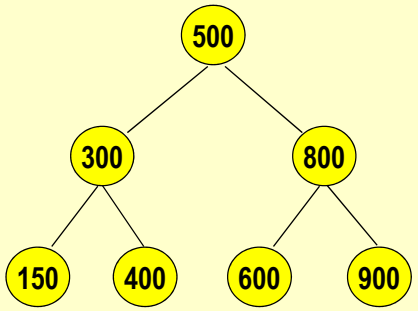
Árvores Balanceadas

Árvores Binárias de Pesquisa

- Apresentam uma relação de ordem
- A ordem é definida pela chave
- Operações:
  - inserir
  - consultar
  - excluir



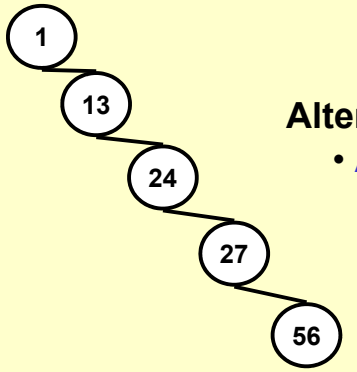
Problemas com ABP

Exemplo:

- Inserção: 10, 5, 15, 20, 25, 30, 35
- Inserção: 1, 13, 24, 27, 56

Problemas com ABP

- Desbalanceamento progressivo
- Exemplo:
  - inserção: 1, 13, 24, 27, 56



Alternativa de solução:

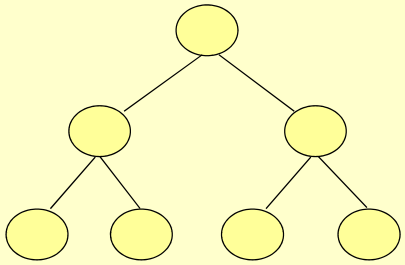
- **Árvores balanceadas**
  - **AVL**

Árvores balanceadas por ALTURA

Uma árvore binária é

completamente balanceada

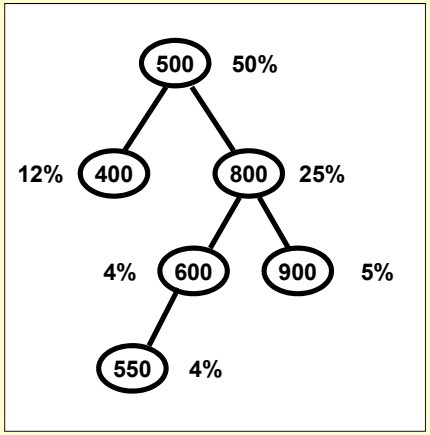
se a distância média dos nodos até a raiz for mínima



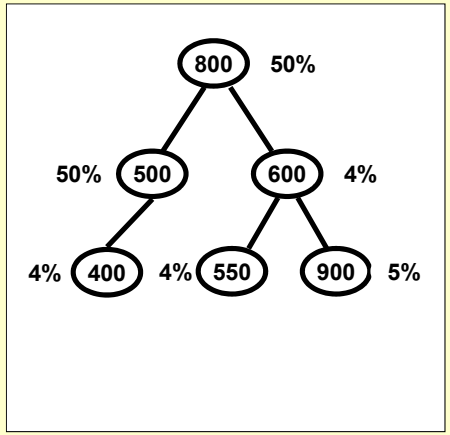
Balanceamento de Árvores

- Distribuição equilibrada dos nós
  - otimizar as operações de consulta
  - diminuir o número médio de comparações
- Distribuição
  - uniforme
  - não uniforme
    - chaves mais solicitadas mais perto da raiz

Por Freqüência X Por Altura

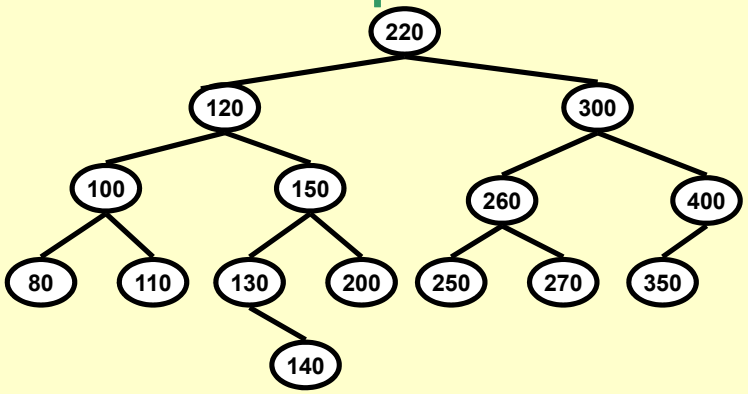


Splay



AVL
Rubro-Negras

Balanceamento por ALTURA



Árvore não completamente balanceada

Árvores AVL

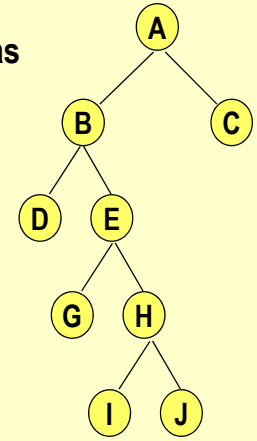
Adelson-Velskii e Landis (1962)

Uma árvore AVL é uma *árvore binária de pesquisa* (ABP) construída de tal modo que a altura de sua subárvore direita difere da altura da subárvore esquerda de no máximo 1.

Árvores balanceadas por altura

HB(k)-Tree → Height-Balanced k -Tree

- árvore binária
- para qualquer nodo, as alturas de suas duas subárvores não diferem de mais do que *k* unidades
- cada uma das subárvores do nodo apresenta a propriedade *FATOR(k)*



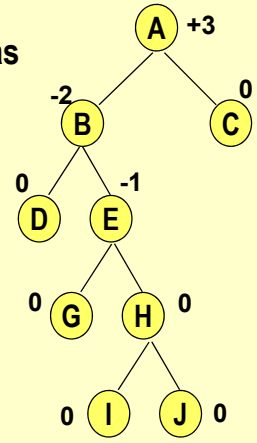
Árvores balanceadas por altura

HB(k)-Tree → Height-Balanced k -Tree

- árvore binária
- para qualquer nodo, as alturas de suas duas subárvores não diferem de mais do que *k* unidades
- cada uma das subárvores do nodo apresenta a propriedade *FATOR(k)*

Ex: verificar se a árvore ao lado é

FATOR(1) → Não
FATOR(2) → Não
FATOR(3) → Sim



Árvores AVL

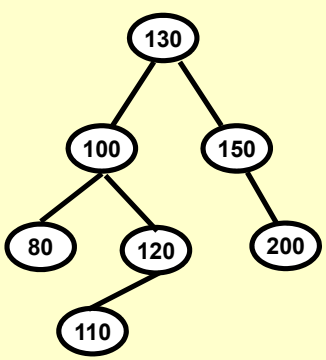
Adelson-Velskii e Landis (1962)

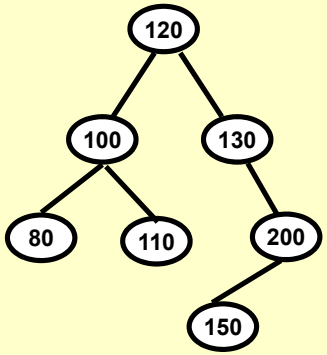
- árvores FATOR(1) são chamadas Árvores AVL

Uma árvore AVL é uma *árvore binária de pesquisa* (ABP) construída de tal modo que a altura de sua subárvore direita difere da altura da subárvore esquerda de no máximo 1.

