ÁRVORE AVL

Prof. André Backes

Problema do balanceamento

- A eficiência da busca em uma árvore binária depende do seu balanceamento.
  - ***O(log N)***, se a árvore está balanceada
  - ***O(N)***, se a árvore não está balanceada
    - **N** corresponde ao número de nós na árvore

Problema do balanceamento

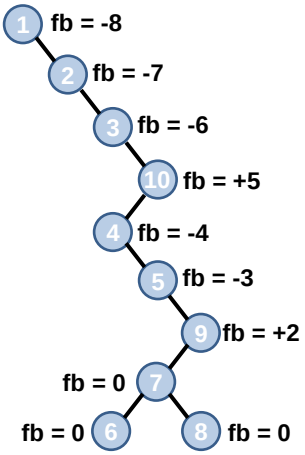
3

- Infelizmente, os algoritmos de inserção e remoção em árvores binárias não garantem que a árvore gerada a cada passo esteja balanceada.
- Dependendo da ordem em que os dados são inseridos na árvore, podemos criar uma árvore na forma de uma escada

Problema do balanceamento

4

- Inserção dos valores {1,2,3,10,4,5,9,7,8,6}



Problema do balanceamento

5

- Solução para o problema de balanceamento
  - Modificar as operações de inserção e remoção de modo a balancear a árvore a cada nova inserção ou remoção.
    - Garantir que a diferença de alturas das sub-árvores esquerda e direita de cada nó seja de no máximo uma unidade
  - Exemplos de árvores balanceadas
    - Árvore AVL
    - Árvore 2-3-4
    - Árvore Rubro-Negra

Árvore AVL

6

- Definição
  - Tipo de árvore binária balanceada com relação a altura das suas sub-árvores
  - Criada por **A**delson-**V**elskii e **L**andis, de onde recebeu a sua nomenclatura, em 1962

Árvore AVL

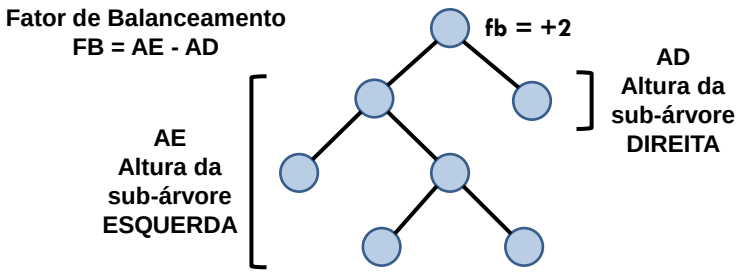
7

- Definição
  - Permite o rebalanceamento local da árvore
    - Apenas a parte afetada pela inserção ou remoção é rebalanceada
  - Usa **rotações simples** ou **duplas** na etapa de rebalanceamento
    - Executadas a cada inserção ou remoção
    - As rotações buscam manter a árvore binária como uma árvore quase completa
    - Custo máximo de qualquer algoritmo é ***O(log N)***

Árvore AVL

8

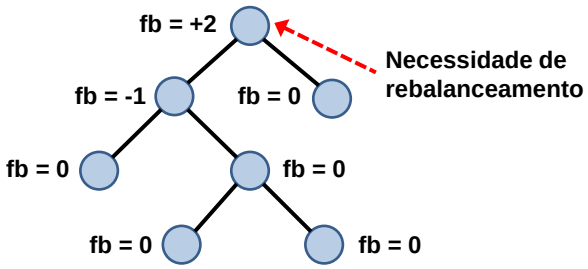
- Objetivo das rotações:
  - Corrigir o **fator de balanceamento** (ou **fb**)
    - Diferença entre as alturas das sub-árvore de um nó
  - Caso uma das sub-árvores de um nó não existir, então a altura dessa sub-arvore será igual a -1.



Árvore AVL

9

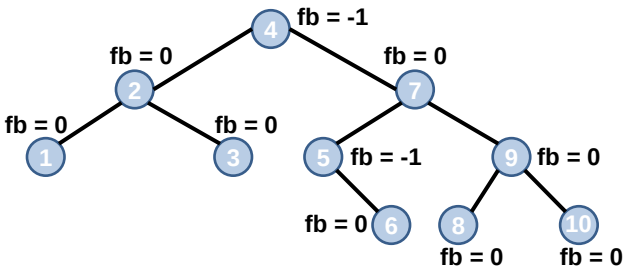
- As alturas das sub-árvores de cada nó diferem de no máximo uma unidade
  - O fator de balanceamento deve ser +1, 0 ou -1
  - Se **fb > +1** ou **fb < -1**: a árvore deve ser balanceada naquele nó



