Árvore Binária de Busca

Com base no que foi discutido codifique uma função, na linguagem C, que implemente a remoção por fusão.

```c
void remocaoPorFusao(ARV_BIN_BUSCA *arvore) {
   if (*arvore) {
      ARV_BIN_BUSCA tmp = *arvore;
      if (!((*arvore)->right)){
         if ((*arvore)->left) (*arvore)->left->father = (*arvore)->father;
         *arvore = (*arvore)->left;
      } else
         if ((*arvore)->left == NULL){
            (*arvore)->right->father = (*arvore)->father;
            *arvore = (*arvore)->right;
         } else {
            tmp = (*arvore)->left;
            while (tmp->right)
               tmp = tmp->right;
            tmp->right = (*arvore)->right;
            tmp->right->father= tmp;
            tmp = *arvore;
            *arvore = (*arvore)->left;
         }
      free (tmp);
   }
}
```

Árvore Binária de Busca

Outra solução é a Remoção por cópia.

Proposta por Thomas Hibbard e Donald Knuth, propõe que um nó com dois filhos a ser removido pode ser reduzido a uma das duas situações básicas: nó com apenas um filho e nó sem nenhum filho.

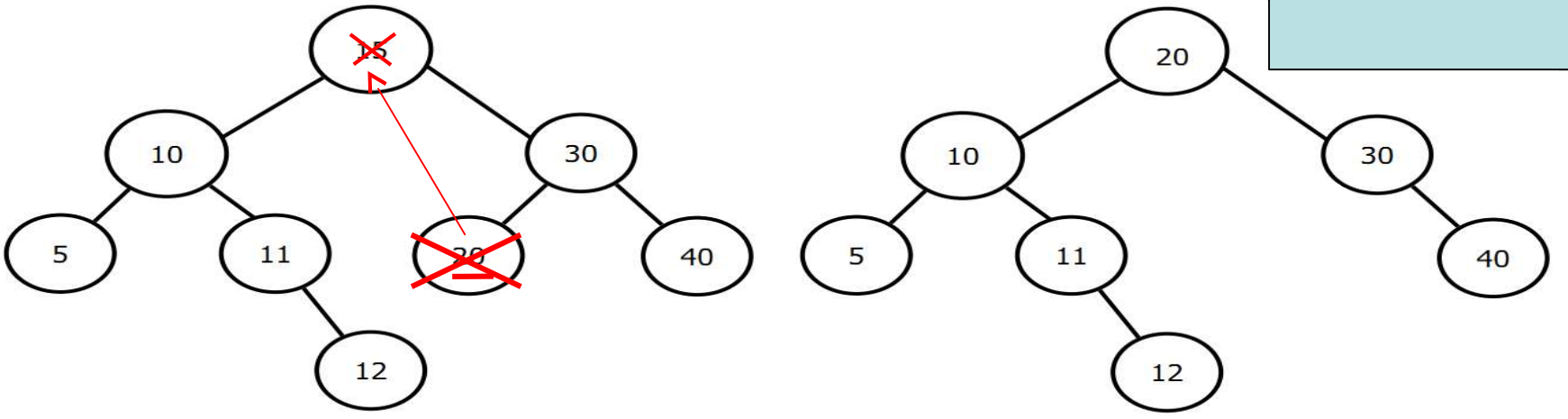
Isso é feito substituindo pela chave de seu sucessor imediato a chave que está sendo removida e em seguida removendo o nó que continha a chave do sucessor imediato.

Obs.: o sucessor imediato de um nó é o nó mais à esquerda em sua subárvore à direita.

UNIVASF

Árvore Binária de Busca

Vejamos um exemplo:



Exercício:

Com base no que foi discutido codifique uma função, na linguagem C, que implemente a remoção por cópia.

```c
void remocaoPorCopia(ARV_BIN_BUSCA *arvore) {
   if (*arvore) {
      ARV_BIN_BUSCA tmp = *arvore;
      if (!((*arvore)->right)){
         if ((*arvore)->left) (*arvore)->left->father = (*arvore)->fath
         *arvore = (*arvore)->left;
      } else
         if ((*arvore)->left == NULL){
            (*arvore)->right->father = (*arvore)->father;
            *arvore = (*arvore)->right;
         } else {
            tmp = (*arvore)->right;
            while (tmp->left!=NULL)
               tmp = tmp->left;
            (*arvore)->info = tmp->info;
            if (tmp->father==*arvore){
               tmp->father->right = tmp->right;
               if (tmp->father->right) tmp->father->right->father = tmp->father;
            } else {
               tmp->father->left = tmp->right;
               if (tmp->father->left) tmp->father->left->father = tmp->father;
            }
         }
      free (tmp);
   }
}
```