Exercício:

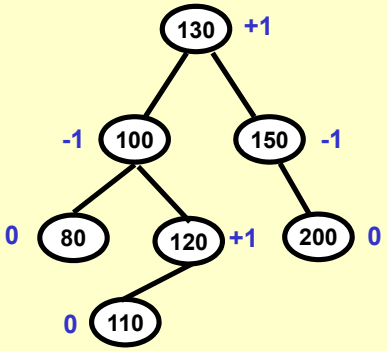
Verifique quais das ABP são AVL:



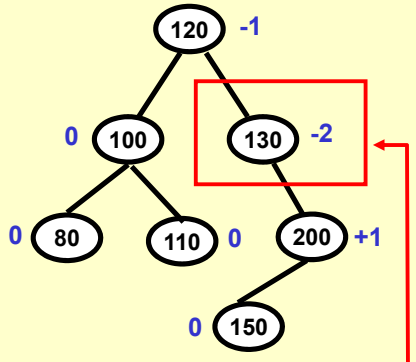


Exercício: Resposta

Verifique quais das ABP são AVL:



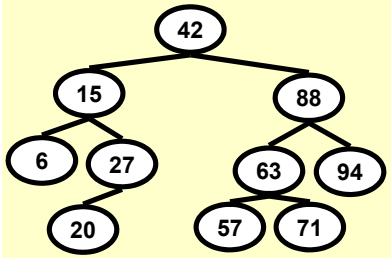
AVL

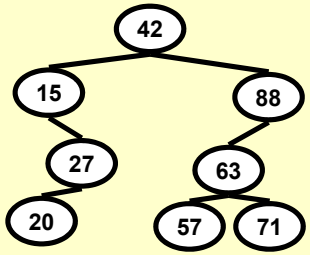


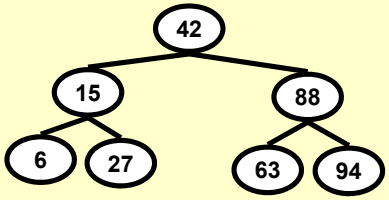
Não AVL

Exercício

Verifique quais das ABP são AVL:

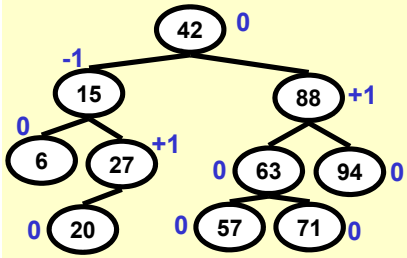




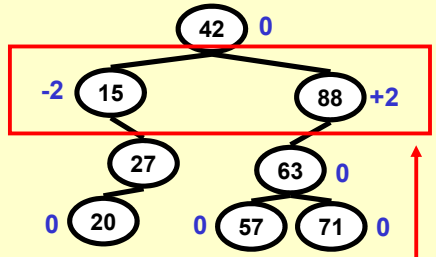


Exercício: Resposta

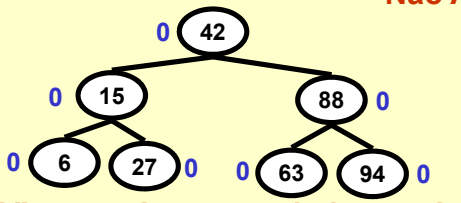
Verifique quais das ABP são AVL:



AVL



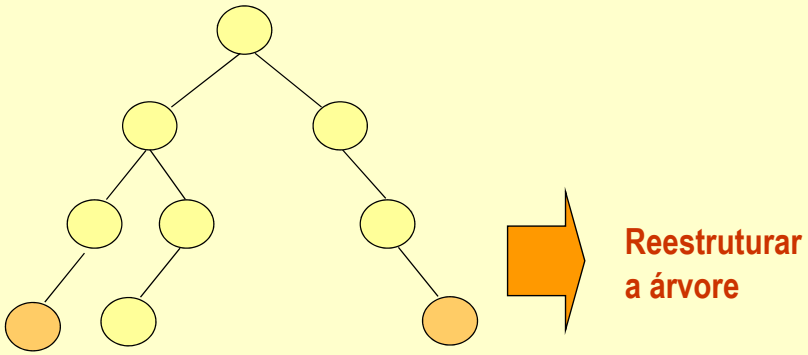
Não AVL



AVL: completamente balanceada

Operações

- por exemplo: INSERÇÃO
  - deve ser preservada a propriedade AVL



Operações

- Como manter uma árvore AVL sempre balanceada após uma inserção ou exclusão?
  - Através de uma operação de ROTAÇÃO
- Característica da operação
  - preservar a ordem das chaves
  - basta uma execução da operação de rotação para tornar a árvore AVL novamente

Balanceamento de Árvore AVL com Rotação

☞ Rotação Simples
- à direita
- à esquerda

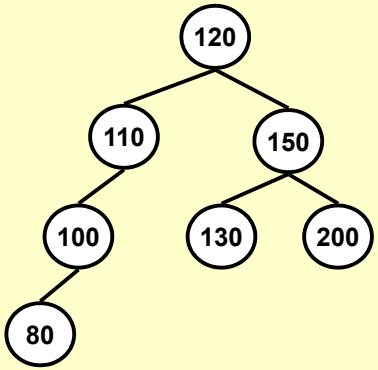
☞ Rotação Dupla
- à direita
- à esquerda

Rotação Simples DIREITA

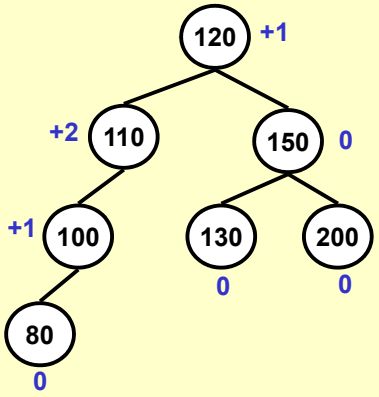
Toda vez que uma subárvore fica com um fator:
- positivo e sua subárvore da esquerda também tem um fator positivo