Árvore AVL

10

- Voltando ao problema anterior
- Inserção dos valores {1,2,3,10,4,5,9,7,8,6}



TAD Árvore AVL

11

- Definindo a árvore
  - Criação e destruição: igual a da árvore binária

```
class NO:
    def __init__(self,info):
        self.info = info
        self.altura = 0
        self.esq = None
        self.dir = None

class ArvAVL:
    def __init__(self):
        self.__raiz = None


arv = ArvAVL()
```

TAD Árvore AVL

12

- Calculando o fator de balanceamento

```
def __alturaNO(self, no):
    if(no == None):
        return -1
    else:
        return no.altura

def __fatorBalanceamento_NO(self, no):
    return abs(self.__alturaNO(no.esq) - self.__alturaNO(no.dir))
```

Rotações

13

- Objetivo: corrigir o **fator de balanceamento** (ou **fb**) de cada nó
  - Operação básica para balancear uma árvore AVL
- Ao todo, existem dois tipos de rotação
  - Rotação simples
  - Rotação dupla

Rotações

14

- As rotações diferem entre si pelo sentido da inclinação entre o nó pai e filho
  - Rotação simples
    - O nó desbalanceado (pai), seu filho e o seu neto estão todos no mesmo sentido de inclinação
  - Rotação dupla
    - O nó desbalanceado (pai) e seu filho estão inclinados no sentido inverso ao neto
    - **Equivale a duas rotações simples.**

Rotações

15

- Ao todo, existem duas rotações simples e duas duplas:
  - Rotação **simples a direita** ou **Rotação LL**
  - Rotação **simples a esquerda** ou **Rotação RR**
  - Rotação **dupla a direita** ou **Rotação LR**
  - Rotação **dupla a esquerda** ou **Rotação RL**

Rotações

16

- Rotações são aplicadas no ancestral mais próximo do nó inserido cujo fator de balanceamento passa a ser +2 ou -2
  - Após uma inserção ou remoção, devemos voltar pelo mesmo caminho da árvore e recalcular o fator de balanceamento, **fb**, de cada nó
  - Se o **fb** desse nó for +2 ou -2, uma rotação deverá ser aplicada

Rotação LL

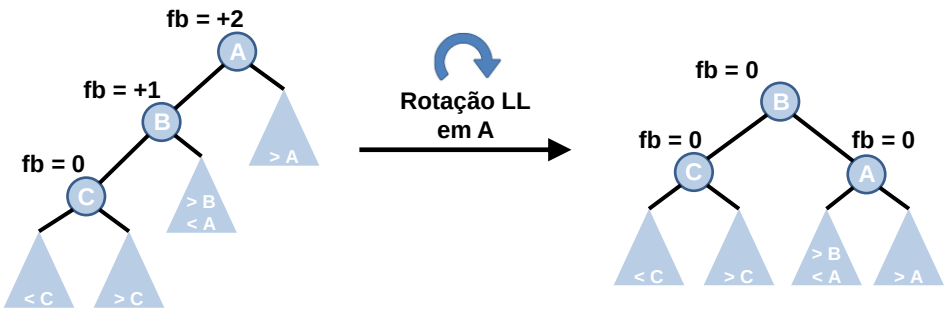
17

- Rotação LL ou rotação simples à direita
  - Um novo nó é inserido na **sub-árvore da esquerda do filho esquerdo** de **A**
    - **A** é o nó desbalanceado
    - Dois movimentos para a esquerda: **LEFT LEFT**
  - É necessário fazer uma rotação à direita, de modo que o nó intermediário **B** ocupe o lugar de **A**, e **A** se torne a sub-árvore direita de **B**

Rotação LL

18

- Exemplo



TAD Árvore AVL

19

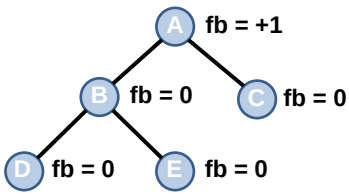
- Rotação LL

```
def __RotacaoLL(self, A):
    B = A.esq
    A.esq = B.dir
    B.dir = A
    A.altura = self.__maior(self.__alturaNO(A.esq),self.__alturaNO(A.dir)) + 1
    B.altura = self.__maior(self.__alturaNO(B.esq),A.altura) + 1
    return B
```

Rotação LL

20

- Passo a passo

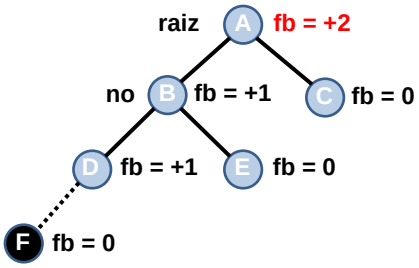


Árvore AVL e fator de balanceamento de cada nó

Rotação LL

21

- Passo a passo



Inserção do nó F na árvore

Árvore fica desbalanceada no nó A.

