# Árvore AVL

Rômulo César Silva

Unioeste

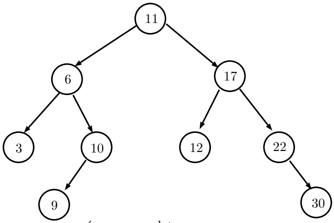
Junho de 2016

# Sumário

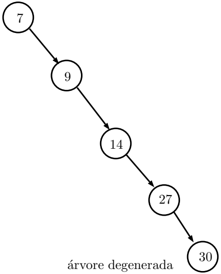
# Eficiência de Operações em Árvores Binárias

Quanto custam as operações de inserção, remoção e busca em uma árvore binária de busca?

- Depende de como estão distribuídos os nós. Caso a árvore seja completa, o custo dessas operações é $O(lg\ n)$, que corresponde à altura da árvore.
- Se a árvore se degenera para uma lista, o custo passa a ser $O(n)$



árvore completa



árvore degenerada

# Balanceamento em Árvores Binárias

**Balanceamento**

A operação de **balanceamento** consiste em fazer alterações na árvore binária de maneira a mantê-la próxima de uma árvore completa.

- As operações de inserção, busca e remoção terão complexidade $O(lg\ n)$
- Há diferentes critérios de balanceamento. Conforme os critérios de balanceamento adotados, a árvore recebe uma denominação distinta.

# Árvore AVL

## Árvore AVL

Uma árvore binária de busca $T$ é do tipo AVL se para qualquer nó $v$ de $T$, as alturas das subárvores esquerda e direita diferem no máximo de 1.

## Fator de Balanceamento

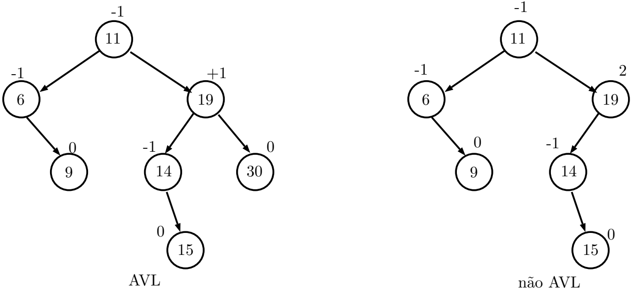
Dado um nó $v$, o **fator de balanceamento** de $v$, é dado por:

$$FB(v) = h_e(v) - h_d(v)$$

Observações:

- o nome AVL vem de seus propositores Adelson-Velskii e Landis (1962)
- Em uma AVL $T$, $FB(v) \in \{-1, 0, 1\} \; \forall v \in T$
- o fator de balanceamento pode ser calculado a cada acesso (cálculo implícito) ou armazenado em cada nó (explícito).

# Árvore AVL



# Árvore AVL - inserção

A inserção é feita como em uma árvore binária de busca, com as seguintes situações:

- a árvore vazia é AVL
- se a inserção é feita do lado mais baixo: a altura final se mantém e portanto, a árvore continua AVL
- se as alturas das subárvores são iguais: a altura final aumenta de 1, mas a árvore continua AVL
- se a inserção é feita do lado mais alto: é necessário rearranjar o nós (rebalanceamento) para manter o critério AVL
- o rebalanceamento quando feito precisa manter a propriedade de árvore binária de busca

# Árvore AVL - inserção

Dado um nó $v$ de uma árvore AVL, se $FB(v)$ é:

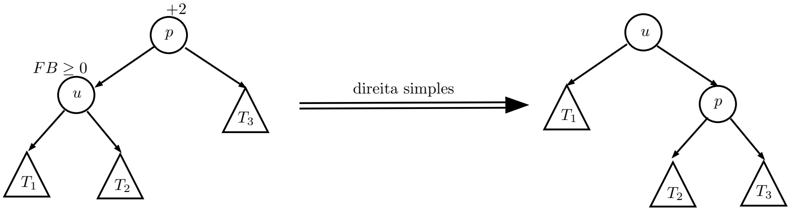
- 0: subárvore esquerda e direita têm a mesma altura
- $-1$: subárvore direita é mais alta que esquerda
- $+1$: subárvore esquerda é mais alta que direita