ROTAÇÃO SIMPLES À DIREITA

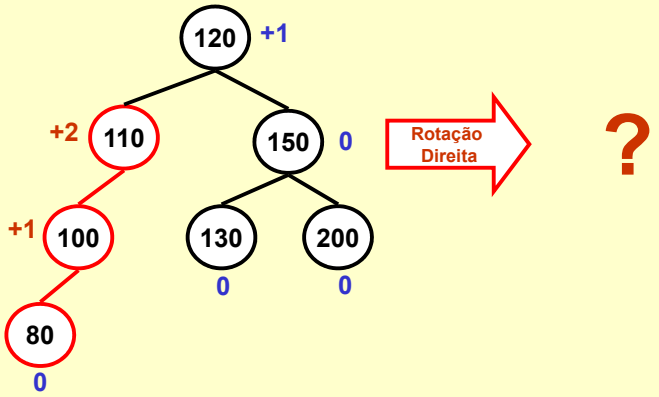
Rotação Direita



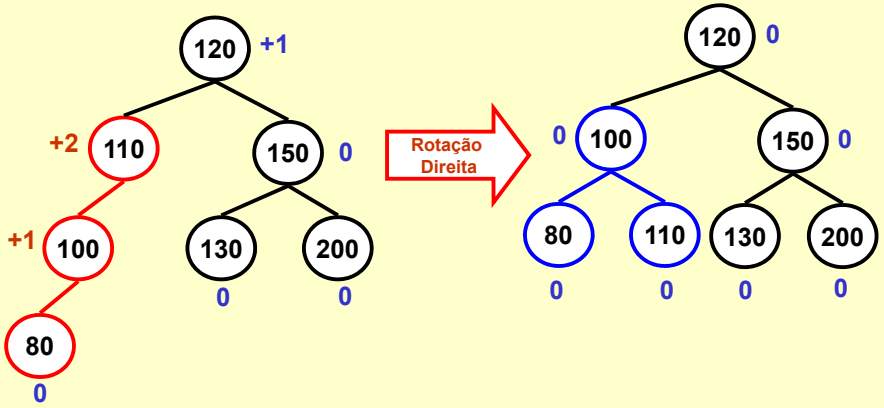
Rotação Direita



Rotação Direita

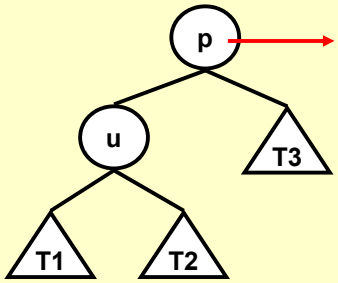


Rotação Direita



Balanceamento de Árvore AVL com Rotação

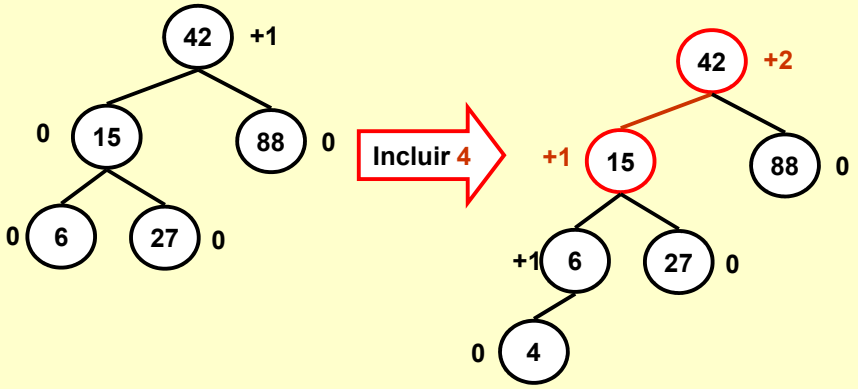
☞ **Rotação Simples**
- à direita



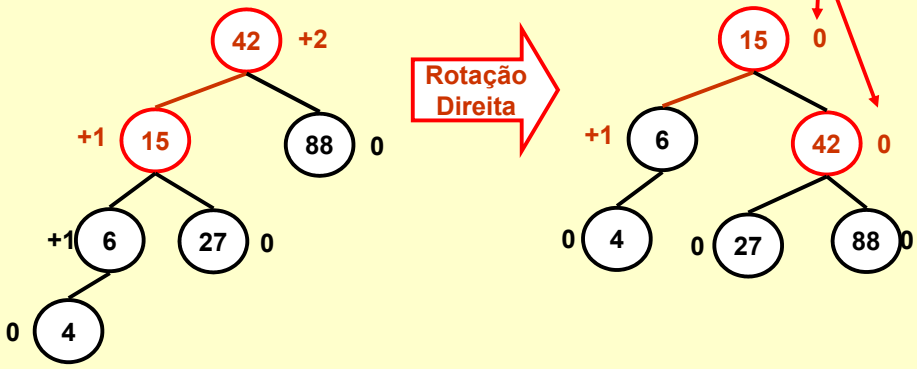
Nó p é a raiz de transformação

T1, T2, T3 e T4 são subárvores (vazias ou não)

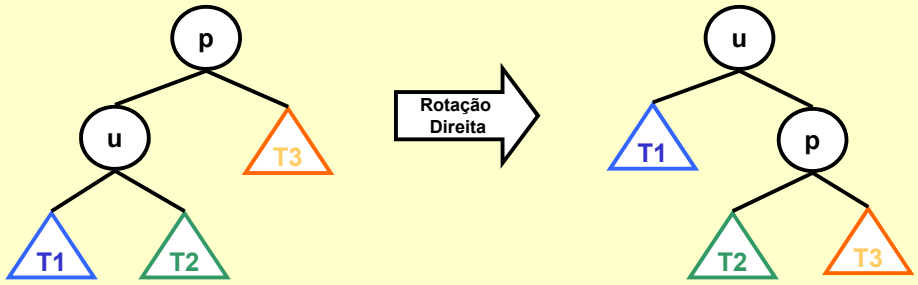
Rotação Direita



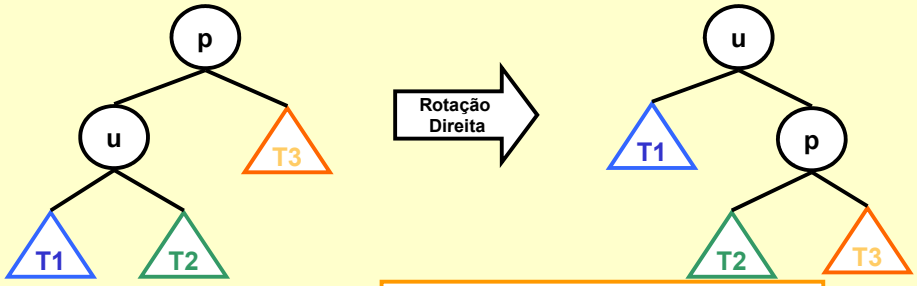
Rotação Direita



Rotação Direita



Rotação Direita




```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
pNodoA* rotacao_direita(pNodoA* pt){
   pNodoA *ptu;

   ptu = pt->esq;
   pt->esq = ptu->dir;
   ptu->dir = pt;
   pt->FB = 0;
   pt = ptu;
   return pt;
}
```

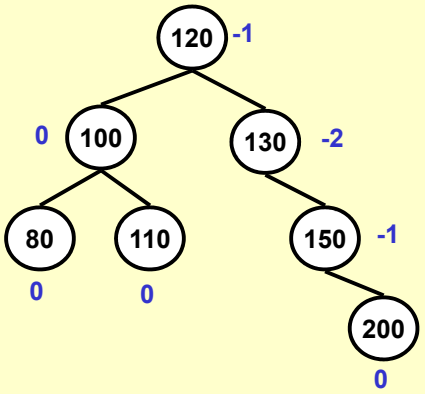

Rotação Simples ESQUERDA

Toda vez que uma subárvore fica com um fator:

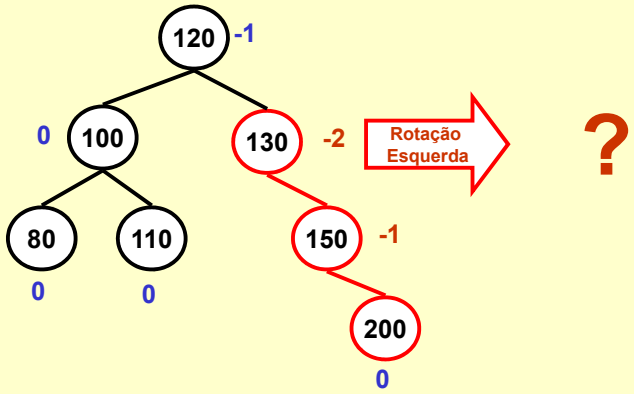
- **negativo e sua subárvore da direita também tem um fator negativo**

