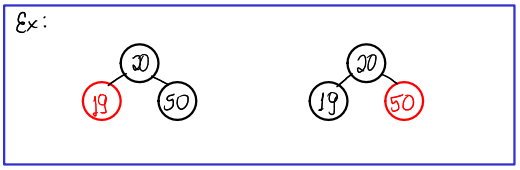
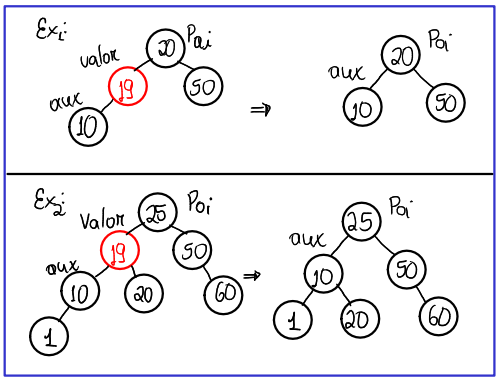
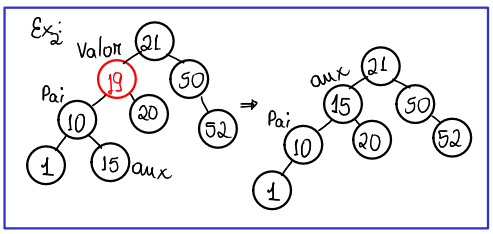
Algoritmo de remoçao em arvore AVL

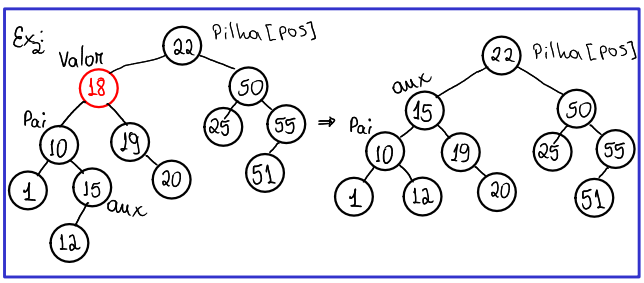
Maria Larissa da silva Andrade, 2021007179

```
Remove(raiz,valor):
  se busca(raiz, valor)== verdadeiro,então:
    aux = raiz
    i = 0
    Pilha[altura_arvore]

    Comentário: Preenche pilha
    Enquanto aux != valor, faça:
      pilha[i] = aux
      Se valor > aux.valor:
        aux = aux.direita
      Senão:
        aux = aux.esquerda
      i++

    Comentário: ultima posição que tem elemento na pilha
    pos = altura_arvore - qtd_pilha - 1

    Comentário: caso não tenha filhos
    Se aux.esquerdo == NULL e aux.direito == NULL, então:
      free(aux)
      aux = Pilha[pos]
      Se Pilha[pos].esquerda.valor == valor, então:
        Pilha[pos].esquerda == NULL
      Senão, então:
        Pilha[pos].direita == NULL

    Comentário: guarda referência do valor a ser removido
    aux_valor = aux

    Comentário: Verifica se possui filho a esquerda
    Se aux.esquerdo != NULL, então:
      pai = Pilha[pos]
      aux = aux.esquerda

      Comentário:Verifica se o filho direito do nó esquerdo é null
      Se aux.direita == NULL, faça:
        pai.esquerda = aux
        free(aux_valor)
        atualiza_balanço(aux)
        Pilha[pos+1] = aux
        pos++
      Comentário: Caso não seja NULL percorre até o ultimo nó mais a direita
      Senão, então:

        Enquanto aux.direita != NULL, faça:
          pai = aux
          aux = aux.direita

        Comentário: Se a esquerda do nó mais a direita for nulo
        Se aux.esquerda == NULL, então:
          aux.esquerda = aux_valor.esquerda
          aux.direita = aux_valor.direita
          atualiza_balanço(aux)
          pai.direita = NULL

        Comentário: Se a esquerda do nó mais direita  não for nulo
        Senão, então:
          pai.direita = aux.esquerda
          aux.esquerda = aux_valor.esquerda
          aux.direita = aux_valor.direita
          atualiza_balanço(aux)

        Comentário: Atualiza ponteiros do pai do nó removido
        Se Pilha[pos].direita.valor == valor, então:
          Pilha[pos].direita = aux
        Senão, então:
          Pilha[pos].esquerda = aux
        Pilha[pos+1] = aux
        pos++
        free(aux_valor)
```

