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

```
   struct NO* procuraMenor(struct NO* atual){

 struct NO *no1 = atual;    //Guarda o ultimo nó visitado( atual  )
    struct NO *no2 = atual->esq; // começa do filho esquerdo
    while(no2 != NULL){    // Enquanto houver um nó mais a esquerda, vai descendo pra esquerda, no1 acompanha o ultimo nó visitado
        no1 = no2;         // e no2 tenta ir mais pra esquerda, quando no2 está nulo, significa que chegamos ao nó mais a esquerda
        no2 = no2->esq;
    }
    return no1; // como no1 está nesse nó , retorna no1
}
```

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

 if(*raiz == NULL){// valor n◆o existe
        printf("valor n◆o existe!!\n");
        return 0;
    }
/* Verifica se o Valor existe pra pensar em remover*/

    int res;//Variável res inteira temporária
    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);
            }
        }
    }
       // Verifica se valor a remover está na subarvore
esquerda, se sim verifica se o nó
       // atual está desbalanceado para a direita, se
sim arruma com as rotações.
    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);
            }
        }
    }
// Verifica se valor a remover está na subarvore
direita, se sim verifica se o nó
       // atual está desbalanceado para a esquerda, se
sim arruma com as rotações.
    if((*raiz)->info == valor){
        if(((*raiz)->esq == NULL || (*raiz)->dir ==
NULL)){// n tem 1 filho ou nenhum
```

```
            struct NO *oldNode = (*raiz);
            if((*raiz)->esq != NULL)
                *raiz = (*raiz)->esq;
            else
                *raiz = (*raiz)->dir;
            free(oldNode);
        }else { // n tem 2 filhos
            struct NO* temp =
procuraMenor((*raiz)->dir);
            (*raiz)->info = temp->info;
            remove_ArvAVL(&(*raiz)->dir,
(*raiz)->info);
            if(fatorBalanceamento_NO(*raiz) >= 2){
                if(altura_NO((*raiz)->esq->dir) <=
altura_NO((*raiz)->esq->esq))
                    RotacaoLL(raiz);
                else
                    RotacaoLR(raiz);
            }
        }
        if (*raiz != NULL)
            (*raiz)->altura =
maior(altura_NO((*raiz)->esq),altura_NO((*raiz)->dir))
+ 1;
        return 1;
    }
       /* Verifica se encontrou o valor a ser removido,
se sim, verifica se ele não
       tem algum filho na esquerda ou na direita , se n
tem, basta dar o free nele,se
       tem chama o fator de balanceamento pra cada caso
e faz as rotações apropriadas.*/
```

```
    (*raiz)->altura =
maior(altura_NO((*raiz)->esq),altura_NO((*raiz)->dir))
+ 1;
/* No final a altura é recorrigida a partir da
verificação em cada
   etapa das recursões, recalculando a altura.*/
    return res;
}
```