ROTAÇÃO SIMPLES À ESQUERDA

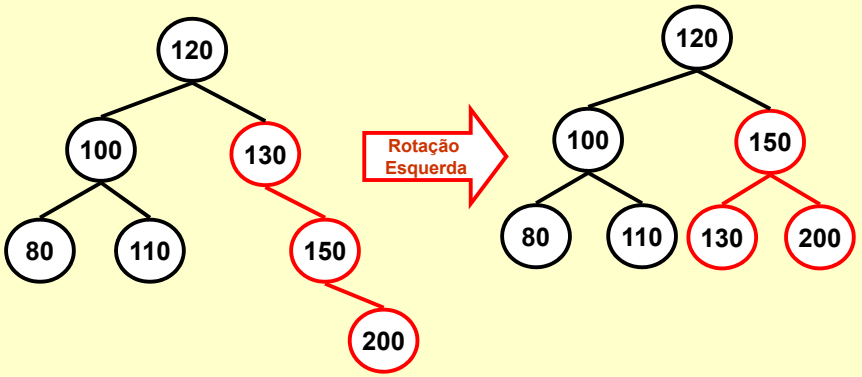
Rotação Esquerda



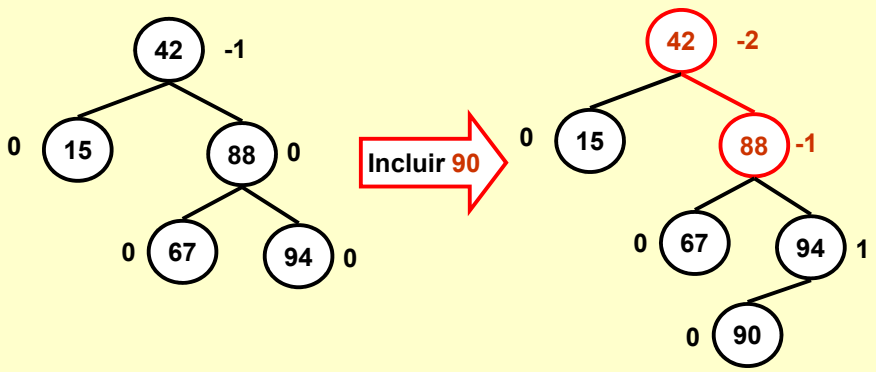
Rotação Esquerda



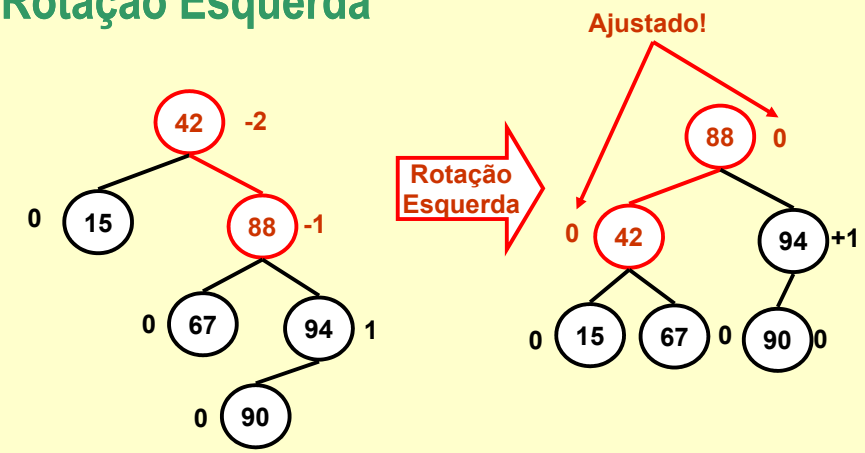
Rotação Esquerda



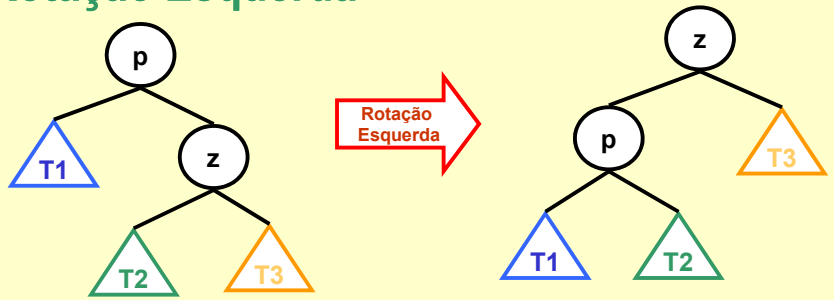
Rotação Esquerda



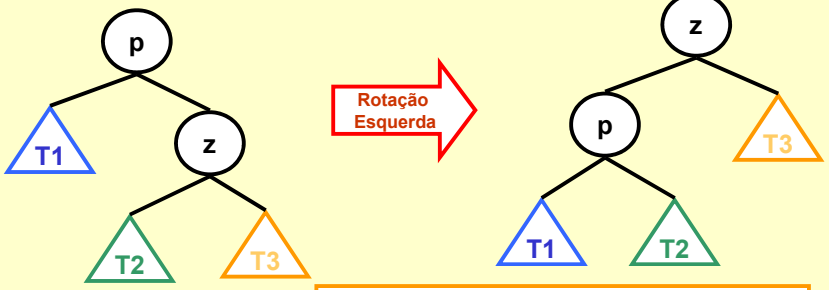
Rotação Esquerda



Rotação Esquerda



Rotação Esquerda




```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
pNodoA* rotacao_esquerda(pNodoA *pt){
   pNodoA *ptu;

   ptu = pt->dir;
   pt->dir = ptu->esq;
   ptu->esq = pt;
   pt->FB = 0;
   pt = ptu;
   return pt;
}
```

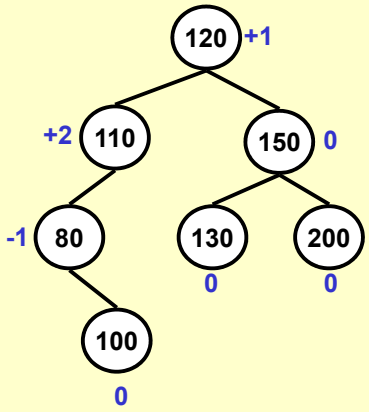

Rotação Dupla DIREITA

Toda vez que uma subárvore fica com um fator:

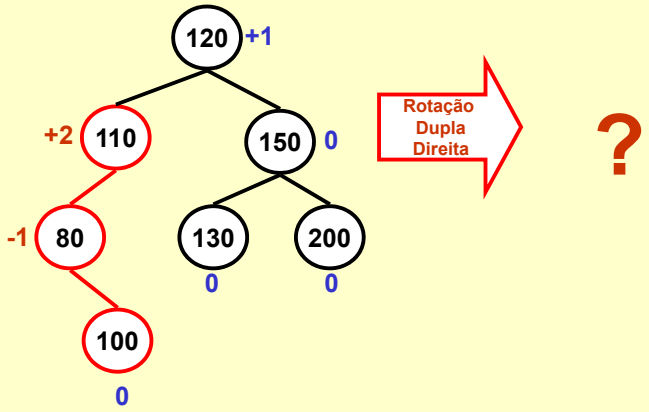
- **positivo e sua subárvore da esquerda tem um fator negativo**