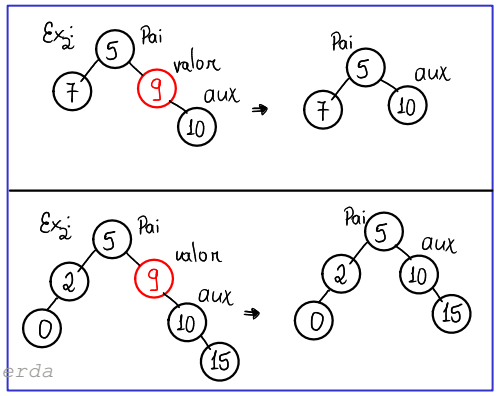


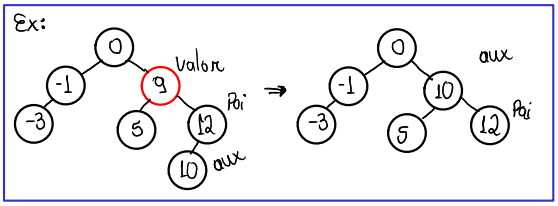


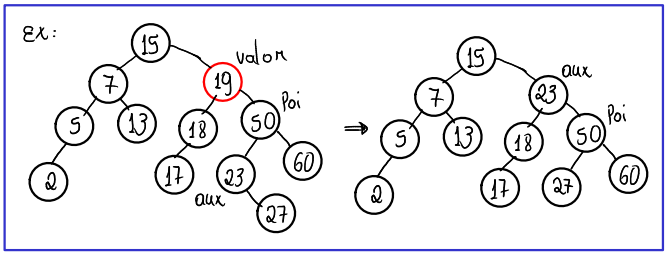


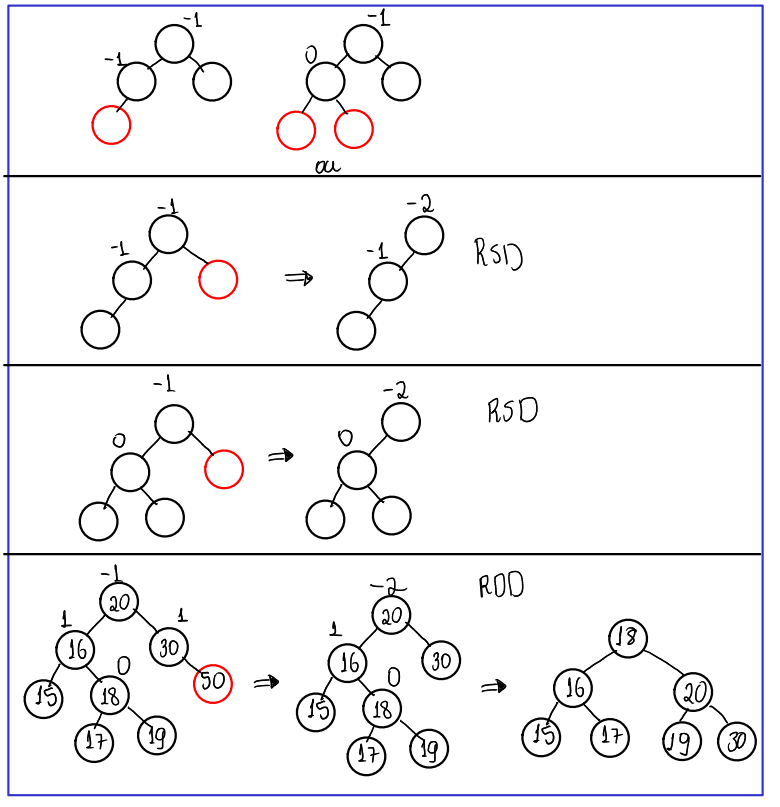


```
Comentário: Não tenho o filho esquerdo e testa se existe o filho direito
Senão, se aux.direita != NULL, então:
    pai = Pilha[pos]
    aux = aux.direita

    Comentário:Verifica se o filho esquerdo do nó direito é null
    Se aux.esquerda == NULL, então:
        pai.direita = aux
        free(aux_valor)
        atualiza_balanço(aux)
        Pilha[pos+1] = aux
        pos++


    Comentário: Caso não seja NULL percorre até o ultimo mais a esquerda
    Senão, então:
        Enquanto aux.esquerda != NULL, faça:
            pai = aux
            aux = aux.esquerda

        Comentário: Se a direita do nó mais a esquerda for nulo
        Se aux.direita == NULL, então:
            aux.direita = aux_valor.direita
            aux.esquerda = aux_valor.esquerda
            atualiza_balanço(aux)
            pai.esquerda = NULL

        Comentário: Se a direita do nó mais esquerda  não for nulo
        Senão, então:
            pai.esquerda = aux.direita
            aux.esquerda = aux_valor.esquerda
            aux.direita = aux_valor.direita
            atualiza_balanço(aux)

        Se Pilha[pos].direita.valor == valor, então:
            Piha[pos].direita = aux
        Senão, então:
            Pilha[pos].esquerda = aux
        Pilha[pos+1] = aux
        pos++
        Free(aux_valor)


Comentário: Banlanceia todo os elementos desbalanciados da pilha
Para i = pos, enquanto i >= 0, i--:
    antes = Pilha[pos].balanço
    depois = atualiza_balanço(Pilha[pos])

    Se antes == -1 e depois == 0, então:
        Continue
    Senão, se antes == -1 e depois == -1, então:
        Continue

    Senão, se antes == -1 e depois == -2, então:
        Pai = Pilha[pos]

        Se pai.esquerda != NULL, então:
            Se pai.esquerda.balanço <= 0, então:
                RSD(pai)
                atualiza_balanço(pai)
                Continue
            Se pai.esquerdo.balanço > 0, então:
                RSE(pai.esquerda)
                RSD(pai)
                atualiza_balanço(pai.esquerda)
                atualiza_balanço(pai)
                Continue
```









```
    Se antes == 0 e depois == 0, então:
    | Continue
    Senão, se antes == 0 e depois == -1, então:
    | Continue
    Senão, se antes == 0 e depois == 1, então:
    | continue


    Se antes == 1 e depois == 0, então:
    | Continue
    Senão, se antes == 1 e depois == 1, então:
    | Continue

    Senão, se antes == 1 e depois == 2, então:
    | Pai = Pilha[pos]
    |
    | Se pai.direita != NULL, então:
    | | Se pai.direita.balanço >= 0, então:
    | | | RSE(pai)
    | | | atualiza_balanço(pai)
    | | | Continue
    | | Se pai.direita.balanço < 0, então:
    | | | RSD(pai.direita)
    | | | RSE(pai)
    | | | atualiza_balanço(pai.esquerda)
    | | | atualiza_balanço(pai)
    | | | Continue

  free(Pilha[altura_arvore])

Senão, então:
  Mensagem: "Elemento não existe na árvore AVL"
```