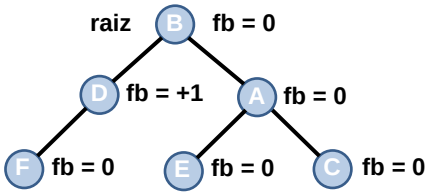
Aplicar Rotação LL no nó A

```
no = raiz.esq
raiz.esq = no.dir
n.dir = raiz;
raiz = no;
```

Rotação LL

22

- Passo a passo



Árvore Balanceada

Rotação RR

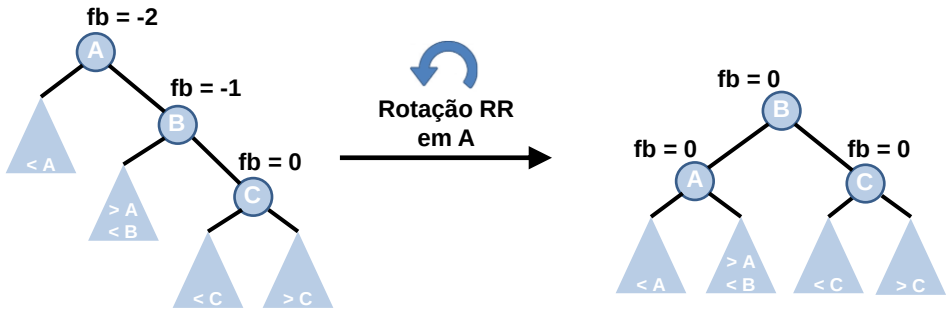
23

- Rotação RR ou rotação simples à esquerda
  - Um novo nó é inserido na **sub-árvore da direita do filho direito** de **A**
    - **A** é o nó desbalanceado
    - Dois movimentos para a direita: **RIGHT RIGHT**
  - É necessário fazer uma rotação à esquerda, de modo que o nó intermediário **B** ocupe o lugar de **A**, e **A** se torne a sub-árvore esquerda de **B**

Rotação RR

24

- Exemplo



TAD Árvore AVL

25

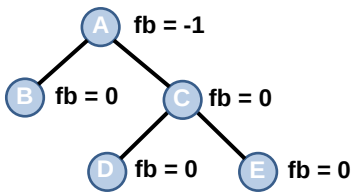
- Rotação RR

```
def __RotacaoRR(self, A):
    B = A.dir
    A.dir = B.esq
    B.esq = A
    A.altura = self.__maior(self.__alturaNO(A.esq),self.__alturaNO(A.dir)) + 1
    B.altura = self.__maior(self.__alturaNO(B.dir),A.altura) + 1
    return B
```

Rotação RR

26

- Passo a passo

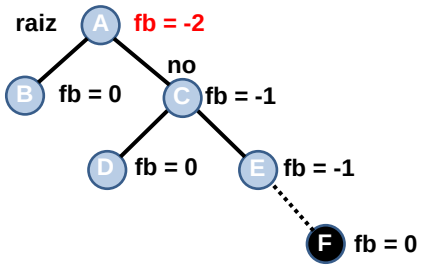


Árvore AVL e fator de balanceamento de cada nó

Rotação RR

27

- Passo a passo



Inserção do nó F na árvore

Árvore fica desbalanceada no nó A.

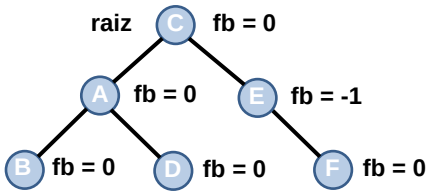
Aplicar Rotação RR no nó A

```
no = raiz.dir
Raiz.dir = no.esq
No.esq = raiz
raiz = no
```

Rotação RR

28

- Passo a passo



Árvore Balanceada

Rotação LR

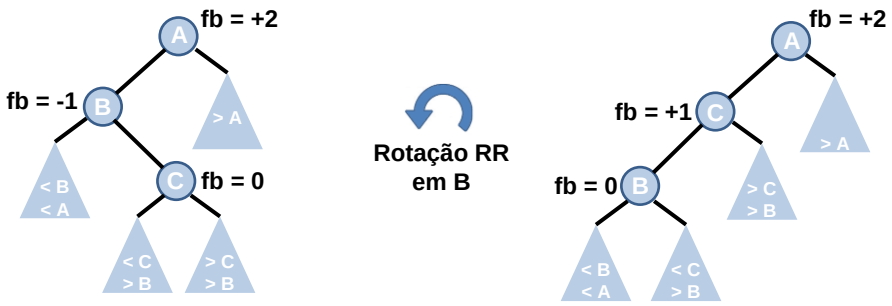
29

- Rotação LR ou rotação dupla à direita
  - Um novo nó é inserido na **sub-árvore da direita do filho esquerdo** de **A**
    - **A** é o nó desbalanceado
    - Um movimento para a esquerda e outro para a direita: **LEFT RIGHT**
  - É necessário fazer uma rotação dupla, de modo que o nó **C** se torne o pai dos nós **A** (filho da direita) e **B** (filho da esquerda)
    - Rotação RR em **B**
    - Rotação LL em **A**

Rotação LR

30

- Exemplo: primeira rotação

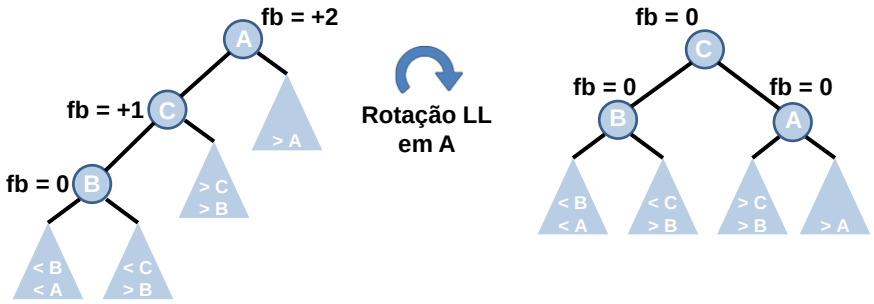



```
def __RotacaoLR(self, A):
    A.esq = self.__RotacaoRR(A.esq)
    A = self.__RotacaoLL(A)
    return A
```

Rotação LR

31

- Exemplo: segunda rotação

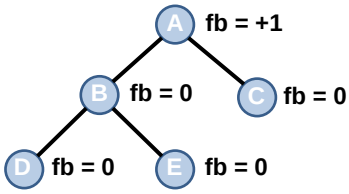



```
def __RotacaoLR(self, A):
    A.esq = self.__RotacaoRR(A.esq)
    A = self.__RotacaoLL(A)
    return A
```

Rotação LR

32

- Passo a passo

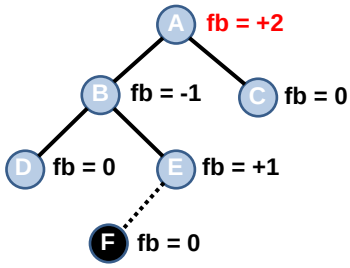


Árvore AVL e fator de balanceamento de cada nó

Rotação LR

33

- Passo a passo



Inserção do nó F na árvore

Árvore fica desbalanceada no nó A.

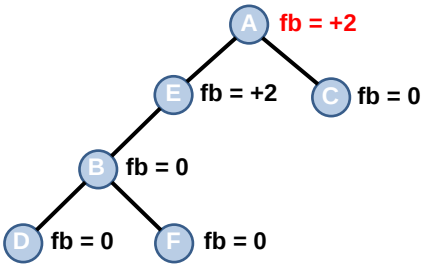
Aplicar Rotação LR no nó A. Isso equivale a:

- Aplicar a Rotação RR no nó B
- Aplicar a Rotação LL no nó A

Rotação LR

34

- Passo a passo

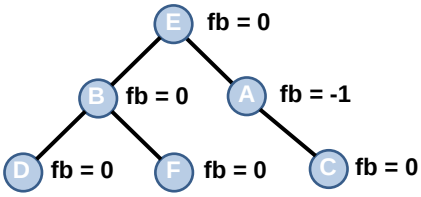


Árvore após aplicar a Rotação RR no nó B

Rotação LR

35

- Passo a passo



Árvore após aplicar a Rotação LL no nó A

Árvore Balanceada

Rotação RL

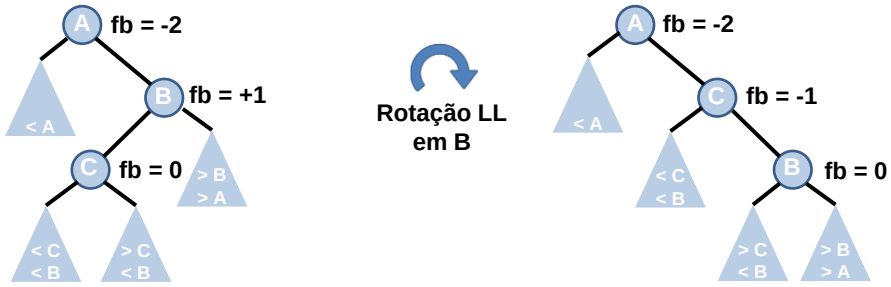
36

- Rotação RL ou rotação dupla à esquerda
  - um novo nó é inserido na **sub-árvore da esquerda do filho direito** de **A**
    - **A** é o nó desbalanceado
    - Um movimento para a direita e outro para a esquerda: **RIGHT LEFT**
  - É necessário fazer uma rotação dupla, de modo que o nó **C** se torne o pai dos nós **A** (filho da esquerda) e **B** (filho da direita)
    - Rotação LL em **B**
    - Rotação RR em **A**

Rotação RL

37

- Exemplo

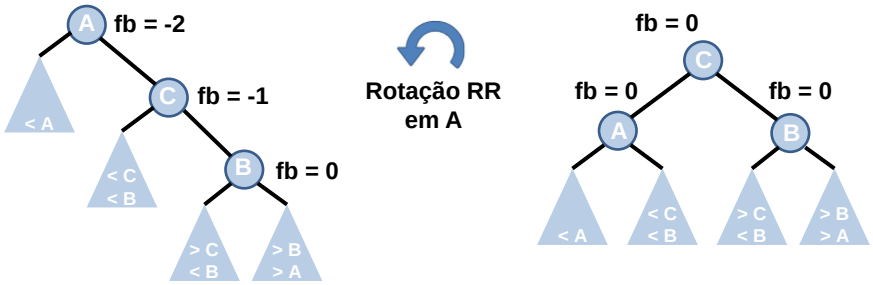



```
def __RotacaoRL(self, A):
    A.dir = self.__RotacaoLL(A.dir)
    A = self.__RotacaoRR(A)
    return A
```

Rotação RL

38

- Exemplo

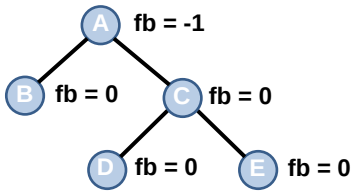



```
def __RotacaoRL(self, A):
    A.dir = self.__RotacaoLL(A.dir)
    A = self.__RotacaoRR(A)
    return A
```

Rotação RL

39

- Passo a passo

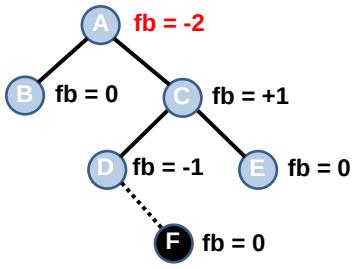


Árvore AVL e fator de balanceamento de cada nó

Rotação RL

40

- Passo a passo



Inserção do nó F na árvore

Árvore fica desbalanceada no nó A.

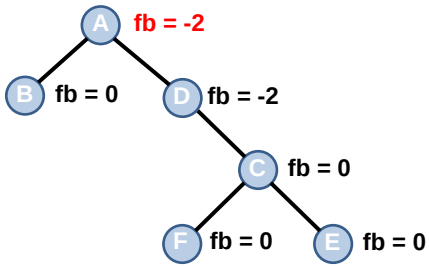
Aplicar Rotação RL no nó A. Isso equivale a:

- Aplicar a Rotação LL no nó C
- Aplicar a Rotação RR no nó A

Rotação RL

41

- Passo a passo

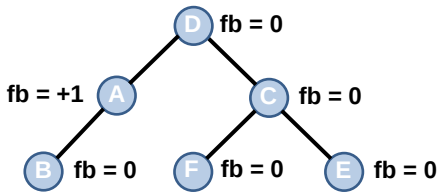


Árvore após aplicar a Rotação LL no nó C

Rotação RL

42

- Passo a passo



Árvore após aplicar a Rotação RR no nó A

Árvore Balanceada

Quando usar cada rotação?

43

- Uma dúvida muito comum é quando utilizar cada uma das quatro rotações

Fator de Balanceamento de A	Fator de Balanceamento de B	Posições dos nós B e C em relação ao nó A	Rotação
+2	+1	B é filho à esquerda de A C é filho à esquerda de B	LL
-2	-1	B é filho à direita de A C é filho à direita de B	RR
+2	-1	B é filho à esquerda de A C é filho à direita de B	LR
-2	+1	B é filho à de direita A C é filho à esquerda de B	RL

Quando usar cada rotação?

44

- Sinais iguais: rotação simples
  - Sinal positivo: rotação à direita (LL)
  - Sinal negativo: rotação à esquerda (RR)

Fator de Balanceamento de A	Fator de Balanceamento de B	Posições dos nós B e C em relação ao nó A	Rotação
+2	+1	B é filho à esquerda de A C é filho à esquerda de B	LL
-2	-1	B é filho à direita de A C é filho à direita de B	RR
+2	-1	B é filho à esquerda de A C é filho à direita de B	LR
-2	+1	B é filho à de direita A C é filho à esquerda de B	RL

Quando usar cada rotação?

45

- Sinais diferentes: rotação dupla
  - **A** positivo: rotação dupla a direita (LR)
  - **A** negativo: rotação dupla a esquerda (RL)

Fator de Balanceamento de A	Fator de Balanceamento de B	Posições dos nós B e C em relação ao nó A	Rotação
+2	+1	B é filho à esquerda de A C é filho à esquerda de B	LL
-2	-1	B é filho à direita de A C é filho à direita de B	RR
+2	-1	B é filho à esquerda de A C é filho à direita de B	LR
-2	+1	B é filho à de direita A C é filho à esquerda de B	RL

Árvore AVL: Inserção

46

- Para inserir um valor **V** na árvore
  - Se a raiz é igual a **NULL,** insira o nó
  - Se **V** é menor do que a raiz: vá para a **sub-árvore esquerda**
  - Se **V** é maior do que a raiz: vá para a **sub-árvore direita**
  - Aplique o método **recursivamente**
- Dessa forma, percorremos um conjunto de nós da árvore até chegar ao nó folha que irá se tornar o pai do novo nó

Árvore AVL: Inserção

47

- Uma vez inserido o novo nó
  - Devemos voltar pelo caminho percorrido e calcular o fator de balanceamento de cada um dos nós visitados
  - Aplicar a rotação necessária para restabelecer o balanceamento da árvore se o fator de balanceamento for **+2** ou **-2**

TAD Árvore AVL

48

- Inserção

```
def insere(self, valor):
    if(self.busca(valor)):
        return False #valor já existe na árvore
    else:
        self.__raiz = self.__insereNO(self.__raiz, valor)
        return True
```

TAD Árvore AVL

49

- Inserção

```
def __insereNO(self,atual,valor):
    if(atual == None): # árvore vazia ou nó folha
        novo = NO(valor)
        return novo
    else:
        if(valor < atual.info):
            atual.esq = self.__insereNO(atual.esq, valor)
            if(self.__fatorBalanceamento_NO(atual) >= 2):
                if(valor < atual.esq.info):
                    atual = self.__RotacaoLL(atual)
                else:
                    atual = self.__RotacaoLR(atual)
            #continua
```

TAD Árvore AVL

50

- Inserção

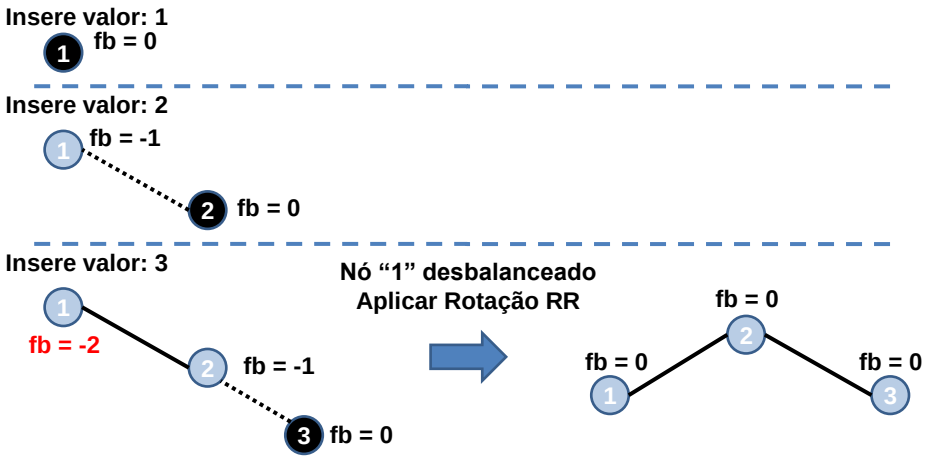
```
    #continuação
else:
    atual.dir = self.__insereNO(atual.dir, valor)
    if(self.__fatorBalanceamento_NO(atual) >= 2):
        if(valor > atual.dir.info):
            atual = self.__RotacaoRR(atual)
        else:
            atual = self.__RotacaoRL(atual)

atual.altura = self.__maior(self.__alturaNO(atual.esq),
                            self.__alturaNO(atual.dir)) + 1
return atual
```

Árvore AVL: Inserção

51

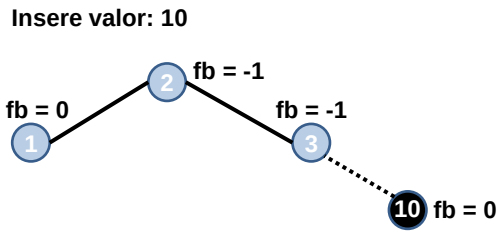
- Passo a passo



Árvore AVL: Inserção

52

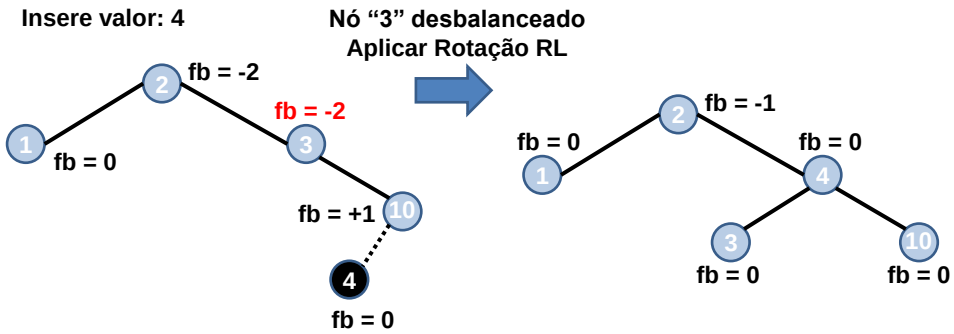
- Passo a passo



Árvore AVL: Inserção

53

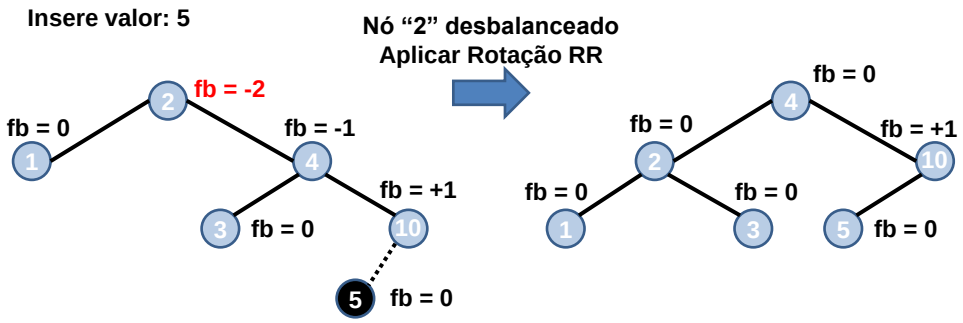
- Passo a passo



Árvore AVL: Inserção

54

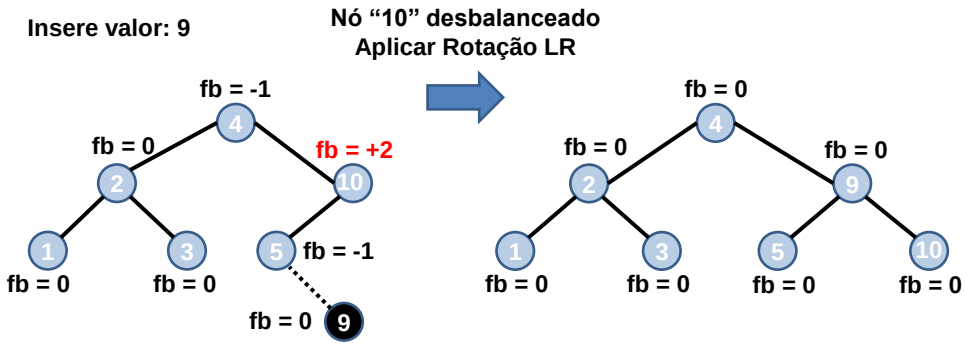
- Passo a passo



Árvore AVL: Inserção

55

- Passo a passo



Árvore AVL: Remoção

56

- Como na inserção, temos que percorremos um conjunto de nós da árvore até chegar ao nó que será removido
  - Existem 3 tipos de remoção
    - Nó folha (sem filhos)
    - Nó com 1 filho
    - Nó com 2 filhos

Árvore AVL: Remoção

57

- Uma vez removido o nó
  - Devemos voltar pelo caminho percorrido e calcular o fator de balanceamento de cada um dos nós visitados
  - Aplicar a rotação necessária para restabelecer o balanceamento da árvore se o fator de balanceamento for **+2** ou **-2**
    - **Remover** um nó da sub-árvore **direita** equivale a **inserir** um nó na sub-árvore **esquerda**

TAD Árvore AVL

58

- Remoção
  - Trabalha com 3 funções
    - Interface
    - Busca pelo nó
    - Remoção do nó com 2 filhos

```
def remove(self, valor):
    # FUNÇÃO QUE FAZ A INTERFACE COM
    # O USUÁRIO

def __removeNO(self, atual, valor):
    # FUNÇÃO RESPONSÁVEL PELA BUSCA
    # DO NÓ A SER REMOVIDO

def __procuraMenor(self, atual):
    # FUNÇÃO RESPONSÁVEL POR TRATAR
    # A REMOÇÃO DE UM NÓ COM 2 FILHOS
```

TAD Árvore AVL

59

- Remoção

```
def remove(self, valor):
    if(self.__raiz == None or not self.busca(valor)):
        return False #árvore vazia ou valor não existe na árvore
    else:
        self.__raiz = self.__removeNO(self.__raiz, valor)
        return True
```

TAD Árvore AVL

60

- Remoção

```
def __removeNO(self, atual, valor):
    if(atual.info == valor): #achou o nó a ser removido
        if(atual.esq == None or atual.dir == None): # nó tem 1 filho ou nenhum
            if(atual.esq != None):
                atual = atual.esq
            else:
                atual = atual.dir
        else: # nó tem 2 filhos
            temp = self.__procuraMenor(atual.dir)
            atual.info = temp.info
            atual.dir = self.__removeNO(atual.dir, atual.info)
            if(self.__fatorBalanceamento_NO(atual) >= 2):
                if(self.__alturaNO(atual.esq.dir) <= self.__alturaNO(atual.esq.esq)):
                    atual = self.__RotacaoLL(atual)
                else:
                    print('ok')
                    atual = self.__RotacaoLR(atual)
    if(atual != None):
        atual.altura = self.__maior(self.__alturaNO(atual.esq),
                                    self.__alturaNO(atual.dir)) + 1
    #continua
```

Pai tem 1 ou nenhum filho

Pai tem 2 filhos: Substituir pelo nó mais a esquerda da sub-árvore da direita

Corrige a altura

TAD Árvore AVL

61

- Remoção

```
    #continuação
else:# procura o nó a ser removido
    if(valor < atual.info):
        atual.esq = self.__removeNO(atual.esq, valor)
        if(self.__fatorBalanceamento_NO(atual) >= 2):
            if(self.__alturaNO(atual.dir.esq) <= self.__alturaNO(atual.dir.dir)):
                atual = self.__RotacaoRR(atual)
            else:
                atual = self.__RotacaoRL(atual)
    else:
        atual.dir = self.__removeNO(atual.dir, valor)
        if(self.__fatorBalanceamento_NO(atual) >= 2):
            if(self.__alturaNO(atual.esq.dir) <= self.__alturaNO(atual.esq.esq)):
                atual = self.__RotacaoLL(atual)
            else:
                atual = self.__RotacaoLR(atual)

    atual.altura = self.__maior(self.__alturaNO(atual.esq),
                                self.__alturaNO(atual.dir)) + 1

return atual
```

TAD Árvore AVL

62

- Remoção

```
def __procuraMenor(self, atual):
    no1 = atual
    no2 = atual.esq
    while(no2 != None):
        no1 = no2
        no2 = no2.esq

    return no1
```

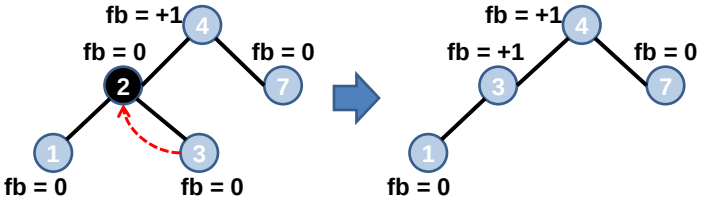
Procura pelo nó mais a esquerda

Árvore AVL: Remoção

63

- Passo a passo

Remove valor: 2

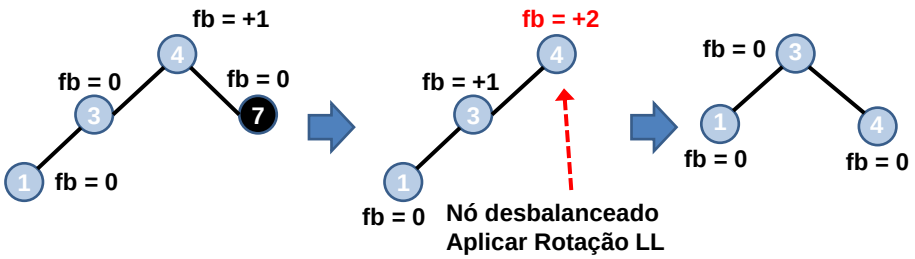


Árvore AVL: Remoção

64

- Passo a passo

Remove valor: 7



Material Complementar

65

- Vídeo Aulas
  - Aula 78 – Árvores Balanceadas:
  - youtu.be/Au-6c55J90c
  - Aula 79: Árvore AVL: Definição:
  - youtu.be/4eO3UbTiRyo
  - Aula 80: Árvore AVL: Implementação:
  - youtu.be/l5cl39jdnow
  - Aula 81: Árvore AVL: Tipos de Rotação:
  - youtu.be/1HkWqH7L2rU
  - Aula 82: Árvore AVL: Implementando as Rotações:
  - youtu.be/6OJ8stXwdq0
  - Aula 83: Árvore AVL: Inserção:
  - youtu.be/lQsVUxa3Auk
  - Aula 84: Árvore AVL: Remoção:
  - youtu.be/F7_Daymw-WM