ROTAÇÃO DUPLA À DIREITA

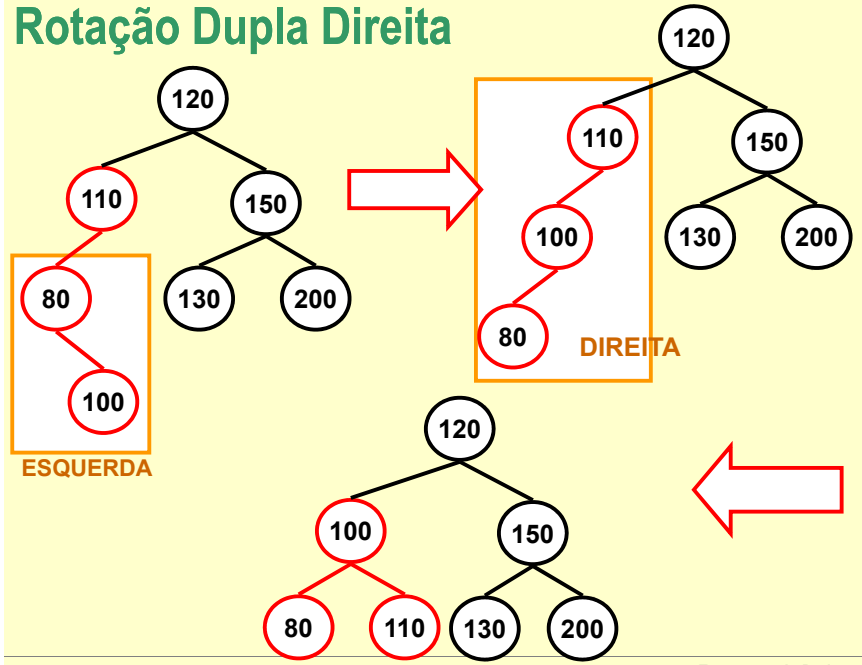
Rotação Dupla Direita



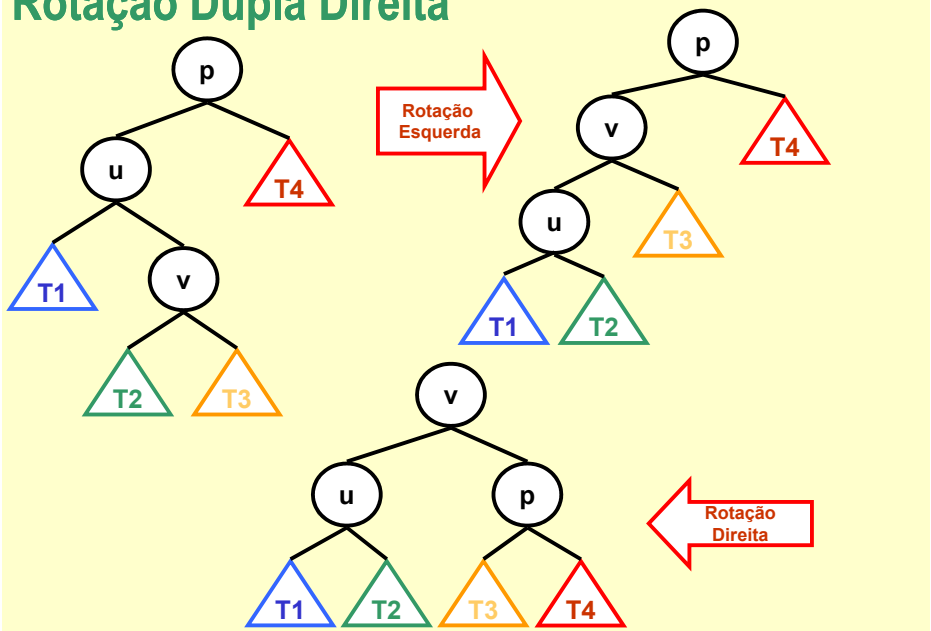
Rotação Dupla Direita



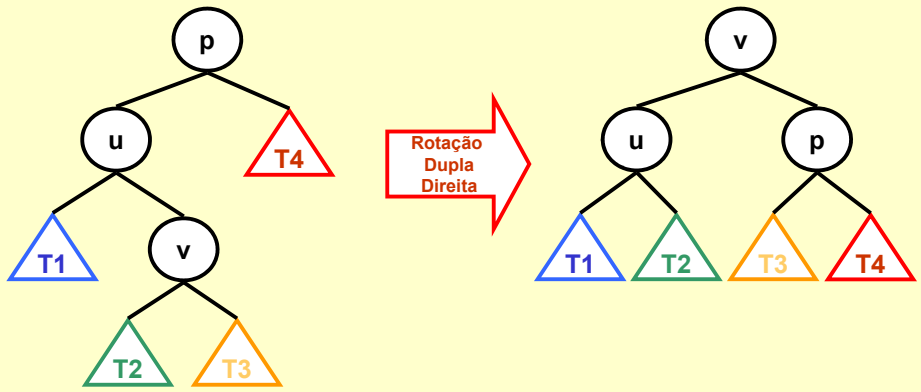
Rotação Dupla Direita



Rotação Dupla Direita



Rotação Dupla Direita



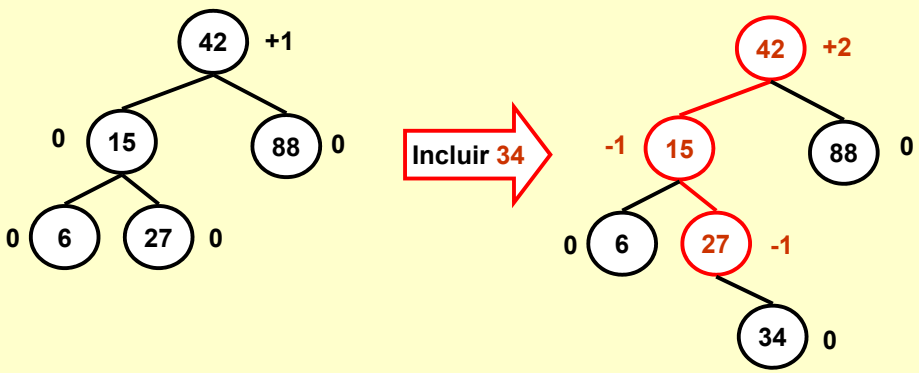
Rotação Dupla Direita

```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

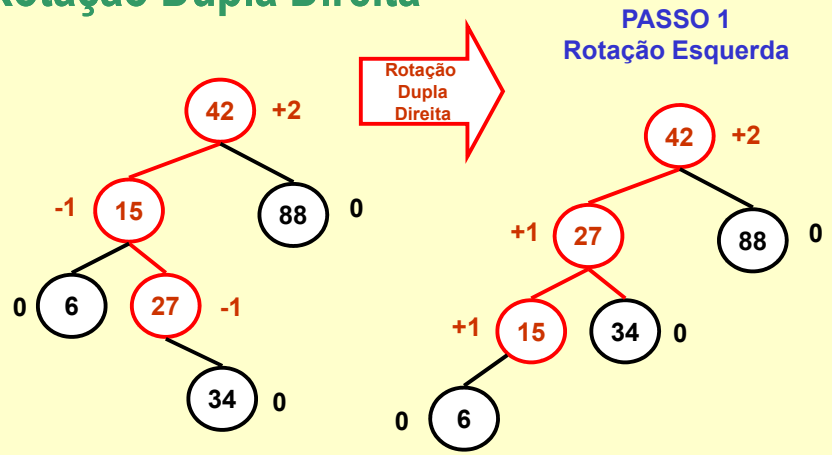
```
pNodoA* rotacao_dupla_direita (pNodoA* pt){
   pNodoA *ptu, *ptv;

