz, 21 a dir de 10 e 15 a esq de 21 —> faz balanceamento RL na árvore —> tornando 15 raiz, 21 a dir de 15 e 10 a esq de 15 —> adiciona 17 a esq de 21 e 16 a esq de 17 —> faz balanceamento LL na sub árvore de 21 —> tornando 17 raiz da sub árvore e 21 a dir de 17 —> Adiciona 19 a esq de 21 —> faz balanceamento LL na árvore —> tornando 17 raiz da árvore, 15 a esq de 17 e 16 a dir de 15 —> Adiciona 20 a dir de 19 —> faz balanceamento LR na sub árvore 21 —> tornando 20 raiz da sub árvore, 19 a esq de 20 e 21 a dir de 20 —> fim da inserção

3 – Utilizando um exemplo simples, se a ordem não importa, se eu adicionar a chave {10,20,15,12,11} em ordem e depois colocar ela em ordem inversa, daria a mesma árvore, porém, não é o que acontece, inserindo em ordem, temos uma árvore de altura 2 a esquerda e 1 a direita, enquanto inserindo ao inverso, forma uma árvore igual dos dois lados.

5 – A função Insere_ArvAVL funciona da seguinte mandeira:

Caso a árvore esteja vazia, ela cria um novo Nó e o aloca, logo em seguida verifica se foi alocado corretamente, caso não, ele retorna da função para evitar falhas de acesso NULL.

```
novo->info = valor;
novo->altura = 0;
novo->esq = NULL;
novo->dir = NULL;
*raiz = novo;
return 1;
```

Nesse trecho ele pega o valor informado na chamada da função, define a altura como 0, afinal, estamos criando a raiz, e do mesmo jeito, a esquerda e direita estarão vazias, por isso elas são iniciadas como NULL, então retorna 1 informando sucesso.

Caso não seja a raiz a ser inserida, é definido um novo ponteiro “atual” que vai apontar para a raiz informada.

Em seguida é feito uma verificação para caso o valor seja menor que o valor de atual.

Caso sim, o nó deve ser movido para a esquerda, para isso ele faz outra verificação, que caso tenha feito a inserção do novo nó, ele vai verificar o fator de balanceamento, se estiver maior ou igual a dois, quer dizer desbalanceado, então nesse caso ele precisa ser balanceado.

Para tal é feito uma verificação se o valor informado é menor que o valor da esquerda da raiz, se for, ele chama a função de rotação LL, caso contrário, ele chama a função de rotação LR.

Caso o valor informado seja maior que o valor de atual, ele vai fazer a mesma coisa que antes, só que agora pensando na direita do atual.

E por fim impede que o valor seja igual a outro, e retorna 0 como falha na inserção.

Depois ele corrige a altura do atual.

6 – A função começa verificando se o valor chamado na função existe dentro da árvore, caso não existe, vai retornar 0 indicando falha.

Depois a função vai ter ifs para se mover pela árvore até o valor da chamada, caso seja menor que o valor da que está na raiz vai para o nó da esquerda chamando a função recursivamente, logo depois se verifica se a raiz está desbalanceada, caso esteja, é chamado outra verificação, caso a altura do lado esquerdo for igual ou menor, é chamada a rotação RR que vai movimentar os ponteiros para balancear a árvore, caso contrário, vai chamar a função rotacao RL que vai realizar uma rotação LL e depois uma RR para então balancear corretamente.

Abaixo vai fazer a mesma verificação mas de forma espelhada, pensando no lado direito da árvore.

```
if(valor < (*raiz)->info){
    if((res = remove_ArvAVL(&(*raiz)->esq,valor)) == 1){
        if(fatorBalanceamento_NO(*raiz) >= 2){
            if(altura_NO((*raiz)->dir->esq) <= altura_NO((*raiz)->dir->dir))
                RotacaoRR(raiz);
            else
                RotacaoRL(raiz);
        }
    }
}

if((*raiz)->info < valor){
    if((res = remove_ArvAVL(&(*raiz)->dir, valor)) == 1){
        if(fatorBalanceamento_NO(*raiz) >= 2){
            if(altura_NO((*raiz)->esq->dir) <= altura_NO((*raiz)->esq->esq) )
                RotacaoLL(raiz);
            else
                RotacaoLR(raiz);          Espelhado
        }
    }
}
```

Depois dessas duas verificações, vai ser feito novos ifs para saber se o nó tem 1 ou nenhum filho, para esses casos, é declarado uma variavel oldNode, que vai apontar para a raiz, logo depois, caso um filhos seja NULL, então é igualado a raiz ao nó que não é NULL e então é liberado a raiz antigo com free(oldNode).

Caso o nó tenha dois filhos, o valor da raiz passa a ser o menor a direita, então vai ser chamado de forma recursiva a função, com o novo valor da raiz.

Novamente é feito verificação de balanceamento e o devido balanceamento caso precise.

E por fim a altura da raiz é atualizada devidamente.