Árvore Balanceada
Árvore AVL

UNIVASF

Árvore Balanceada

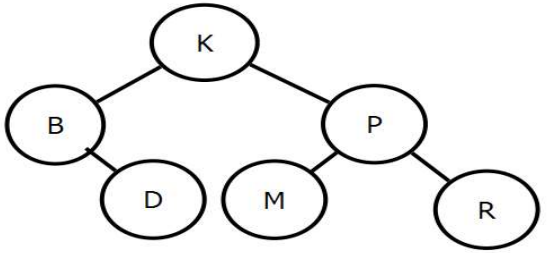
Além das árvores serem muito apropriadas para representar a estrutura hierárquica de um certo domínio, o processo de busca por um elemento em uma árvore tende a ser muito mais rápido do que em uma lista encadeada.

Contudo, para que efetivamente uma busca por um determinado elemento em uma árvore seja eficiente esta, além de ser uma árvore binária de busca, deve ter seus nós adequadamente distribuídos.

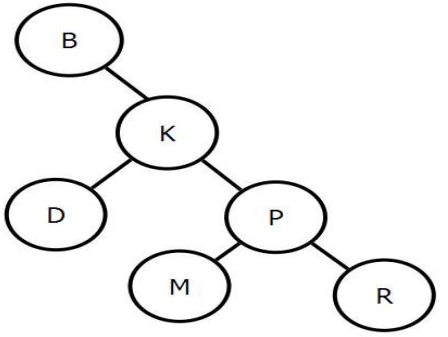
UNIVASF

Árvore Balanceada

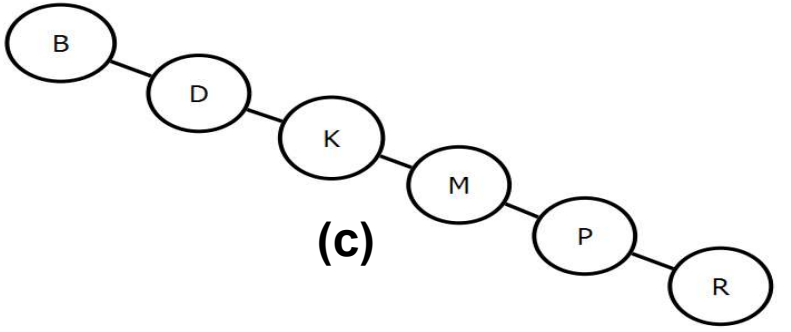
Observe as árvores a seguir:



(a)



(b)



(c)

Sendo assim, podemos estabelecer a seguinte definição: Uma árvore binária é ***balanceada em altura*** ou simplesmente ***balanceada*** se a diferença na altura de ambas as subárvores de qualquer nó na árvore é zero ou um.

Árvore Balanceada

Uma definição complementar é a de árvore ***perfeitamente balanceada***.

Uma árvore binária é ***perfeitamente balanceada*** quando, além de ser balanceada, todas as suas folhas encontram-se em um ou dois níveis.

Para facilitar a visualização, uma árvore que possui 10.000 nós pode ser configurada em uma árvore com altura igual a $log_2(10000) = ceil(13.289) = 14$. Ou seja, qualquer elemento é passível de ser localizado com no máximo 14 comparações se a árvore for perfeitamente balanceada.

UNIVASF

Árvore Balanceada

Neste ponto cabe a seguinte pergunta:

Ao se analisar uma árvore pode-se determinar qual dentre os seus nós seria uma raiz adequada para torná-la perfeitamente balanceada?

Qual seria este nó?

O nó cuja sua chave (valor) representa a mediana das chaves presentes nos nós que compõem a árvore, para uma árvore com número impar de nós. Ou o nó cuja sua chave (valor) representa um dentre os dois valores mais próximos da mediana das chaves presentes nos nós que compõem a árvore, para uma árvore com número par de nós.

UNIVASF

Árvore Balanceada

Uma estratégia baseada nesta observação que pode ser utilizada para balancear uma árvore é:

- Retire os dados da árvore e armazene-os em um vetor;

- Após todos os dados terem sido armazenados no vetor, ordene-o;

- Agora determine como raiz o elemento do meio do vetor;

- O vetor consistirá agora em dois subvetores. O filho esquerdo da raiz será o nó com valor no meio do subvetor constituído do início do vetor até o elemento escolhido como raiz;

UNIVASF

Árvore Balanceada

- Um procedimento similar é adotado para a definição do filho direito da raiz;

- Este processo se repete até não existirem mais elementos a serem retirados do vetor.

Uma proposta de exercício de fixação consiste em implementar uma função, na linguagem C, que implementa este processo.

UNIVASF

Árvore Balanceada

O algoritmo mencionado possui um sério inconveniente, pois, caso a árvore já exista, os seus elementos devem ser retirados e colocado em um vetor, para que a mesma seja recriada.

Caso a árvore ainda não exista, o inconveniente ainda persiste. Pois, todos os dados precisam ser colocados em um vetor antes da árvore ser criada.

Uma pequena melhoria pode ser feita. Pois, mesmo que a árvore binária de busca esteja desbalanceada, se for efetuado um percurso in-ordem elimina-se a necessidade de ordenar o vetor.

Árvore Balanceada

Existem formas mais eficientes de se balancear uma árvore.

Um exemplo é o algoritmo denominado DSW (proposto por Colin Day e posteriormente melhorado por Quentin F. Stout e Bette L. Warren). O qual baseia-se em percorrer uma árvore binária de busca tornando-a uma arvore degenerada (similar a uma lista encadeada) e posteriormente percorrê-la novamente tornando-a uma árvore perfeitamente balanceada.

UNIVASF

Árvore AVL

Até o momento foram mencionados algoritmos que balanceiam árvores globalmente.

Contudo, o rebalanceamento pode ocorrer localmente se alguma porção da árvore for desbalanceada por uma operação de inserção ou remoção de um elemento da árvore.

Um método clássico, que nomeia uma árvore modificada pelo mesmo como AVL. (Em função de seus idealizadores, Adel'son-Vel'skii e Landis)

UNIVASF

Árvore AVL

Uma árvore AVL é uma árvore binária de busca onde a diferença em altura entre as subárvores esquerda e direita de cada nó é no máximo um (positivo ou negativo).

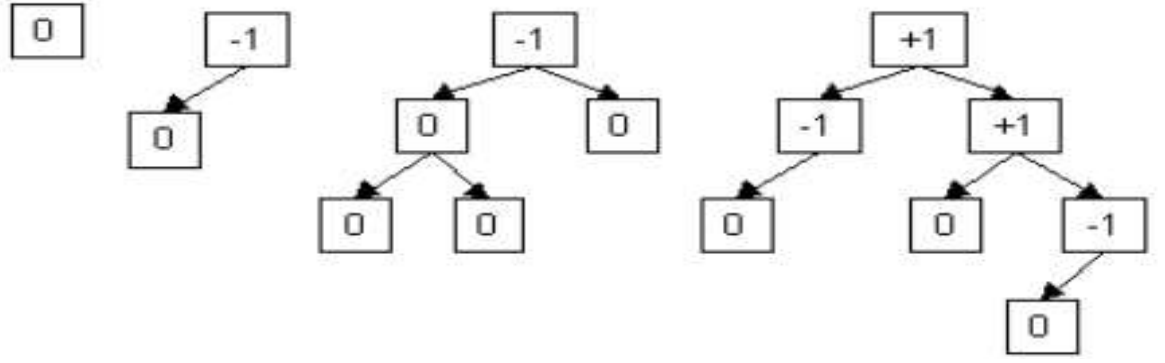
Esta diferença é chamada de fator de balanceamento (FB).

O FB (ou informações que permitam sua obtenção) é acrescentado a cada nó da árvore AVL.

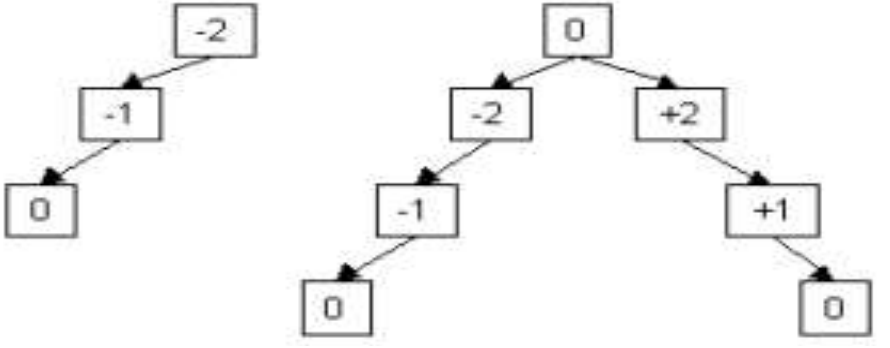
FB = altura subárvore – altura subárvore
nó *p* direita de *p* esquerda de *p*

Árvore AVL

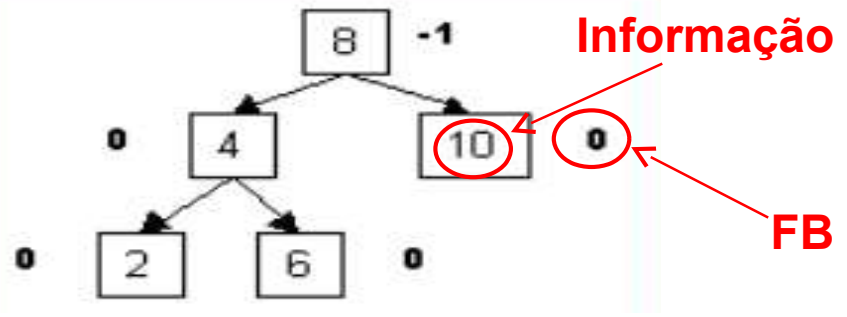
Árvores AVL:



Árvores não AVL:



Árvore AVL



Árvore AVL

Ao inserirmos um novo nó em uma árvore AVL, podemos ou não violar a propriedade de balanceamento.

Caso, ocorra uma violação devemos rebalancear a árvore através da execução de operações de rotação sobre nós da árvore.

Árvore AVL

Após uma inserção que gere um des-balanceamento na árvore AVL podemos nos deparar com duas classes de desbalanceamento.

Onde estas são identificadas com base na análise dos FB's.

Ao inserirmos um novo nó devemos ajustar os FB's, desde o nó inserido até a raiz ou até encontrarmos um fator de balanceamento inaceitável, ou seja, com valor 2 ou -2.

UNIVASF

Árvore AVL

Quando o FB do nó filho com valor 1 (+ ou -) possuir o mesmo sinal do FB de seu pai (nó com FB 2 ou -2) trata-se da classe de desbalanceamento 1, que requer apenas uma rotação simples para a árvore ser rebalanceada.

Quando o FB do nó filho com valor 1 (+ ou -) possuir sinal oposto ao FB de seu pai (nó com FB 2 ou -2) trata-se da classe de desbalanceamento 2 que requer uma rotação dupla, ou em outras palavras, duas rotações para a árvore ser rebalanceada.

UNIVASF

Árvore AVL

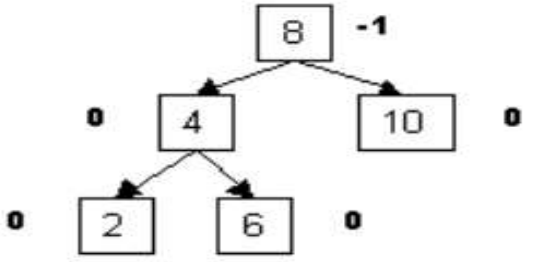
Se o sinal do FB do nó que caracteriza o desbalanceamento for positivo a rotação será para a esquerda.