   ptu = pt->esq;
   ptv = ptu->dir;
   ptu->dir = ptv->esq;
   ptv->esq = ptu;
   pt->esq = ptv->dir;
   ptv->dir = pt;
   if (ptv->FB == 1)   pt->FB = -1;
      else pt->FB = 0;
   if (ptv->FB == -1)  ptu->FB = 1;
      else ptu->FB = 0;
   pt = ptv;
   return pt;
}
```

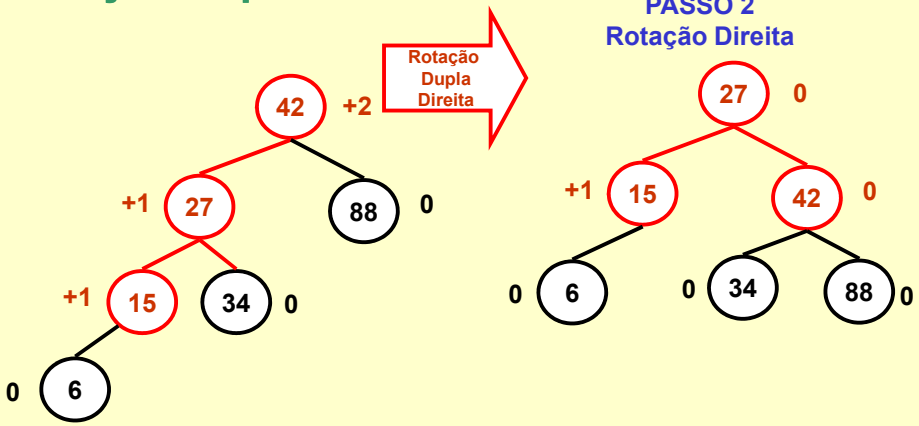
Rotação Dupla Direita



Rotação Dupla Direita



Rotação Dupla Direita



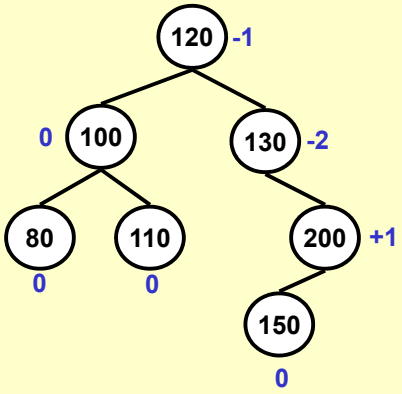
Rotação Dupla ESQUERDA

Toda vez que uma subárvore fica com um fator:

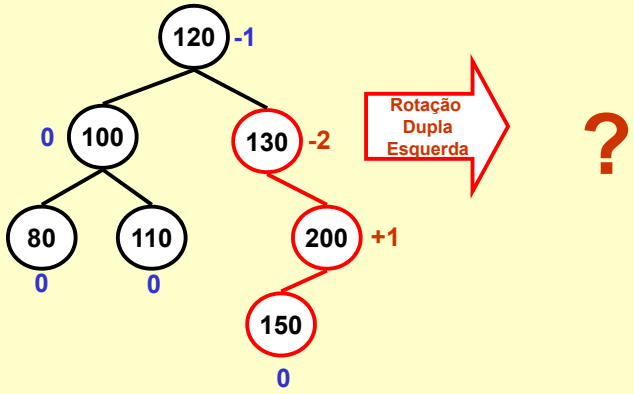
- **negativo** e sua subárvore da direita tem um fator **positivo**

ROTAÇÃO DUPLA À ESQUERDA

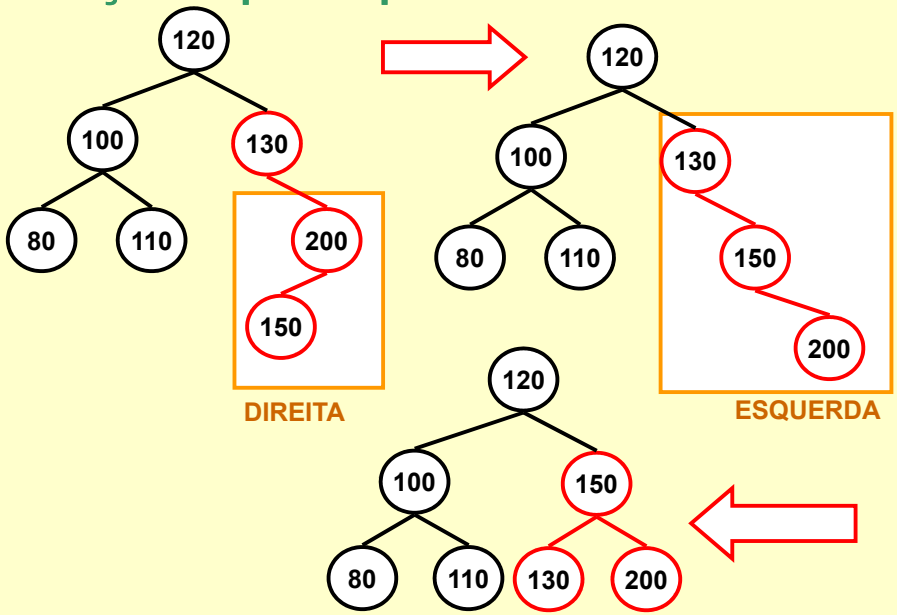
Rotação Dupla Esquerda



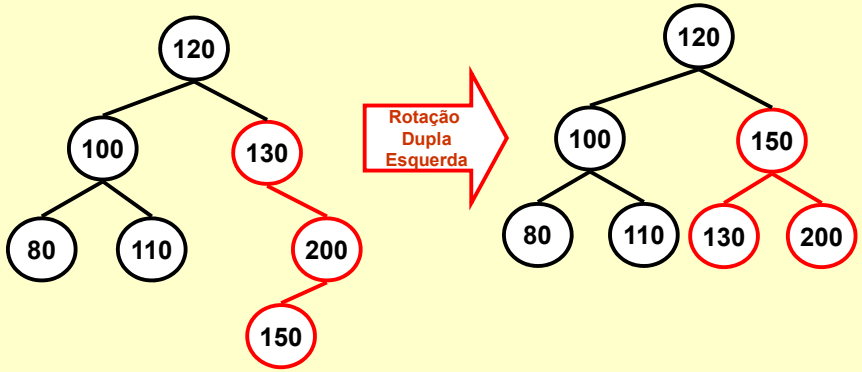
Rotação Dupla Esquerda



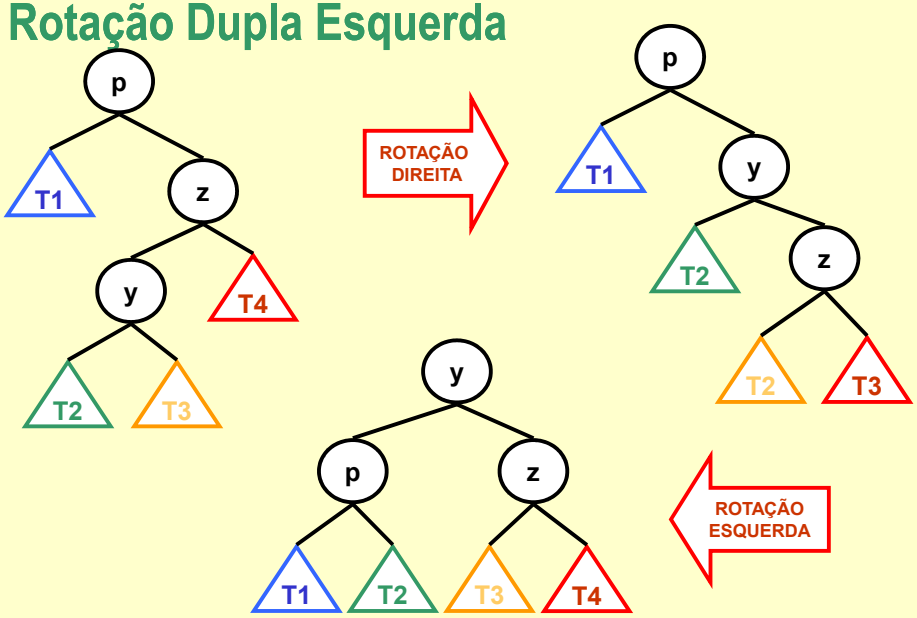
Rotação Dupla Esquerda



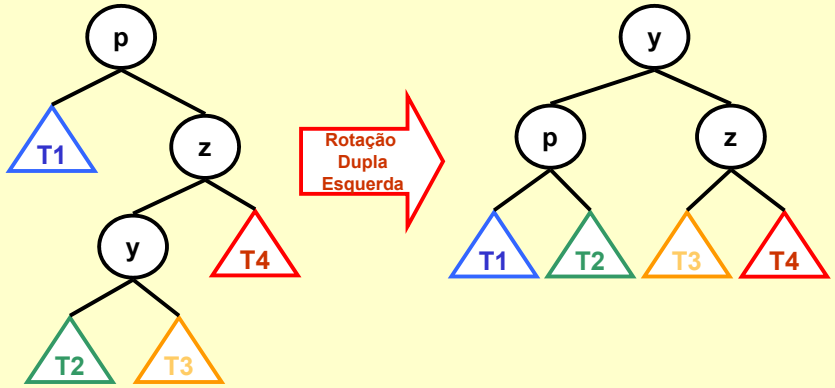
Rotação Dupla Esquerda



Rotação Dupla Esquerda



Rotação Dupla Esquerda



Rotação Dupla Esquerda