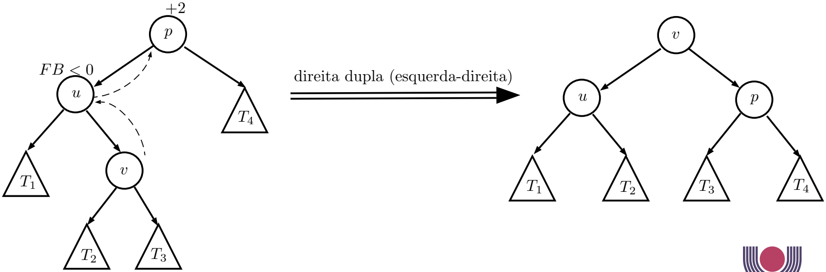
# Árvore AVL - inserção

Um nó $v$ está desbalanceado quando $FB(v)$ é 2 ou $-2$. Assim, se a inserção ou remoção de uma chave ocasiona o desbalanceamento de um nó, é necessário rebalancear a árvore, através de rotações. Há 4 rotações possíveis em árvores AVL:

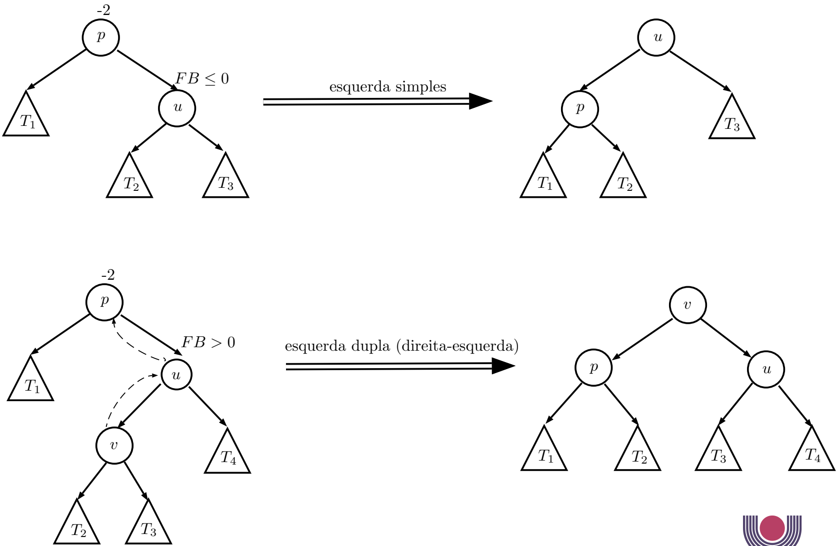
- simples à esquerda
- simples à direita
- dupla à esquerda: corresponde a 1 rotação à direita e depois 1 à esquerda
- dupla à direita: corresponde a 1 rotação à esquerda e depois 1 à direita

# Árvore AVL


+2
p
FB ≥ 0
u
T1
T2
T3
direita simples
u
T1
p
T2
T3


+2
p
FB < 0
u
T1
v
T2
T3
T4
direita dupla (esquerda-direita)
v
u
p
T1
T2
T3
T4

# Árvore AVL

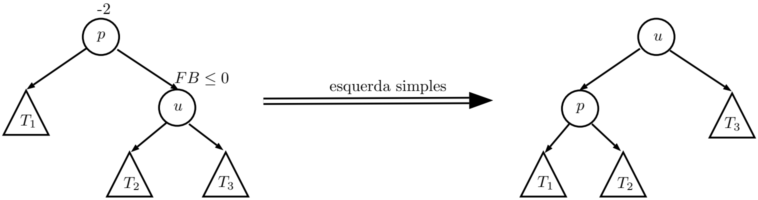

-2
p
FB ≤ 0
u
T1
T2
T3
esquerda simples
u
p
T3
T1
T2
-2
p
FB > 0
u
v
T1
T4
T2
T3
esquerda dupla (direita-esquerda)
v
p
u
T1
T2
T3
T4