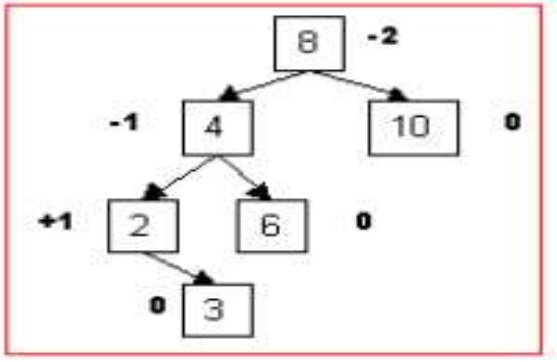
Se o sinal do FB do nó que caracteriza o desbalanceamento for negativo a rotação será para a direita.

Analisaremos agora um exemplo de cada uma das classes citadas.

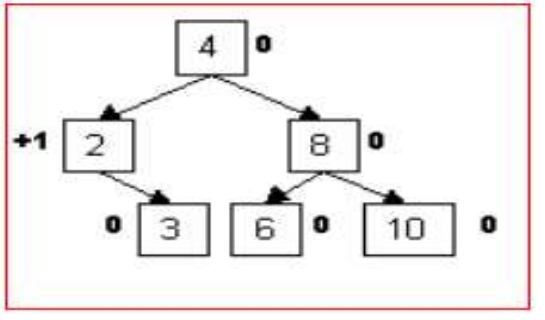
UNIVASF

Ao inserirmos o valor 3 na árvore abaixo teremos:

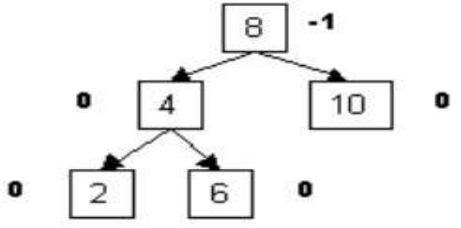


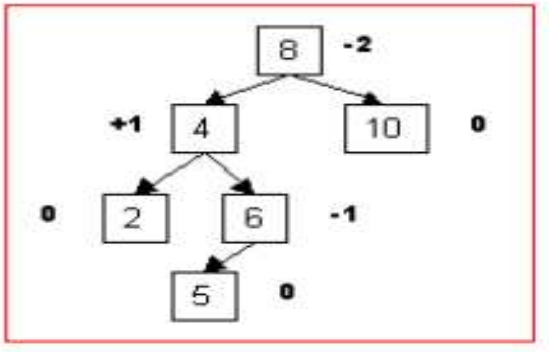


Identificamos a classe 1, que demanda uma rotação simples à direita. Após a rotação teremos:



Ao inserirmos o valor 5 na árvore abaixo teremos:



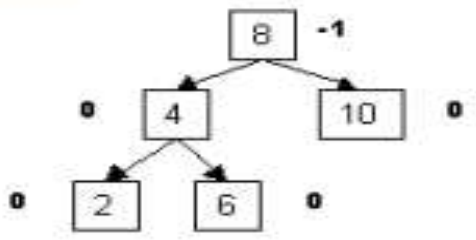


Identificamos a classe 2, que demanda uma rotação dupla à direita.

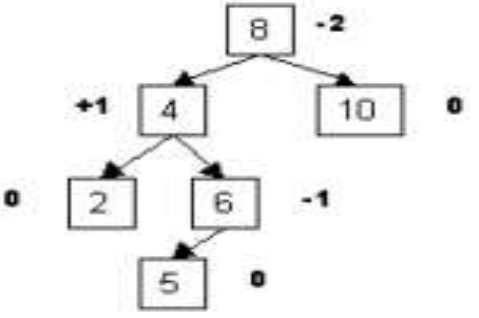
Iniciamos por uma rotação no nó filho com FB +1, no caso, com uma rotação à esquerda devido ao sinal +. Seguindo por uma rotação à direita no nó com FB -2.

O que gera a sequência de árvores a seguir.

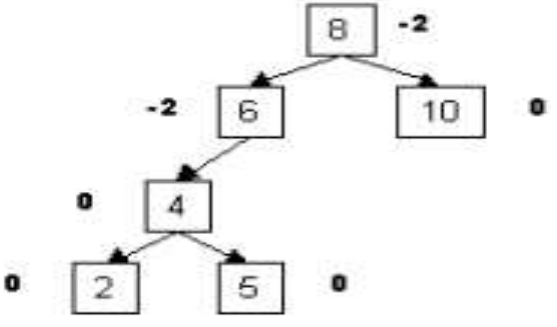
Árvore AVL



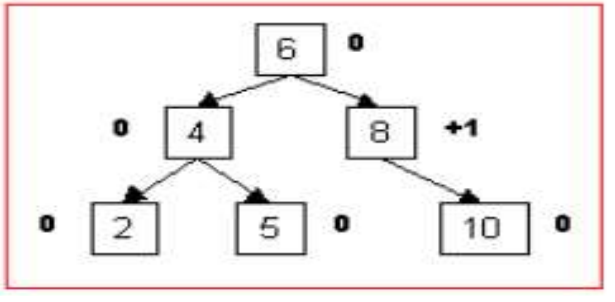
Árvore inicial



Árvore desbalanceada
(após inserção do valor 5)



Árvore em processo
de balanceamento
(após a 1ª rotação)



Árvore balanceada
(após a 2ª rotação)

Árvore AVL

Descreva uma sequência de passos para a construção de uma árvore AVL.

1. insira o novo nó normalmente, ou seja, da mesma maneira que insere-se um nó em uma árvore binária de busca;

2. iniciando com o nó pai do nó recém inserido, teste se a propriedade AVL foi violada, ou seja, atualize e teste se algum dos FB's passou a ser 2 ou -2. Temos duas possibilidades:

- **2.1. A condição AVL foi violada**
  - **2.1.1. Execute a operação de rotação conforme o caso (tipo 1 ou tipo 2);**
  - **2.1.2. Volte ao passo 1;**
- **2.2. A condição AVL não foi violada, volte ao passo 1;**

UNIVASF

Árvore AVL

Com base no que foi visto, proponha uma estrutura para um nó de uma árvore AVL implementada com alocação dinâmica. Considere que a informação armazenada em cada nó da árvore resumi-se a um valor inteiro.

```
typedef struct nodo
{
   int num, altd, alte;
   struct nodo *dir, *esq;
}NODO;
```

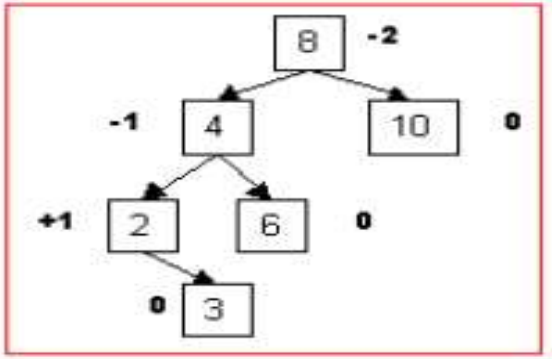
Como seria definido o tipo árvore AVL?

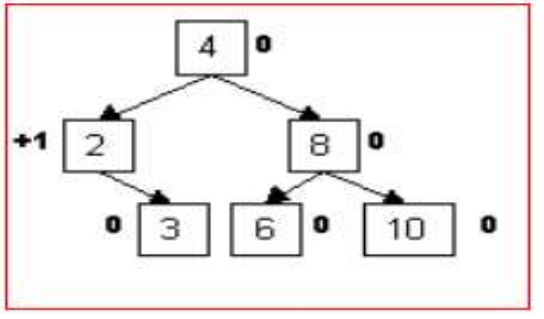
```
typedef NODO * ArvoreAVL;
```

Árvore AVL

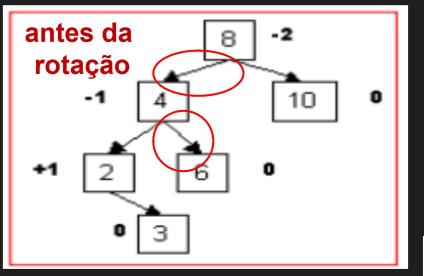
Com base no que foi visto, implemente a operação de rotação à direita sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