```
struct TNodoA{
  TipoInfo info;
  int FB;
  struct TNodoA *esq;
  struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
pNodoA rotacao_dupla_esquerda (pNodoA *pt){
  pNodoA *ptu, *ptv;

  ptu = pt->dir;
  ptv = ptu->esq;
  ptu->esq = ptv->dir;
  ptv->dir = ptu;
  pt->dir = ptv->esq;
  ptv->esq = pt;
  if (ptv->FB == -1) pt->FB = 1;
    else pt->FB = 0;
  if (ptv->FB == 1) ptu->FB = -1;
    else ptu->FB = 0;
  pt = ptv;
  return pt;
}
```


Exercícios

- Inserir em AVL, refazendo a árvore quando tiver rotação e anotando as rotações realizadas:

  –50, 40, 30, 45, 47, 55, 56, 1, 2, 3

- *Na inserção de quais elementos será necessário fazer rotação?*

Inserção de nodos em árvores AVL
Alguns Problemas

- Percorre-se a árvore verificando se a chave já existe ou não
  - Em caso positivo, encerra a tentativa de inserção
  - Caso contrário, a busca encontra o local correto de inserção do novo nó
- Verifica-se se a inclusão tornará a árvore desbalanceada
  - Em caso negativo, o processo termina
  - Caso contrário, deve-se efetuar o balanceamento da árvore
- Descobre-se qual a operação de rotação a ser executada
- Executa-se a rotação

Inserção de nodos em árvores AVL

- Como saber se a árvore está balanceada?
  - Verificando se existe um nó "desregulado"
- Como saber se um nó está desregulado?
  - Determina-se as alturas de suas sub-árvores e subtrai-se uma da outra
- Procedimento muito lento!
- Como ser mais eficiente?
  - Para cada nó **v** de uma árvore, armazena-se uma variável balanço, onde

