5) Descreva passo a passo o funcionamento da função insere_ArvAVL(ArvAVL *raiz, int valor), descreva também o uso de qualquer outras funções dentro dessa. Utilize trechos do código para exemplificar.

Rotações: (A, B, C)

RR:Ocorre quando um nó é inserido na subárvore direita do filho direito de um nó desbalanceado. Para corrigir , simples rotação a esquerda.

LL:Ocorre quando um nó é inserido na subárvore esquerda do filho esquerdo de um nó desbalanceado. Para corrigir , simples rotação a direita.

RL:Ocorre quando um nó é inserido na subárvore esquerda do filho direito de um nó desbalanceado. Para corrigir ,rotação à direita em B e rotação à esquerda em A

LR:Ocorre quando um nó é inserido na subárvore direita do filho esquerdo de um nó desbalanceado. Para corrigir , rota à esquerda em A, rotação à direita em C.

```
  int insere_ArvAVL(ArvAVL *raiz, int valor){

int res; //Variável res inteira temporária
   if(*raiz == NULL){//�rvore vazia ou n� folha
       struct NO *novo;
       novo = (struct NO*)malloc(sizeof(struct NO));
       if(novo == NULL)
           return 0;

       novo->info = valor;
       novo->altura = 0;
       novo->esq = NULL;
```

```
        novo->dir = NULL;
        *raiz = novo;
        return 1;
    }
/* Se a raiz existir, mas for nula, quer dizer que se
está inserindo
   o primeiro elemento. Então cria-se o nó, malloca
ele, faz o tratamento
   de erro e estabelece info, altura e para onde os
ponteiros apontam. */

    struct NO *atual = *raiz;
    if(valor < atual->info){
        if((res = insere_ArvAVL(&(atual->esq), valor))
== 1){
            if(fatorBalanceamento_NO(atual) >= 2){
                if(valor < (*raiz)->esq->info ){
                    RotacaoLL(raiz);
                }else{
                    RotacaoLR(raiz);
                }
            }
        }
    }else{
        if(valor > atual->info){
            if((res = insere_ArvAVL(&(atual->dir),
valor)) == 1){
                if(fatorBalanceamento_NO(atual) >= 2){
                    if((*raiz)->dir->info < valor){
                        RotacaoRR(raiz);
                    }else{
                        RotacaoRL(raiz);
                    }
                }
```

```
            }
        }else{
            printf("Valor duplicado!!\n");
            return 0;
        }
    }

    atual->altura =
maior(altura_NO(atual->esq),altura_NO(atual->dir)) + 1;

    return res;
}
/* Cria nó auxiliar igual a raiz. Série de ifs:
Verifica se o valor é menor que
   o atual info, se for menor, verifica se o valor de
res que
   recebe o resultado da propria função ( recursiva)
passando pro atual esquerda e valor
   é igual a 1. Caso sim, verifica se o fator de
balanceamentoatual é maior igual a 2 , caso
   sim ,ele verifica se valor é menor que raizesq, se
sim rotaçãoLL, senão rotação LR. Se for maior, verifica
se o valor de res que
   recebe o resultado da propria função ( recursiva)
passando pro atual direita e  valor
   é igual a 1. Caso sim, verifica se o fator de
balanceamentoatual é maior igual a 2 , caso
   sim ,verifica se valor é maior que raizdir, se sim
rotação RR, senão RL. Se valor não for
   nem menor e nem maior que atual, quer dizer que está
duplicado. No final a altura é recorrigida a partir da
verificação em cada
   etapa das recursões, recalculando a altura. */
```