# Árvore AVL - estrutura

Estrutura:

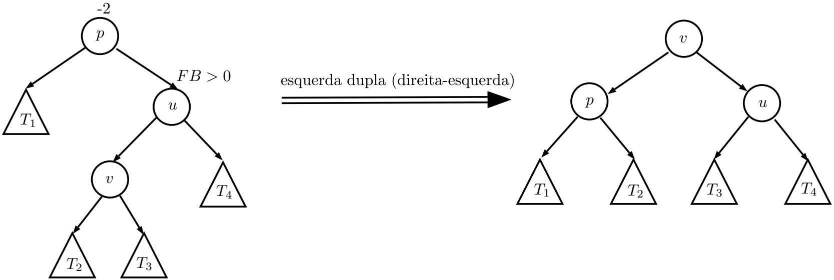
```
 struct no {
    int info;          // informação armazenada
    struct no * esq; // subárvore esquerda
    struct no * dir; // subárvore direita
    int fb;            // fator de balanceamento
 };
typedef struct no* arvoreAVL; //árvore é um ponteiro
                              //  para um nó
```

# Árvore AVL - rotação à esquerda




```
arvoreAVL rotacaoEsquerda(arvoreAVL p) {
   arvoreAVL u = p->dir;
   arvoreAVL t2 = u->esq;
   u->esq = p;
   p->dir = t2;
   return u;
}
```

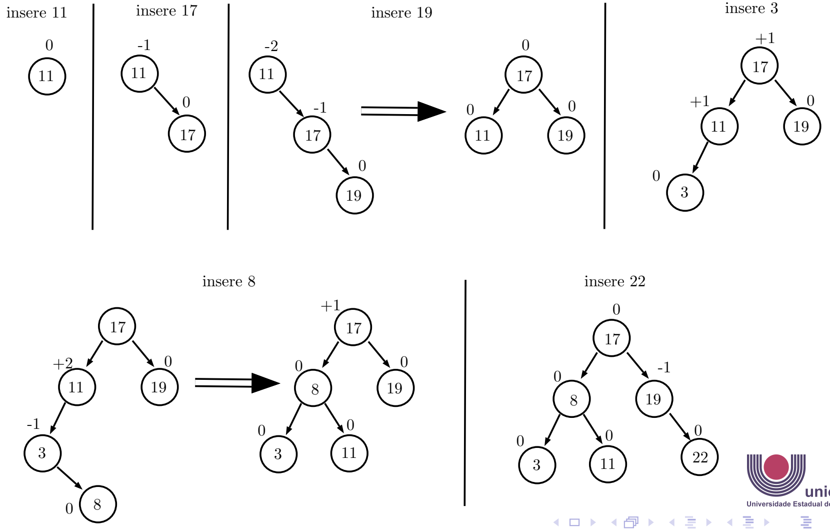
# Árvore AVL - rotação à esquerda dupla




```
arvoreAVL rotacaoDireitaEsquerda(arvoreAVL p) {
   arvoreAVL u = p->dir;
   arvoreAVL v = u->esq;
   arvoreAVL t2 = v->esq;
   arvoreAVL t3 = v->dir;
   p->dir = t2;
   u->esq = t3;
   v->esq = p;
   v->dir = u;
   return v;
}
```