```
balanço(v) = altura(v↑.esq)- altura(v↑.dir)
```


Inserção de nodos em árvores AVL
Alguns problemas...

- Que valores são possíveis para balanço?
  - -1, 0, 1
- De novo, como ser eficiente no cálculo do balanço?
  - Dado **q**, como o nodo inserido.
  - Se **q** pertencer à sub-árvore esquerda de **v** e essa inclusão resultar em aumento na altura da sub-árvore, então
    balanço(v) := balanço(v) + 1
    - Se **balanço(v) = 2**, então **v** está desregulado
  - Se **q** pertencer à sub-árvore direita de **v** e essa inclusão resultar em aumento na altura da sub-árvore, então
    balanço(v) := balanço(v) - 1
    - Se **balanço(v) = -2**, então **v** está desregulado

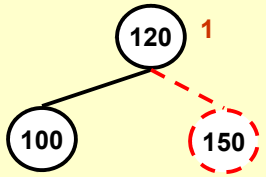
Inserção de nodos em árvores AVL
Alguns problemas...

- Mas, quando é que a inclusão de **q** causa aumento na altura da sub-árvore **v**?
- Suponha que **q** seja incluído na sub-árvore à esquerda de **v**.
- Para **q** incluído na sub-árvore à direita, considera-se o caso simétrico.

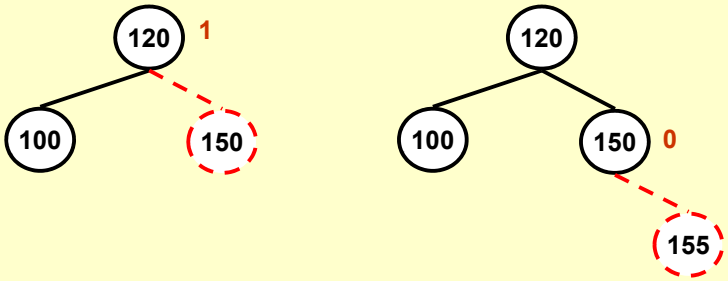
Inserção de nodos em árvores AVL
INSERÇÃO A DIREITA

- Se, antes da inclusão:
  - **Balanço(v)= 1**, então **Balanço(v) se tornará 0**
    - altura da árvore não foi alterada
    - Por consequência, altura dos outros nós no caminho até a raiz, não se altera também.
  - **Balanço(v)= 0**, então **Balanço(v) se tornará -1**
    - altura da árvore foi modificada
    - Por consequência, altura dos outros nós no caminho até a raiz, pode ter sido alterada também.
    - Repetir o processo (recursivamente), com **v** substituído por seu pai.
  - **Balanço(v)= -1**, então **Balanço(v) se tornará -2**
    - altura da árvore foi modificada e o nó está desregulado
    - Rotação correta deve ser empregada.
    - Como a árvore será redesenhada, não é necessário verificar os outros nós.

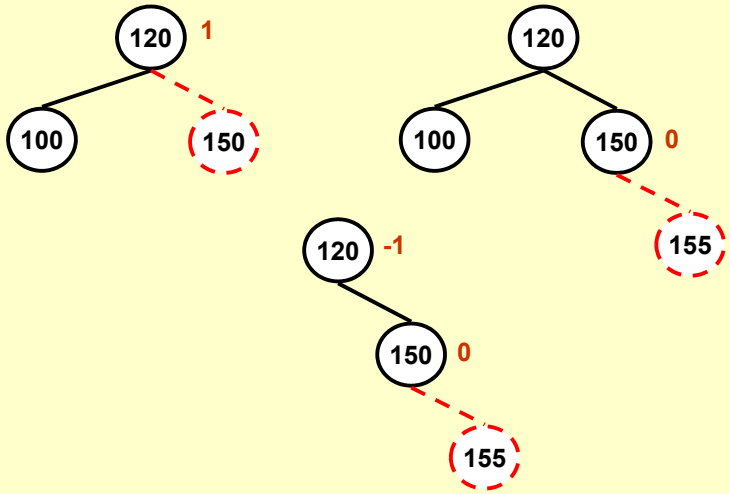
Inserção de nodos em árvores AVL
INSERÇÃO A DIREITA



Inserção de nodos em árvores AVL
INSERÇÃO A DIREITA



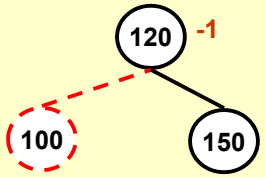
Inserção de nodos em árvores AVL
INSERÇÃO A DIREITA



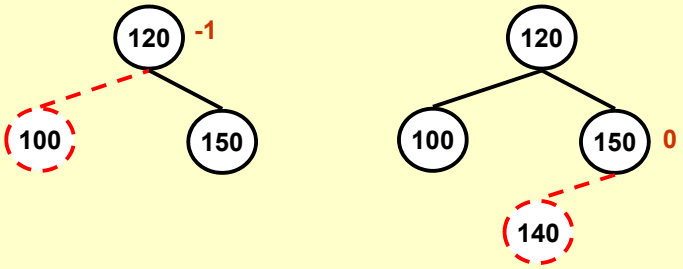
Inserção de nodos em árvores AVL
INSERÇÃO A ESQUERDA

- Se, antes da inclusão:
  - **Balanço(v)= - 1**, então **Balanço(v) se tornará 0**
    - altura da árvore não foi alterada
    - Por consequência, altura dos outros nós no caminho até a raiz, não se altera também.
  - **Balanço(v)= 0**, então **Balanço(v) se tornará 1**
    - altura da árvore foi modificada
    - Por consequência, altura dos outros nós no caminho até a raiz, pode ter sido alterada também.
    - Repetir o processo (recursivamente), com **v** substituído por seu pai.
  - **Balanço(v)= 1**, então **Balanço(v) se tornará 2**
    - altura da árvore foi modificada e o nó está desregulado
    - Rotação correta deve ser empregada.
    - Como a árvore será redesenhada, não é necessário verificar os outros nós.

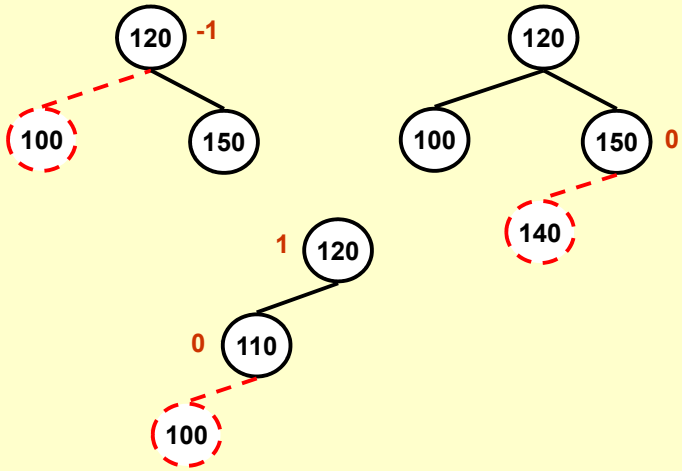
Inserção de nodos em árvores AVL
INSERÇÃO A ESQUERDA



Inserção de nodos em árvores AVL
INSERÇÃO A ESQUERDA



Inserção de nodos em árvores AVL
INSERÇÃO A ESQUERDA



Inserção de nodos em árvores AVL

```
pNodoA* InsereAVL (pNodoA *a, TipoInfo x, int *ok){
/* Insere nodo em uma árvore AVL, onde A representa a raiz da árvore,
  x, a chave a ser inserida e h a altura da árvore */
    if (a == NULL) {
       a = (pNodoA*) malloc(sizeof(pNodoA));
       a->info = x;
       a->esq = NULL;
       a->dir = NULL;
       a->FB = 0;
         *ok = 1;
    }
    else if (x < a->info) {
              a->esq = InsereAVL(a->esq,x,ok);
          if (ok) {
                switch (a->FB) {
                   case -1:  a->FB = 0; *ok = 0; break;
                   case  0:  a->FB = 1; break;
                   case  1:  a=Caso1(a,ok); break;
                }
          }
    }
```

```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
else {
        a->dir = InsereAVL(a->dir,x,ok);
          if (ok) {
            switch (a->FB) {
                case  1:  a->FB = 0; ok = 0; break;
                case  0:  a->FB = -1; break;
                                case -1:  a = Caso2(a,ok);
break;
             }
            }
       }
   return a;
}
```


Inserção de nodos em árvores AVL

```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
pNodoA* Caso1 (pNodoA *a , int *ok)
{
   pNodoA *ptu;

      ptu = a->esq;
      if (ptu->FB == 1)  {
        printf("fazendo rotacao direita em %d\n",a->info);
        a = rotacao_direita(a);
      }
      else {
        printf("fazendo rotacao dupla direita em %d\n",a->info);
        a = rotacao_dupla_direita(a);
      }
      a->FB = 0;
      *ok = 0;
       return a;
}
```


Inserção de nodos em árvores AVL

```
struct TNodoA{
   TipoInfo info;
   int FB;
   struct TNodoA *esq;
   struct TNodoA *dir;
};
typedef struct TNodoA pNodoA;
```

```
pNodoA* Caso2 (pNodoA *a , int *ok)
{
   pNodoA *ptu;

     ptu = a->dir;
     if (ptu->FB == -1)    {
      printf("fazendo rotacao esquerda em %d\n",a->info);
      a=rotacao_esquerda(a);
     }
     else {
       printf("fazendo rotacao dupla esquerda em %d\n",a->info);
       a=rotacao_dupla_esquerda(a);
     }
      a->FB = 0;
      *ok = 0;
      return a;
}
```


Remoção de nodos em árvores AVL

- Caso parecido com as inclusões.
- No entanto, nem sempre se consegue solucionar com uma única rotação...

- http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html