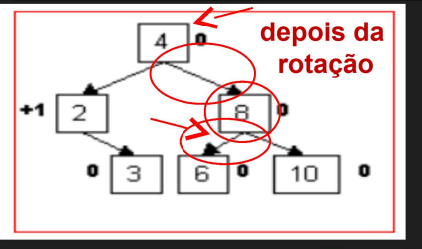
void rotacao_direita(ArvoreAVL *arvore);





```c
void rotacao_direita(ArvoreAVL *arvore)
{
    ArvoreAVL aux1, aux2;
    aux1 = (*arvore)->esq;
    aux2 = aux1->dir;
    (*arvore)->esq = aux2;
    aux1->dir = (*arvore);
    if ((*arvore)->esq == NULL)
        (*arvore)->alte = 0;
    else
        if ((*arvore)->esq->alte > (*arvore)->esq->altd)
            (*arvore)->alte = (*arvore)->esq->alte+1;
        else
            (*arvore)->alte = (*arvore)->esq->altd+1;
    if (aux1->dir->alte > aux1->dir->altd)
        aux1->altd = aux1->dir->alte + 1;
    else
        aux1->altd = aux1->dir->altd + 1;
    *arvore = aux1;
}
```

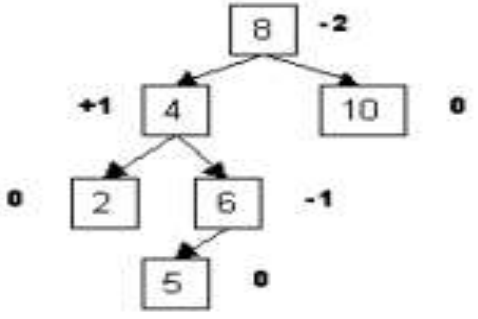


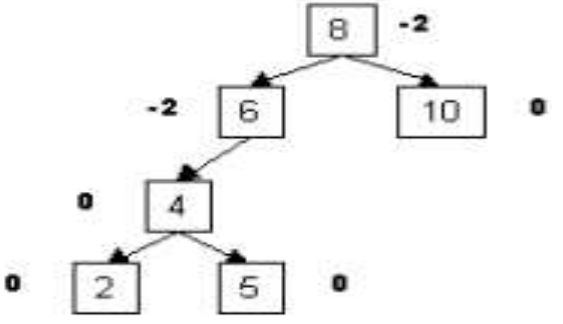


Árvore AVL

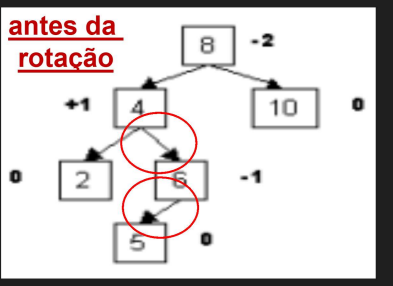
Com base no que foi visto, implemente a operação de rotação à esquerda sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

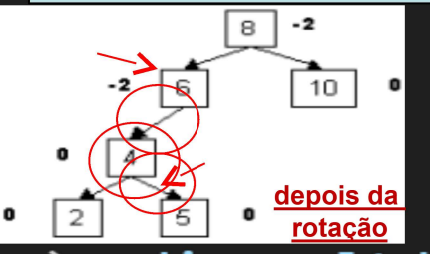
void rotacao_esquerda(ArvoreAVL *arvore);





```c
void rotacao_esquerda(ArvoreAVL *arvore)
{
    ArvoreAVL aux1, aux2;
    aux1 = (*arvore)->dir;
    aux2 = aux1->esq;
    (*arvore)->dir = aux2;
    aux1->esq = (*arvore);
    if ((*arvore)->dir == NULL)
        (*arvore)->altd = 0;
    else
        if ((*arvore)->dir->alte > (*arvore)->dir->altd)
            (*arvore)->altd = (*arvore)->dir->alte+1;
        else
            (*arvore)->altd = (*arvore)->dir->altd+1;
    if (aux1->esq->alte > aux1->esq->altd)
        aux1->alte = aux1->esq->alte + 1;
    else
        aux1->alte = aux1->esq->altd + 1;
    *arvore = aux1;
}
```





Árvore AVL

Com base nas operações de rotação à esquerda e à direita, implemente a operação de balanceamento sobre o nó recebido como parâmetro. Considere o protótipo abaixo para a função que implementará a operação em questão.

void balanceamento (ArvoreAVL *arvore);

UNIVASF