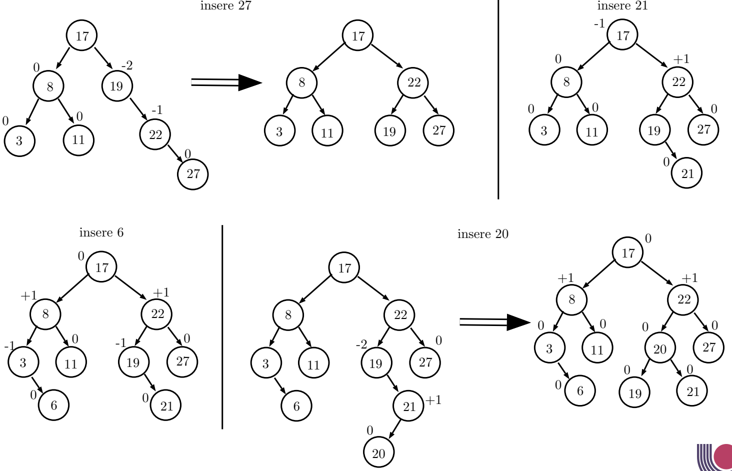
# Árvore AVL - inserção - exemplo

Inserção das chaves 11, 17, 19, 3, 8, 22, 27, 21, 6, e 20, nessa ordem.



# Árvore AVL - inserção - exemplo

Inserção das chaves 11, 17, 19, 3, 8, 22, 27, 21, 6, e 20, nessa ordem.



# Árvore AVL - inserção

```
//Insere uma chave na árvore AVL
//Entrada: raiz da árvore AVL e valor da chave
//Retorno: 1 se altura da árvore aumentou, 0 caso contrário

int insercao(arvoreAVL* r, int x) {
   if(vazia(*r)) { // caso 1: árvore vazia
      *r = (arvoreAVL) malloc(sizeof(struct no));
      (*r)->info = x;
      (*r)->esq = (*r)->dir = NULL;
      (*r)->fb = 0;
      return 1;
   }
   // árvore não vazia
   if(x < (*r)->info){ // caso 2: inserir na árvore esquerda
```

# Árvore AVL - inserção (cont.)

```
if(x < (*r)->info){ // caso 2: inserir na árvore esquerda
  if(insercao(&((*r)->esq), x)) {
    switch((*r)->fb){
      case -1: (*r)->fb = 0; return 0;
      case 0: (*r)->fb = 1; return 1;
      case 1: //rebalancear
             if((*r)->esq->fb >= 0) {
               *r = rotacaoDireita(*r);
               (*r)->dir->fb = 0;
             } else {
               *r = rotacaoEsquerdaDireita(*r);
               switch((*r)->fb) {
                 case -1: (*r)->esq->fb = 1; (*r)->dir->fb = 0; break;
                 case 0:  (*r)->esq->fb = 0; (*r)->dir->fb = 0; break;
                 case 1:  (*r)->esq->fb = 0; (*r)->dir->fb = -1; break;
               }
             }
             (*r)->fb = 0; // atualiza FB da nova raiz
             return 0;
    }
  }
}
else if(insercao(&((*r)->dir), x)){ // caso 3: inserir na árvore direita
```

# Árvore AVL - inserção (cont.)

```
else if(insercao(&((*r)->dir), x)){ // caso 3: inserir na árvore direita
       switch((*r)->fb){
         case 1: (*r)->fb = 0; return 0;
         case 0: (*r)->fb = -1; return 1;
         case -1: //rebalancear
                 if((*r)->dir->fb <= 0){
                   *r = rotacaoEsquerda(*r);
                   (*r)->esq->fb = 0;
                 } else {
                   *r = rotacaoDireitaEsquerda(*r);
                   switch((*r)->fb) {
                     case -1: (*r)->esq->fb = 1; (*r)->dir->fb = 0; break;
                     case 0:  (*r)->esq->fb = 0; (*r)->dir->fb = 0; break;
                     case 1:  (*r)->esq->fb = 0; (*r)->dir->fb = -1; break;
                   }
                 }
                 (*r)->fb = 0;
                 return 0;
     }
  }
}
```

# Árvore AVL - remoção

A remoção é feita como em uma árvore binária de busca. Em seguida, caso seja necessário é preciso fazer rotações para manter o balanceamento da árvore.

# Árvore AVL - remoção - exemplo

Remoção da chave 9: