ÁRVORES

Prof. Flávio Rogério Uber

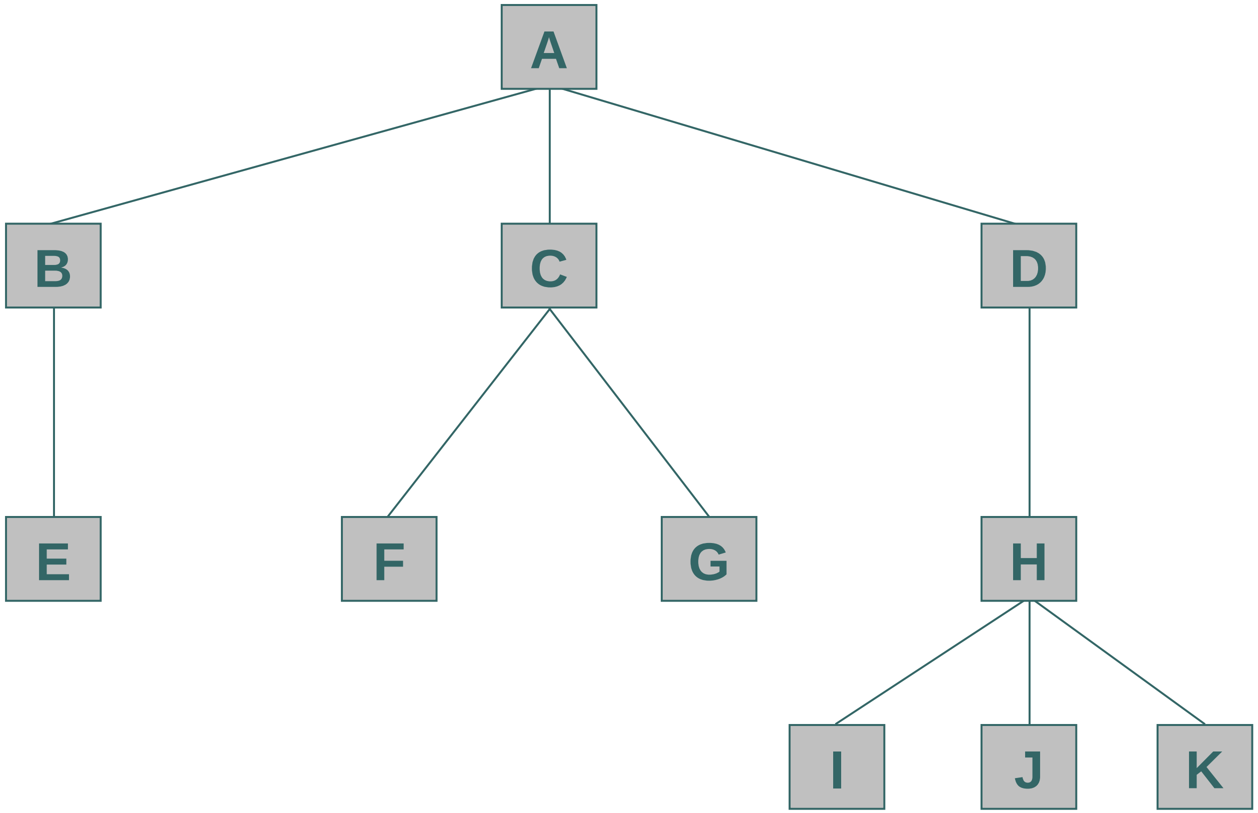
Prof. Yandre Maldonado e Gomes da Costa

Árvores

- Árvores são estruturas de dados que caracterizam uma relação entre os dados que a compõem;
- A relação existente entre os dados (nós) de uma árvore, é uma relação de hierarquia;
- Formalmente, uma árvore é um conjunto finito T de um ou mais nós, tais que:
  - Existe um nó denominado raiz da árvore;
  - Os demais nós formam $m \geq 0$ conjuntos separados T_1, T_2, ..., T_m, onde cada um destes conjuntos é uma árvore. As árvores T_i $(1 \leq i \leq m)$ recebem a denominação de subárvores.

Árvores

- Esquematicamente, uma árvore pode ser representada da seguinte forma:



Árvores

- Terminologia:
  - No exemplo anterior, os quadrados representam os nós da árvore, cuja **raiz** é o nó ‘A’;
  - Pela definição de árvore, cada nó da árvore é raiz de uma subárvore. O número de subárvores de um nó é o **grau** deste nó;
  - O **grau de uma árvore** é igual ao grau do nó de maior grau pertencente à mesma;
  - Um nó de grau zero é chamado de **folha** ou **nó terminal**;
  - O **nível** do nó é definido como sendo o igual ao número de nós que o separam da raiz;
  - A **altura** de uma árvore é definida como sendo o nível mais alto da árvore;
  - Um conjunto de zero ou mais árvores disjuntas é chamado de **floresta**;

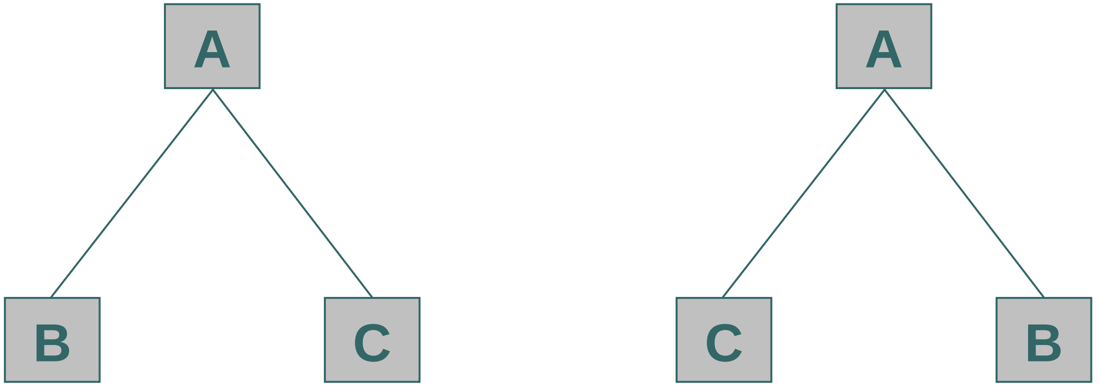
Árvores

- Em relação a árvore descrita anteriormente, pode-se observar:

Nodo	Grau	Nível	Observações
A	3	0	Raiz da Árvore
B	1	1	
C	2	1	
D	1	1	
E	0	2	Nó folha
F	0	2	Nó folha
G	0	2	Nó folha
H	3	2	
I	0	3	Nó folha
J	0	3	Nó folha
K	0	3	Nó folha

Árvores

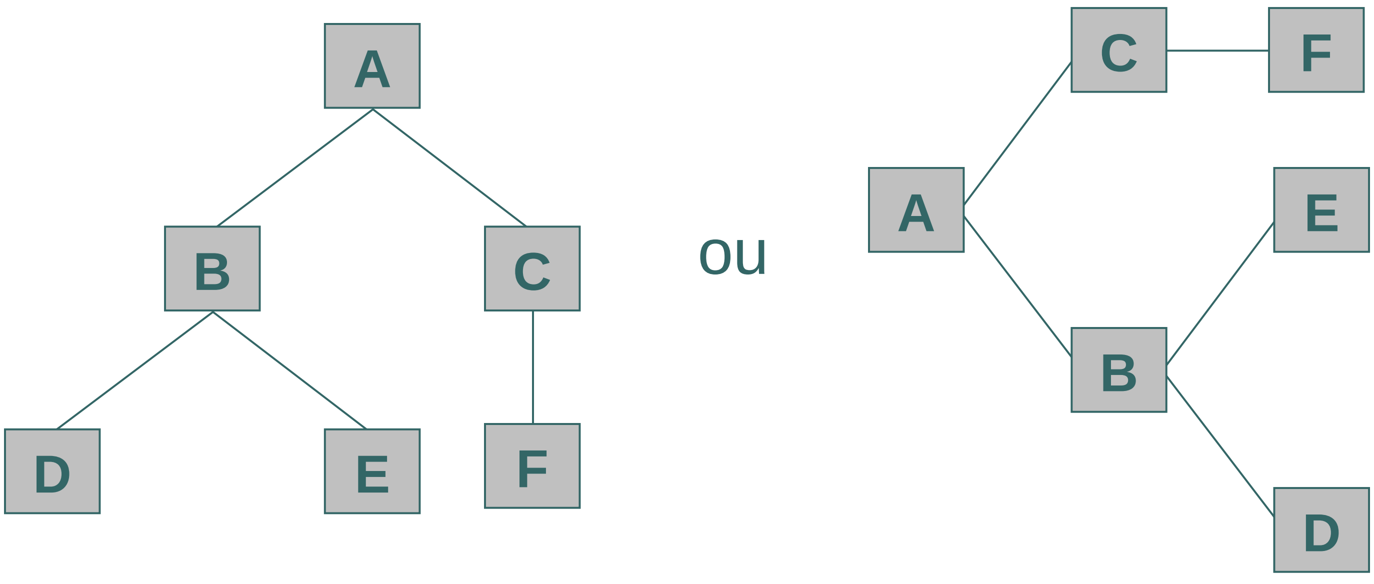
- Quando a ordem das subárvores é significativa, a árvore é chamada de ordenada. Neste caso, há diferença entre as seguintes árvores:



- Entretanto, quando a ordem das subárvores não é relevante, diz-se que a árvore é orientada, uma vez que apenas a orientação dos nós é importante (neste caso as duas árvores mostradas acima são iguais);

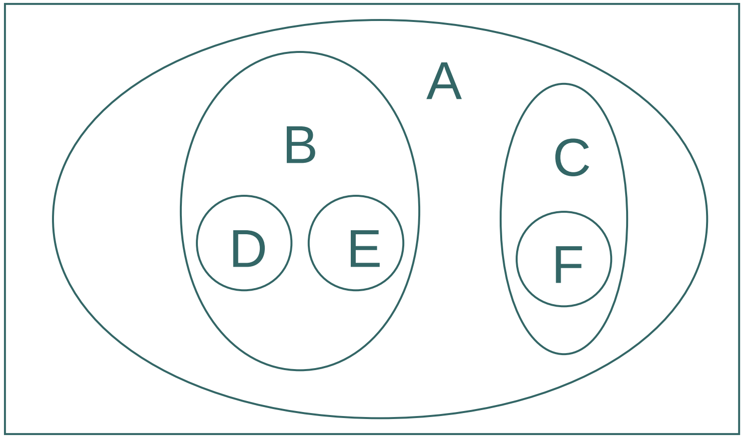
Árvores

- A raiz de uma árvore é chamada de **pai** das raízes de suas subárvores. As raízes das subárvores de um nó são chamadas de **irmãos** que, por sua vez, são **filhos** de seu nó pai.
- Formas de Representação de Árvores:
  - Árvore convencional (grafos)



Árvores

- Conjuntos Aninhados ou Diagramas de Venn:



- Parênteses Aninhados:

(A (B (D) (E)) (C (F)))

Árvores

- Barramento ou Tabelas:

A ————————————————

    B ——————————————

        D ————————————

        E ————————————

    C ——————————————

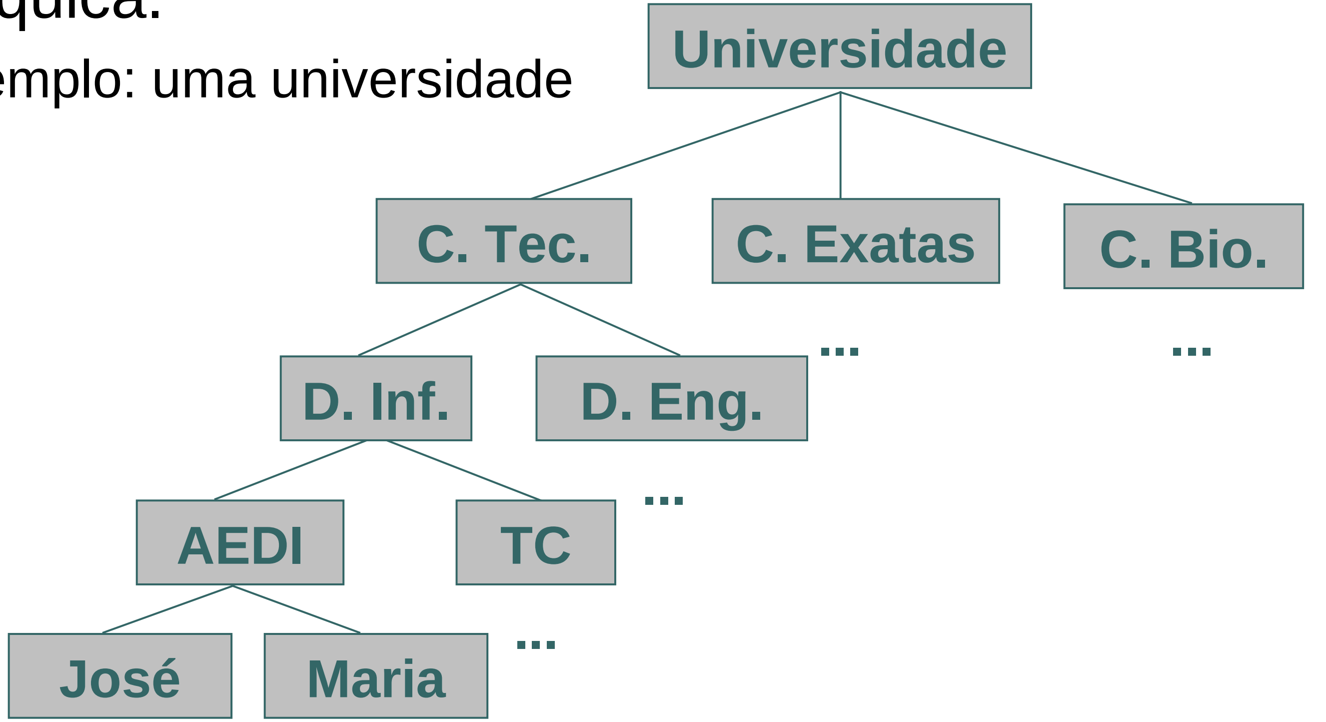
        F ————————————

- Notação Decimal (DEWEY):

  1.A, 1.1.B, 1.1.1.D, 1.1.2.E, 1.2.C, 1.2.1.F.

Árvores

- Aplicações de árvores:
  - Representações genealógicas;
  - Representação de objetos que possuem relação hierárquica.
    - Exemplo: uma universidade

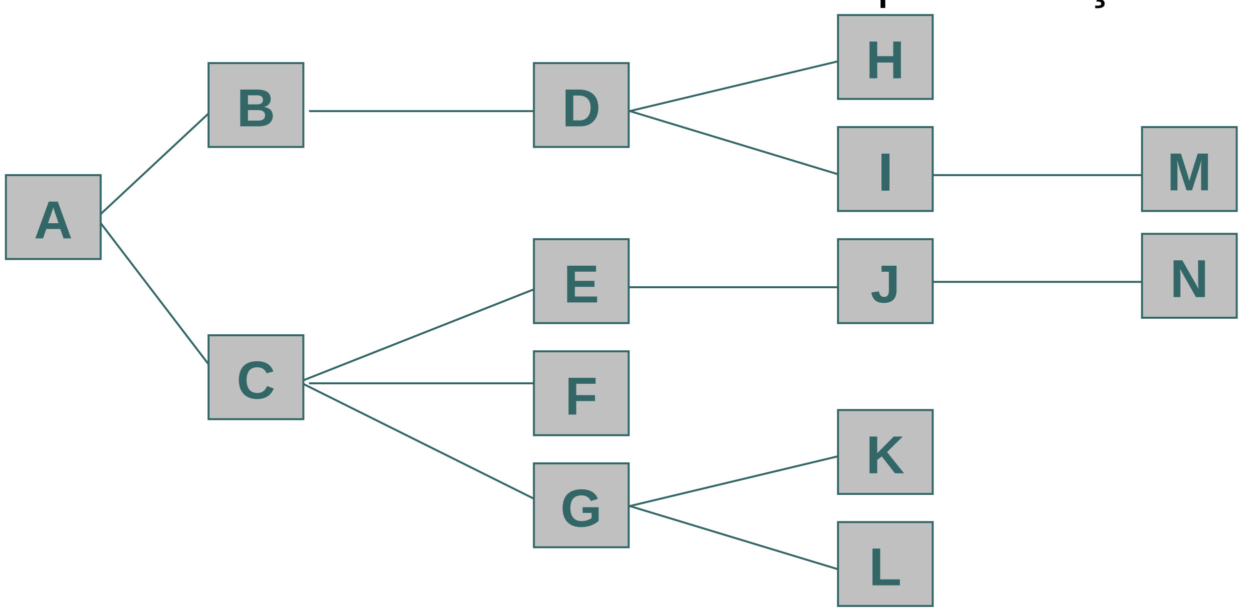


Árvores

- Aplicações:
  - Índices de arquivos;
  - Árvores genealógicas ou hereditárias;
  - Organização de empresa (organograma);
  - Avaliação de expressões;
  - Árvores de decisão;
  - Algoritmos e Classificação

Árvores

- Exercício: dada a seguinte árvore, encontre:
  - A raiz da árvore;
  - Todos os nós folha;
  - O grau e o nível de cada nó;
  - A altura da árvore;
  - Todas as relações entre nós (irmão, pai, filho);
  - Descreva a árvore com todas as representações estudadas;



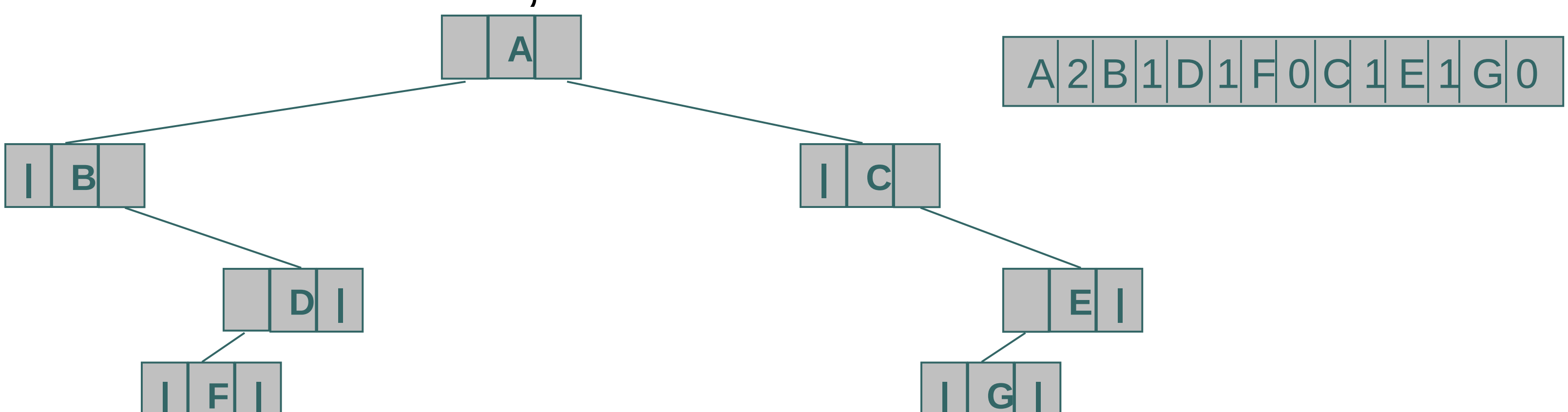
Árvores Binárias

"Conjunto finito de nós que, ou é vazio, ou consiste de uma raiz ligando até duas outras árvores binárias."

- São árvores onde o grau de cada nó é menor ou igual a dois;
- As subárvores de cada nó são chamadas subárvore esquerda e subárvore direita;
- Assim, se um nó possuir apenas uma subárvore, ela deve ser estabelecida como direita ou esquerda;
- Uma árvore binária pode ser vazia, isto é, não possuir nenhum nó;

Árvores Binárias

- Alocação:
  - Por adjacência:
    - Nós representados seqüencialmente na memória;
    - Esta alocação não é conveniente na maioria dos casos, pois dificulta a manipulação da estrutura;

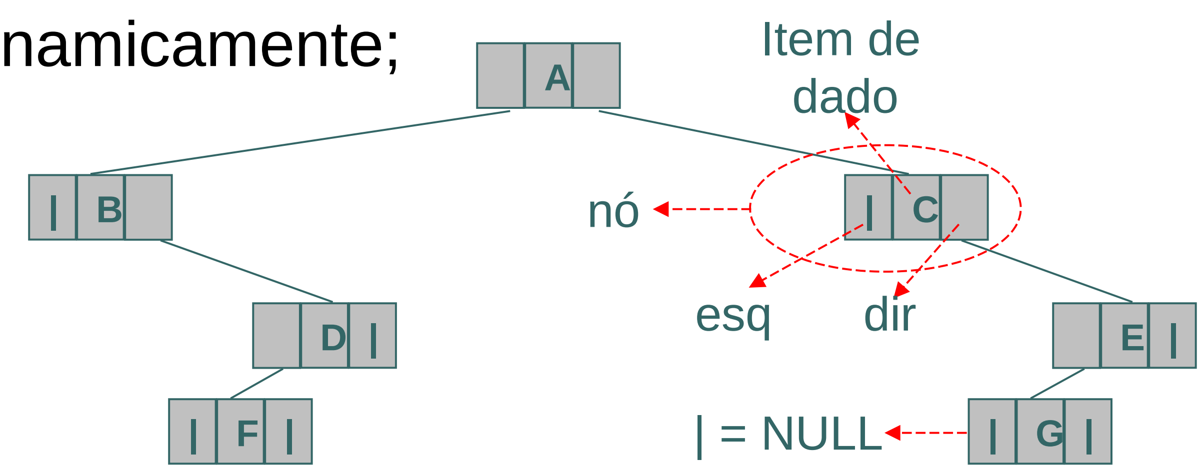


Árvores Binárias

- Alocação:
  - Encadeada:
    - Mais adequada;
    - Permite melhor manipulação dos dados com diversas ordens de acesso aos nós;
    - Os nós são alocados dinamicamente;

```
typedef struct
 {
  int chave;
  //Outros Campos
 } Registro;

struct Nodo
 {
  Registro Reg;
  Nodo* Dir;
  Nodo* Esq;
 };
```



Árvores Binárias

- Caminhamento em Árvores Binárias:
  - É a forma de percorrer todos os nós da árvore, com o objetivo de consultar ou alterar suas informações;
  - Existem vários métodos de tal forma que cada nó seja “visitado” apenas uma vez;
  - Um completo percurso da árvore nos dá um arranjo linear sobre os nós;
  - São três os principais caminhos utilizados para percorrer uma árvore binária: visitar os nós em ordem **pré-fixada**, **pós-fixada**, ou **in-fixa** (in-ordem).

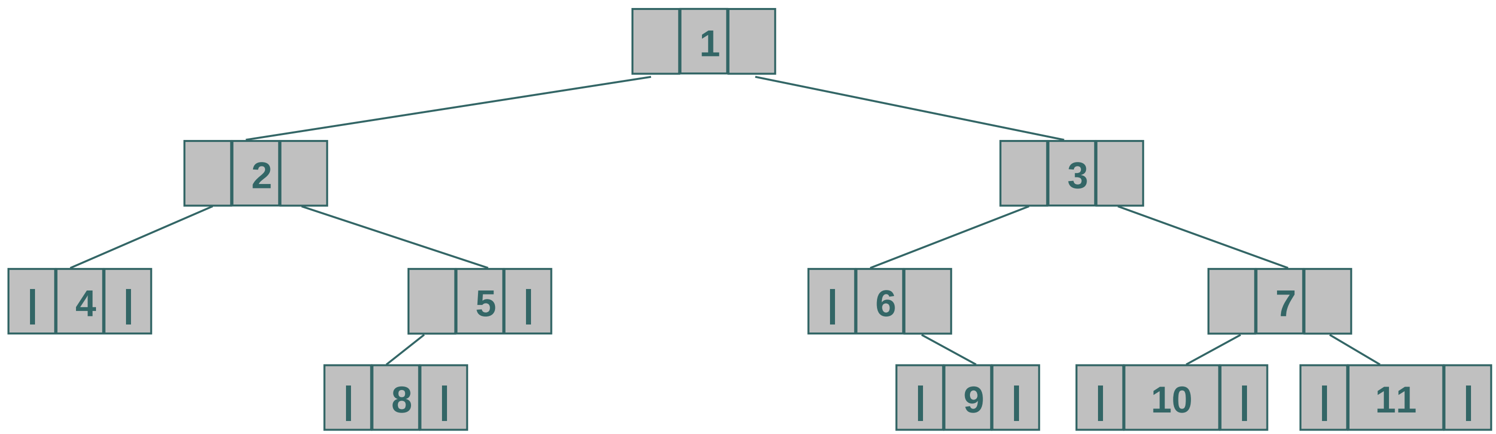
Árvores Binárias

- Formas de caminhar:
  - Pré-ordem: <u>R</u>ED
    - Visitar a raiz;
    - Percorrer a subárvore esquerda;
    - Percorrer a subárvore direita;
  - In-ordem: E<u>R</u>D (percorre as chaves em ordem crescente)
    - Percorrer a subárvore esquerda;
    - Visitar a raiz;
    - Percorrer a subárvore direita;
  - Pós-ordem: ED<u>R</u> (ou forma polonesa)
    - Percorrer a subárvore esquerda;
    - Percorrer a subárvore direita;
    - Visitar a raiz;

Obs.: o termo visita indica alguma manipulação sobre o nó.

Árvores Binárias

- Exemplo: dada a seguinte árvore, verifique a seqüência de nos percorridos segundo as três formas de caminhar sobre árvores binárias.



Pré-ordem: 1, 2, 4, 5, 8, 3, 6, 9, 7, 10, 11

In-ordem: 4, 2, 8, 5, 1, 6, 9, 3, 10, 7, 11

Pós-ordem: 4, 8, 5, 2, 9, 6, 10, 11, 7, 3, 1 (polonesa)

Árvore Binária

- Algoritmos de travessia em árvores binárias
  - observe que os procedimentos são recursivos, devido à natureza recursiva da estrutura (árvore).

```
void pre_ordem (Nodo* T)
{
 if (T!=NULL)
  {
   printf("Item: %d",T->Reg.chave);
   pre_ordem (T->Esq);
   pre_ordem (T->Dir);
  }
}
```

```
void in_ordem (Nodo* T)
{
 if (T!=NULL)
  {
   in_ordem (T->Esq);
   printf("Item: %d",T->Reg.chave);
   in_ordem (T->Dir);
  }
}
```

Árvores Binárias

```
void pos_ordem (Nodo* T)
 {
  if (T!=NULL)
   {
    pos_ordem (T->Esq);
    pos_ordem (T->Dir);
    printf("Item: %d",T->Reg.chave);
   }
 }
```

Exercício

- Escreva uma função que calcule o número de nós de uma árvore binária.
- Escreva uma função que imprima (seguindo a travessia in-ordem) o conteúdo das folhas de uma árvore binária.

Árvores Binárias de Busca

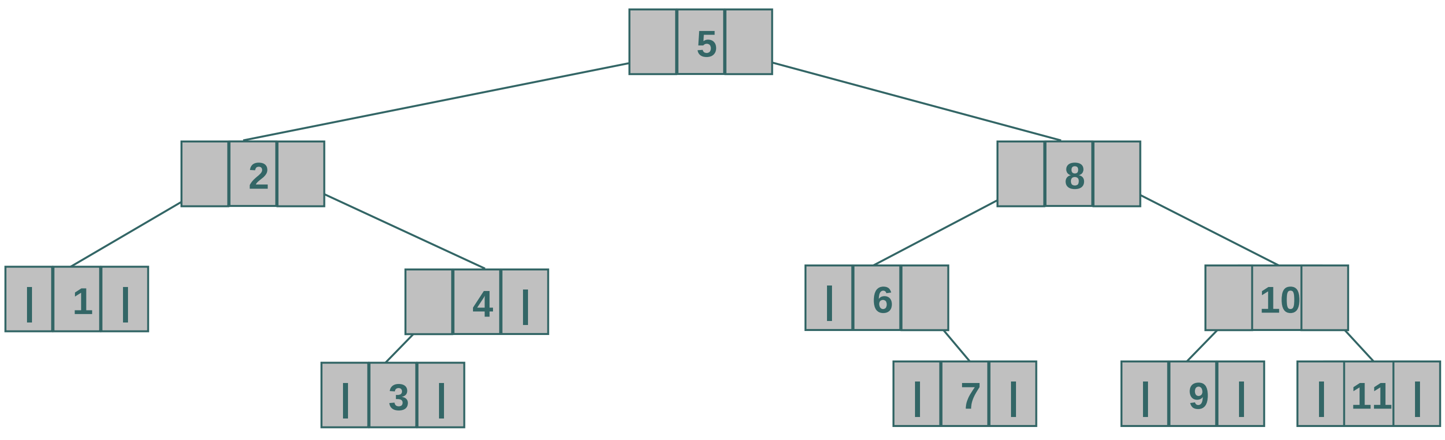
- Árvore Binária de Busca ou Árvore de Pesquisa:
  - Uma ABB para um subconjunto *S* é uma árvore binária com rótulos no qual cada vértice *v* está rotulado com elementos $e(v) \in S$ |:
    1. Para cada vértice μ na subárvore Esq de $v \Rightarrow e(\mu) < e(v)$;
    2. Para cada vértice μ na subárvore Dir de $v \Rightarrow e(\mu) > e(v)$;
    3. Para cada elemento $a \in S$, existe exatamente um vértice v | $e(a)=v$.

Árvores Binárias de Busca

- Em resumo, uma árvore binária de pesquisa é uma árvore binária onde cada nó interno possui um registro, tal que:
  - todo registro alocado na sua subárvore esquerda é menor do que o nó pai;
  - e todo registro alocado na subárvore direita é maior do que o nó pai.

Árvores Binárias de Busca

- Exemplo:



Árvores Binárias de Busca

- A estrutura de dados para esta árvore poderia ser dada por:

```
typedef struct
 {
  int chave;
  //Outros Campos
 } Registro;

struct Nodo
 {
  Registro Reg;
  Nodo* Dir;
  Nodo* Esq;
 };
```

Árvores Binárias de Busca

- Operações básicas para uma árvore binária de busca:
  - Inicialização, inserção, remoção (e balanceamento);
- Para isto, é preciso utilizar os processos recursivos de busca da árvore;
  - Procura-se um elemento Y na raiz, se ele não for encontrado deve-se procurá-lo na subárvore esquerda caso ele seja menor que a raiz, ou na subárvore direita se ele for maior que a raiz;
- Nas operações de alteração, remoção e consulta a busca deve ter sucesso, nas operações de inserção a busca deve fracassar;

Árvores Binárias de Busca

```
void insere_elemento (Nodo* T, Registro X)
 {
  if (T==NULL)
   {
    T=(Nodo*)malloc(sizeof(Nodo));
    T->Reg=X;
    T->Esq=NULL;
    T->Dir=NULL;
   }
  else
   if (X.chave<T->Reg.chave)
    insere_elemento (T->Esq, X);
   else
    if (X.chave>T->Reg.chave)
     insere_elemento (T->Dir, X);
     else
      T->Reg=X; //Substitui
 }
```

```
void inicializa_arvore (Nodo* T)
 {
  T=NULL;
 }
```

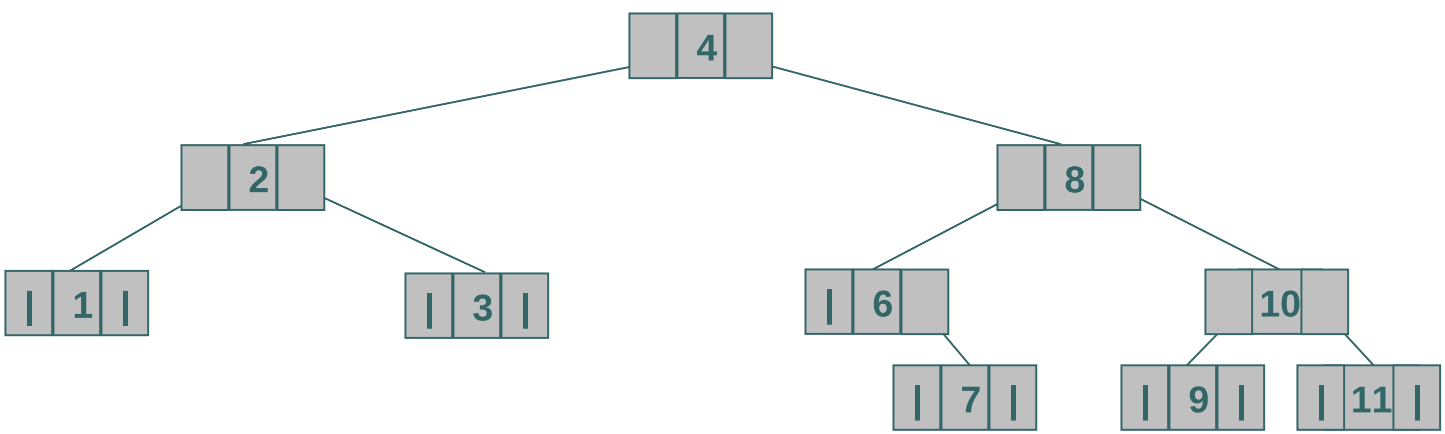
Árvores Binárias de Busca

- Remoção

"Para se criar esta função deve-se fazer uma análise. Pois, se o elemento a ser removido tiver apenas um descendente, a remoção será simples. Mas se o elemento a ser removido tiver dois descendentes, ele deverá ser substituído por aquele que estiver mais a direita em sua subárvore esquerda (maior dos menores); ou por aquele que estiver mais a esquerda em sua subárvore direita (menor dos maiores)."

Árvores Binárias de Busca

- Exemplo: na árvore do slide 24, se removêssemos o nó com chave 5, poderíamos substituí-lo pelo nó com chave 4 (como mostra a figura abaixo), ou pelo nó com chave 6.



Árvores Binárias de Busca

Neste algoritmo foi utilizado o sucessor a esquerda, ou seja, o maior dos menores.

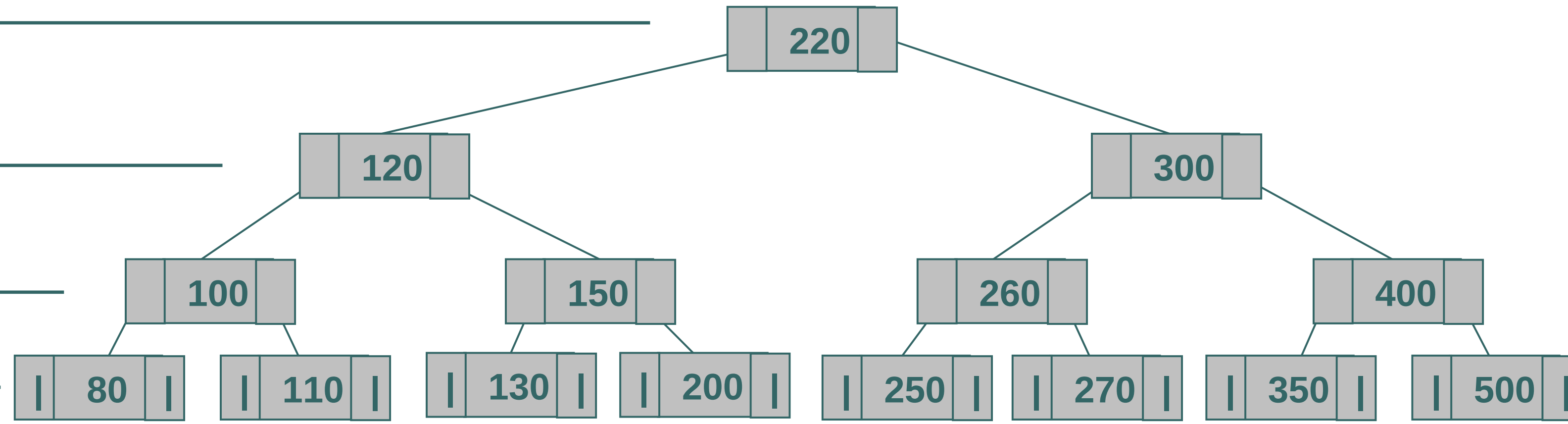
```
Registro Maior (Nodo* Q)
 {
  while (Q->Dir != NULL)
   Q=Q->Dir;
  return(Q->Reg);
 }...
```

```
void Remove_elemento (Nodo* T, Registro X)
 {
  Nodo* A;

  if (T==NULL)
    printf ("Elemento nao encontrado na arvore.");
  else
    if (X.chave<T->Reg.chave)
      Remove_elemento (T->Esq, X);
    else
      if (X.chave>T->Reg.chave)
        Remove_elemento (T->Dir, X);
      else
           if (T->Dir==NULL)
            {
             A=T;
             T=T->Esq;
             free(A);
            }
           else
           if (T->Esq==NULL)
            {
             A=T;
             T=T->Dir;
             free(A);
            }
           else
            {
             T->Reg=Maior(T->Esq);
             Remove_elemento (T->Esq,T->Reg);
            }
 }
```

Árvores Binárias de Busca

- Ordem de complexidade da árvore binária:



4 consultas – 15 chaves

3 consultas – 7 chaves

2 consultas – 3 chaves

1 consulta – 1 chave

Capacidade = 2^N-1
Sendo N o número de níveis da árvore

A Árvore Binária tem complexidade igual a pesquisa binária:
-Melhor caso: 1 consulta;
-Média: $\log_2^{n}$;
-Pior caso: $\log_2^{n}+1$.
Onde n é o número de elementos armazenados na árvore.

Árvores Binárias de Busca

- Balanceamento:
  - Busca uma distribuição equilibrada dos nós;
  - Busca otimizar a consulta;
  - Busca minimizar o número médio de comparações necessário para a localização de uma chave.

Árvores Binárias de Busca

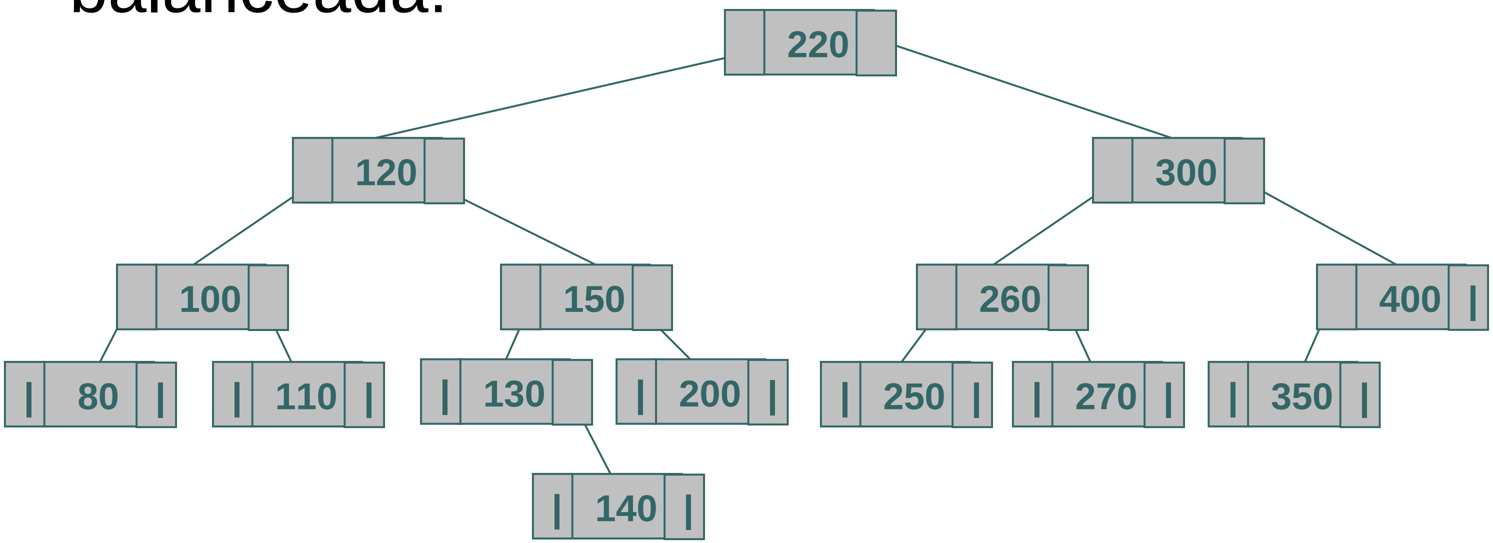
- Balanceamento por altura:
  - Busca-se minimizar a altura da árvore;
- Árvore Completamente Balanceada:
  - Uma árvore é completamente balanceada quando a distância média dos nós até a raiz for mínima;
  - Uma árvore binária é dita completamente balanceada se, para cada nó, o número de nós de suas subárvores diferem de no máximo, 1;
  - Árvore completamente balanceada é a árvore com menor altura para o seu número de nós.

Árvore AVL

- Árvores não completamente balanceadas:
  - Uma árvore balanceada é uma árvore onde a diferença de altura de qualquer subárvore é no máximo 1;
  - O grande esforço exigido para a manutenção de uma árvore completamente balanceada pode não ser compensado pelo ganho de eficiência no processo de busca;
  - Árvore não completamente balanceadas beneficiam o processo de busca, exigindo manutenção do balanceamento pouco onerosa.

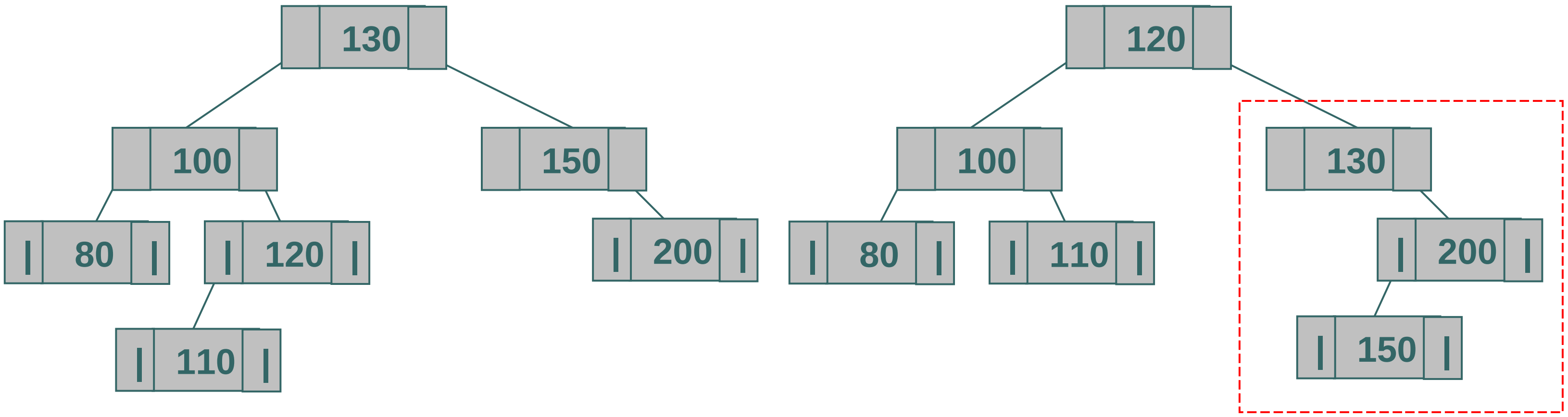
Árvore AVL

- Exemplo de árvore não completamente balanceada:



Neste contexto, destacam-se as árvores **AVL**, concebidas em 1962, por Adel'son-Vel'skii e Landis, caracterizadas pela seguinte propriedade: *para todo nó de uma árvore AVL, a diferença entre as alturas de suas subárvores não excede a uma unidade*.

Árvore AVL



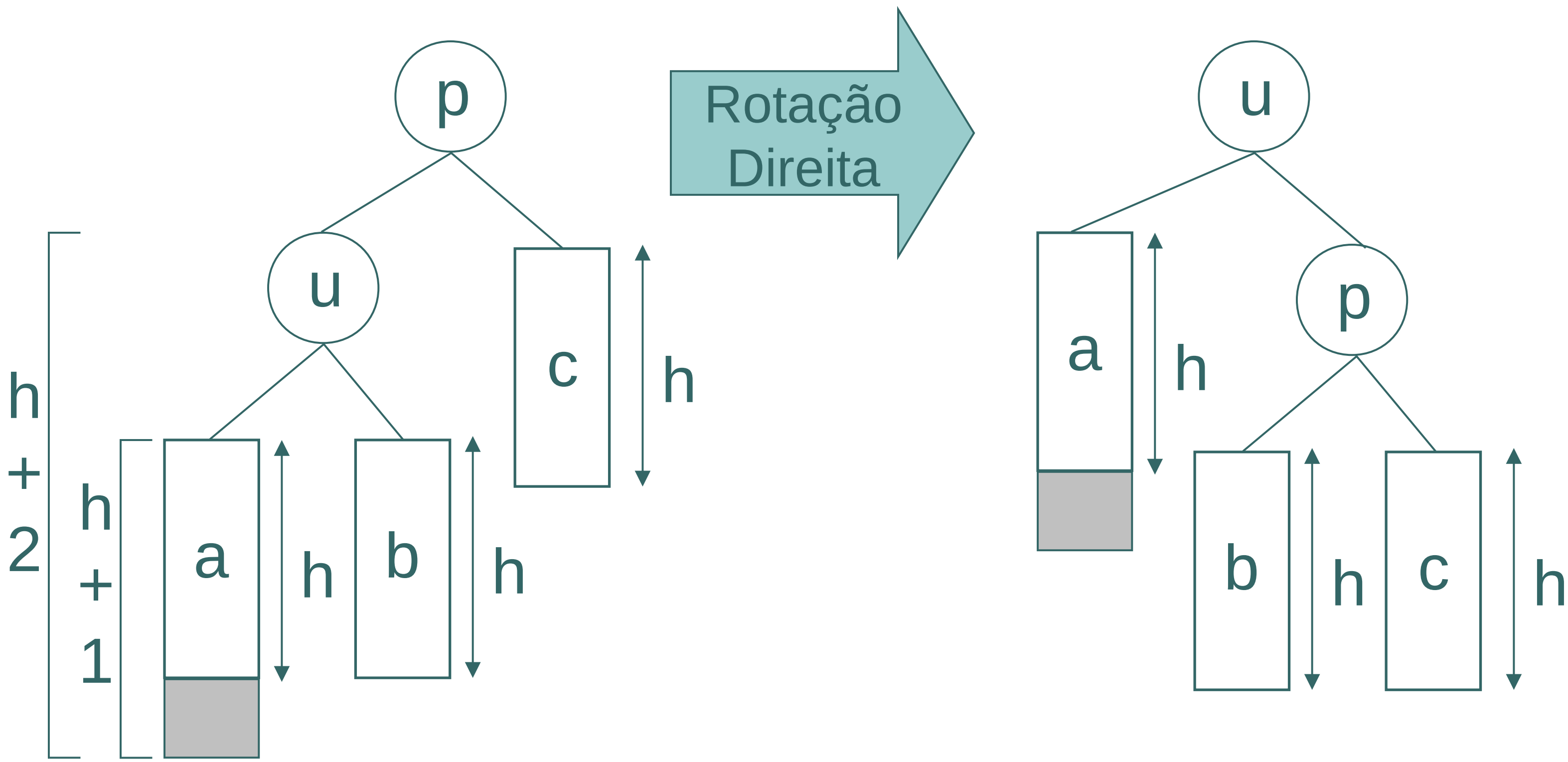
Árvore AVL **Árvore não AVL**

As constantes inserções e remoções de nós de uma árvore podem provocar o desbalanceamento da mesma. Para corrigir este problema em uma árvore AVL, é necessária a aplicação de uma das quatro rotações que serão vistas a seguir.

Árvore AVL

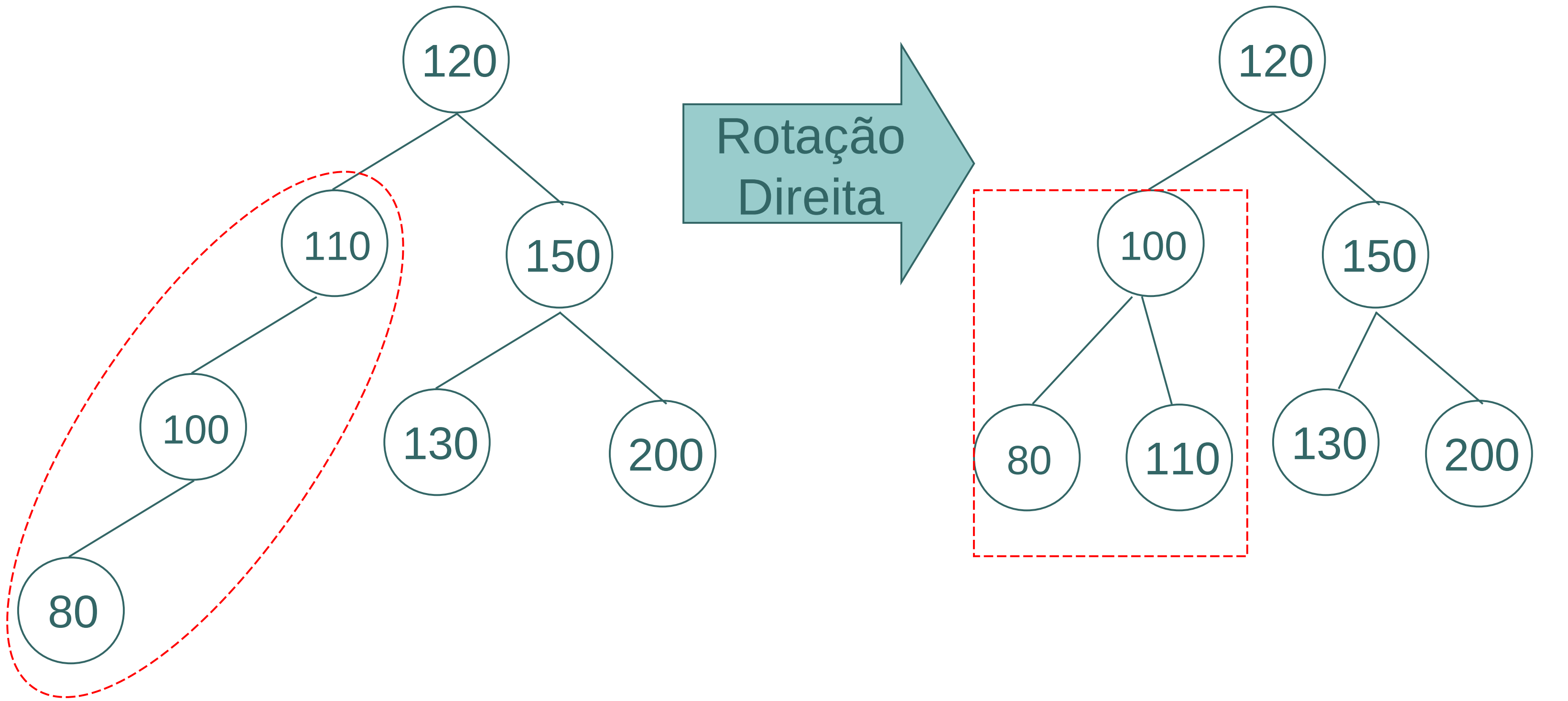
- Rotação Direita:

- u<b<p
- u passa a ser a raiz
-b é pendurada à esquerda de p
- h≥0



Árvore AVL

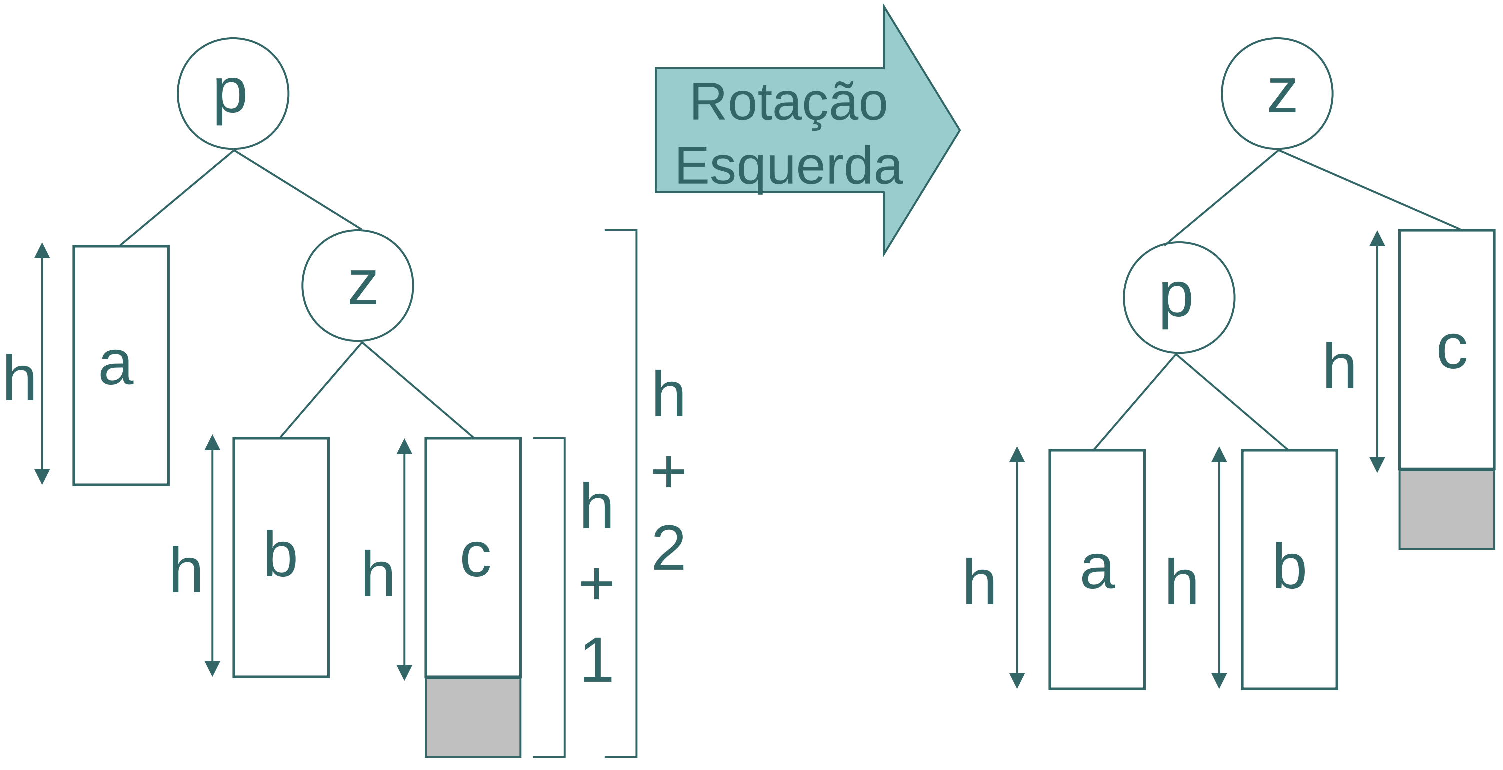
- Exemplo de Rotação Direita:



Árvore AVL

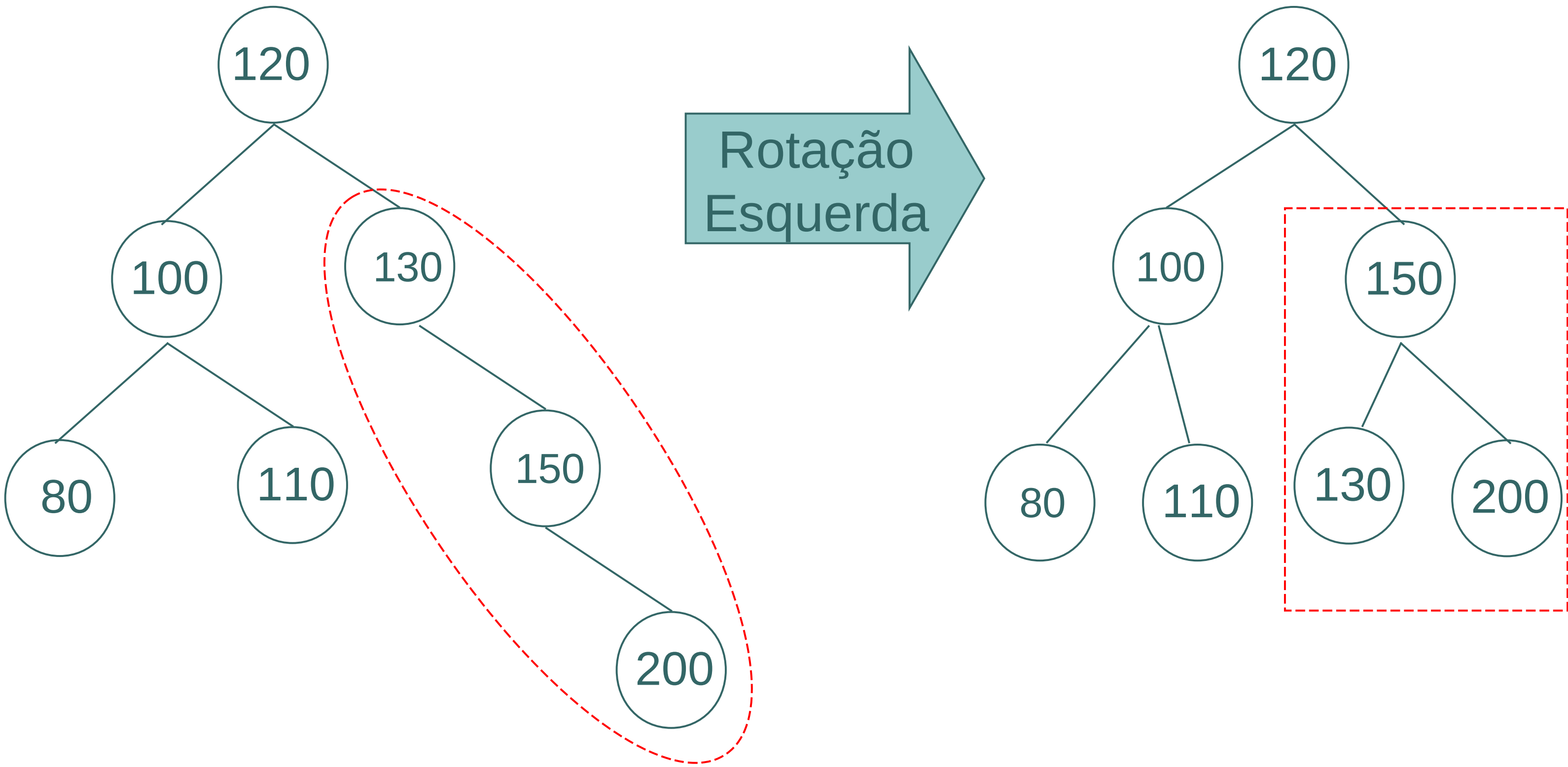
- Rotação Esquerda:

- p<b<z
- z passa a ser a raiz
- b é pendurada à direita de p
- $h \geq 0$



Árvore AVL

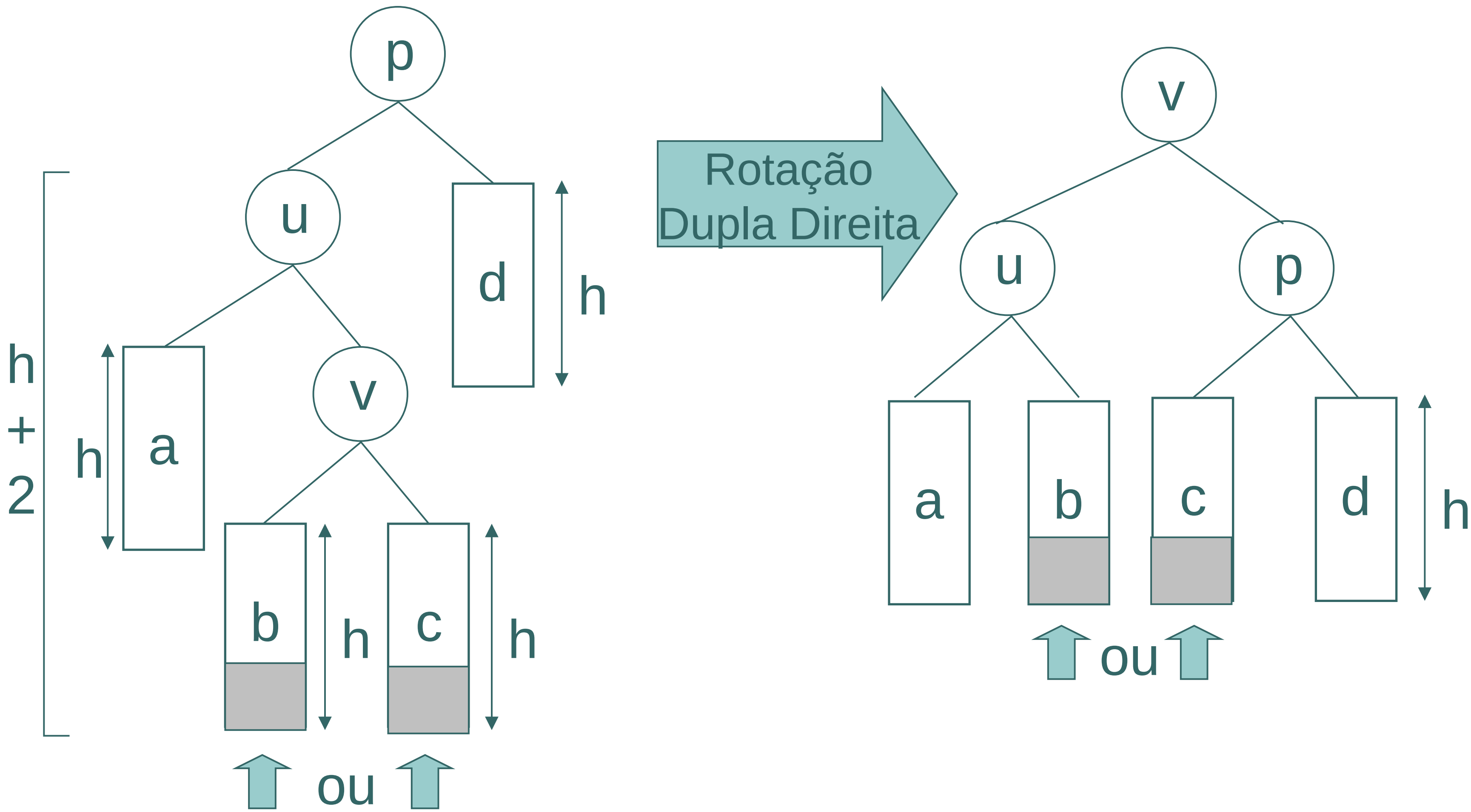
- Exemplo de Rotação Esquerda:



Árvore AVL

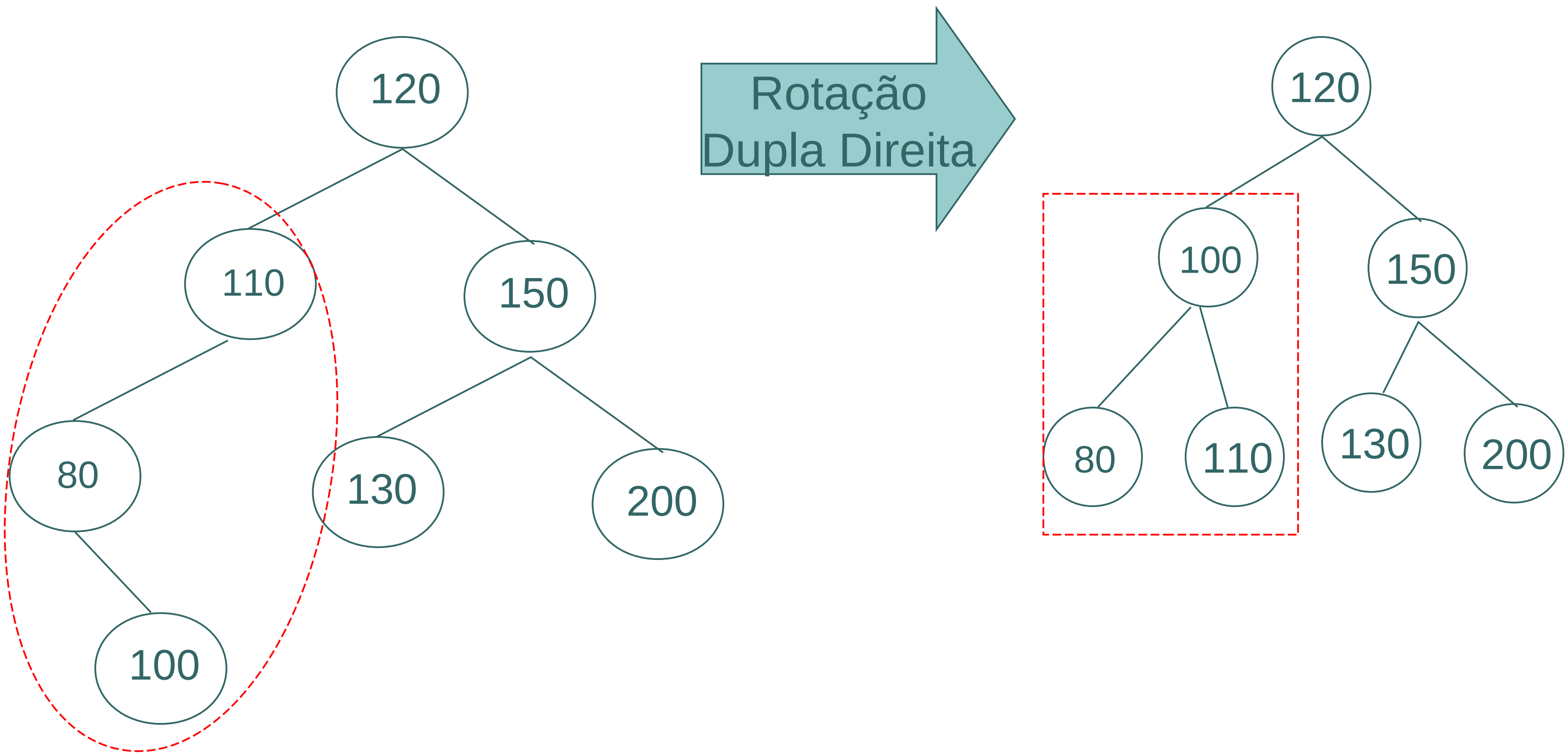
- Rotação Dupla Direita:

- b<v<c
- u<v<p
- v passa a ser a raiz
- -h≥0



Árvore AVL

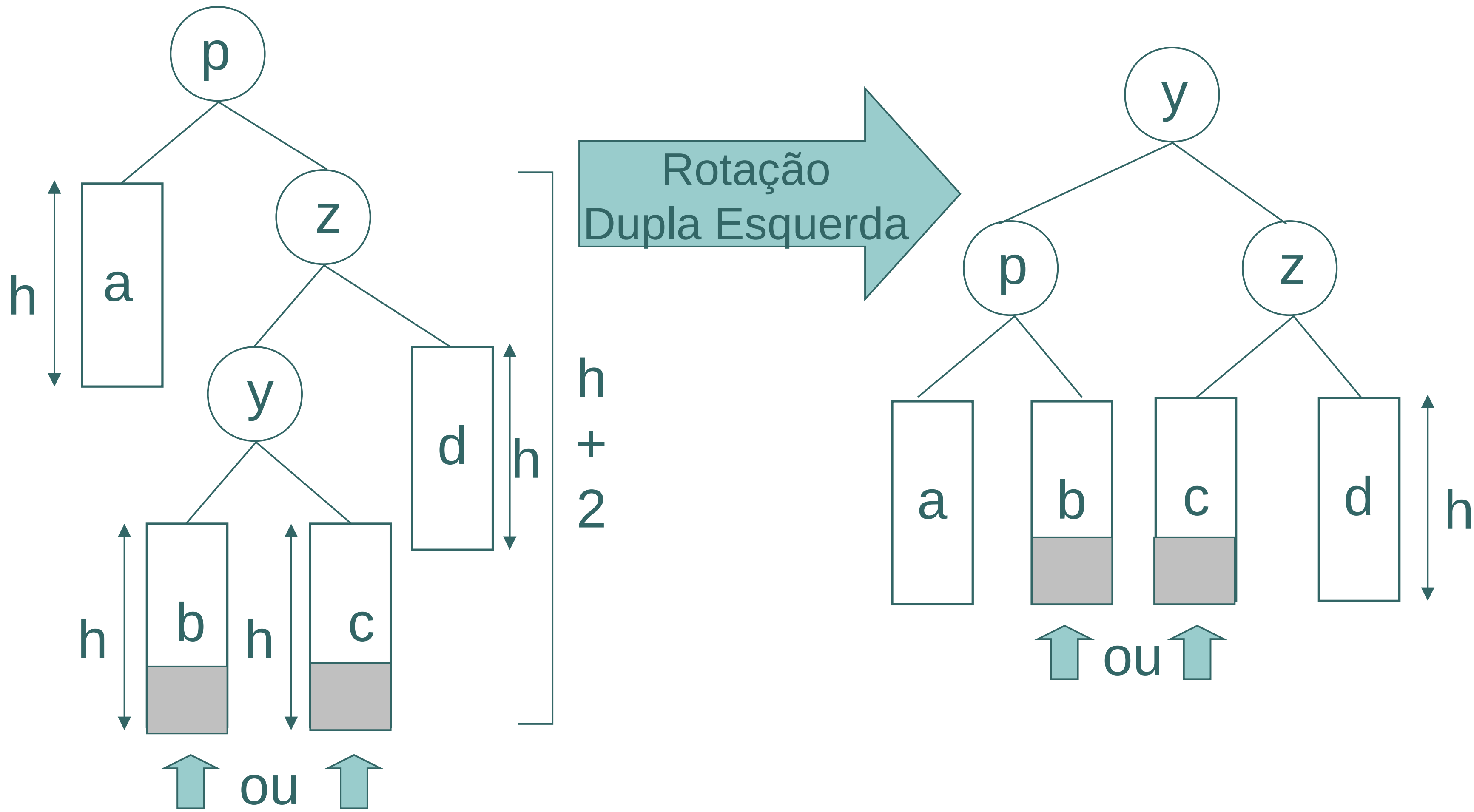
- Exemplo de Rotação Dupla Direita:



Árvore AVL

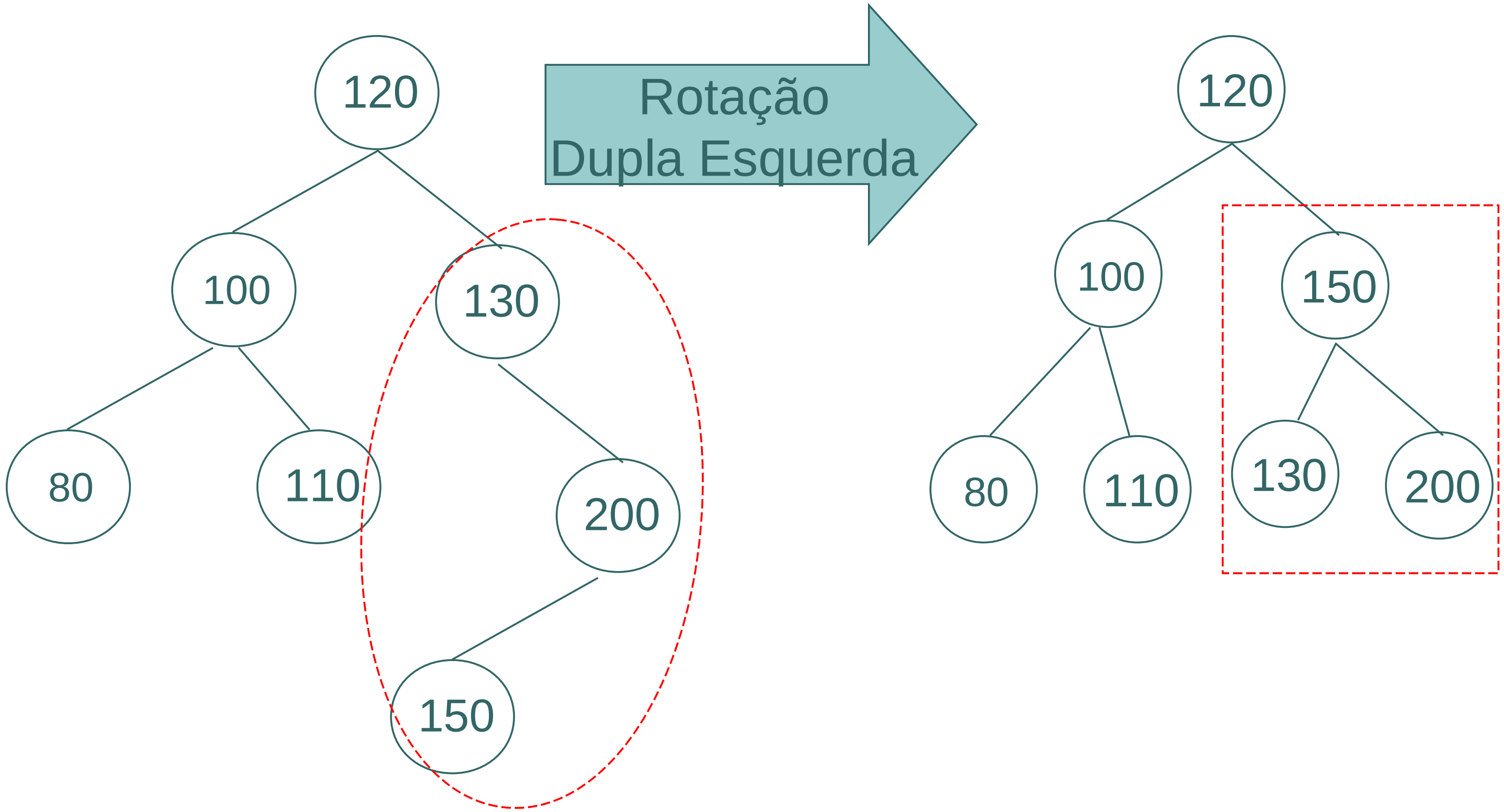
- Rotação Dupla Esquerda:

- b<y<c
- p<y<z
- y passa a ser a raiz
-h≥0



Árvore AVL

- Exemplo de Rotação Dupla Esquerda:



Árvore AVL

- Identificação do caso a ser aplicado:
  - Supondo que o nó **q** foi incluído na árvore **T**, se houver desbalanceamento da árvore, sendo **p** o nó raíz do desbalanceamento mais próximo das folhas de **T**:
    - |he(p) – hd(p)| = 2

    he: altura da subárvore esquerda

    hd: altura da subárvore direita

    - Caso 1: he(p)>hd(p)

      Sendo **u** o filho à esquerda de **p**:

      1.1. he(u)>hd(u) ⇒ rotação direita

      1.2. hd(u)>he(u) ⇒ rotação dupla direita

    - Caso 2: hd(p)>he(p)

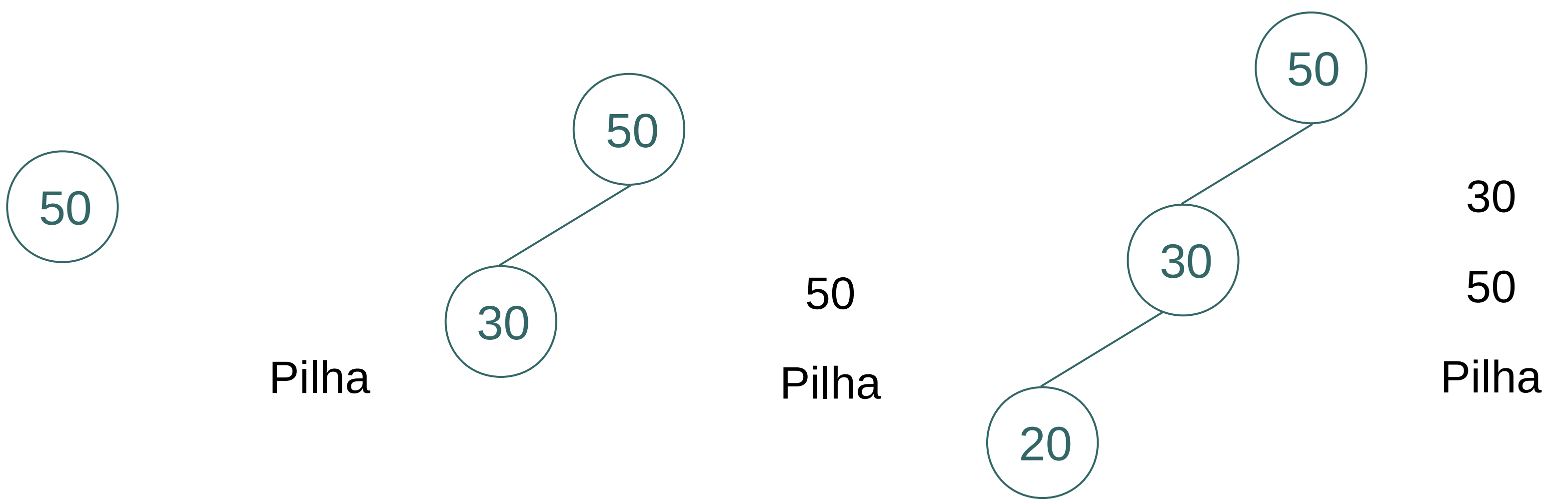
      Sendo **z** o filho à direita de **p**:

      2.1. hd(z)>he(z) ⇒ rotação esquerda

      2.2. he(z)>hd(z) ⇒ rotação dupla esquerda

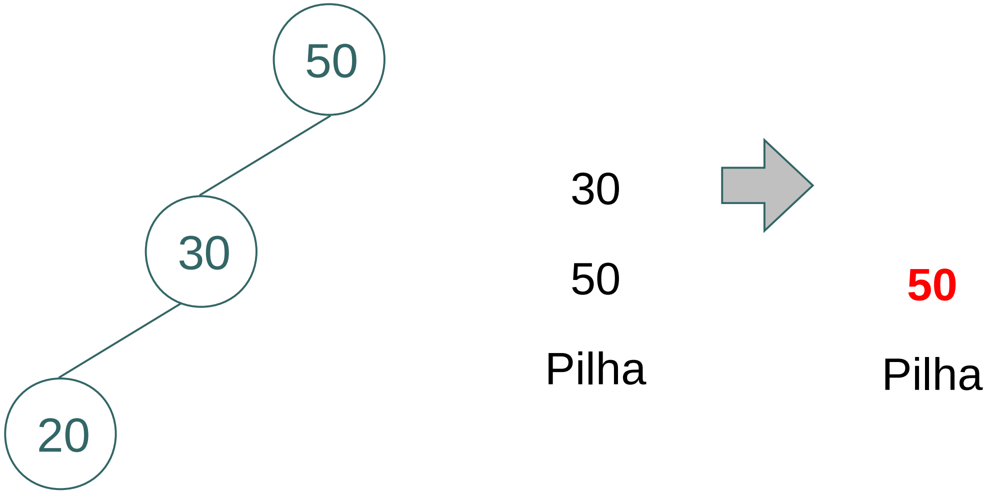
Árvore AVL

- Exemplo de “crescimento” de uma árvore AVL (com balanceamento):
  - Suponhamos a inserção das chaves na seguinte seqüência: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



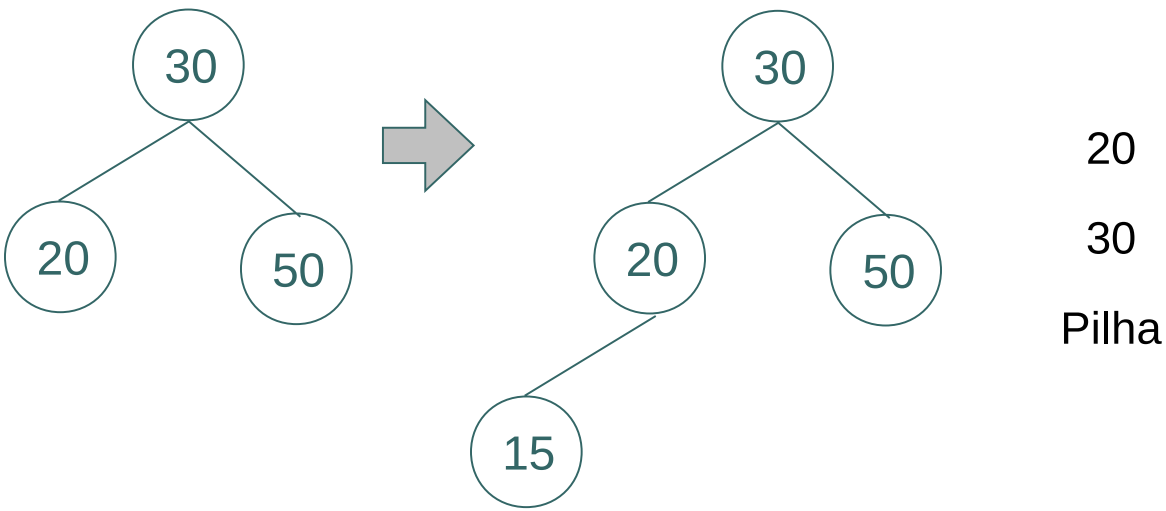
Árvore AVL

- Seqüência inserida: <u>50, 30, 20,</u> 15, 10, 12, 18, 17, 25, 24.



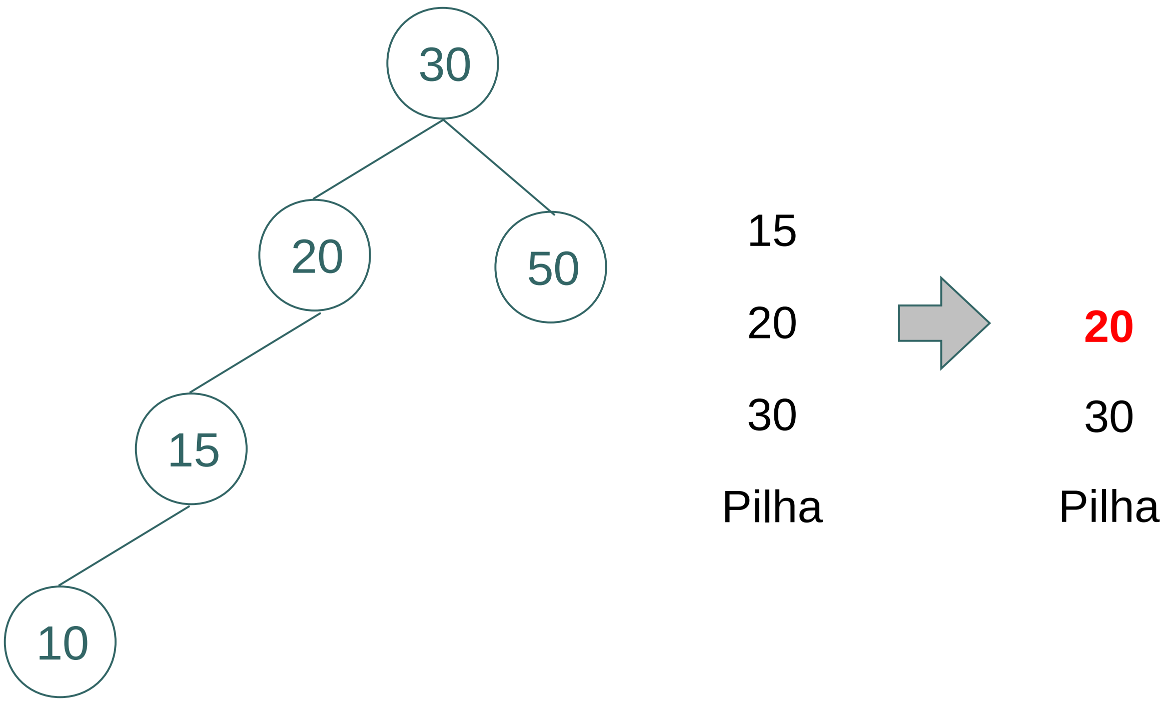
Árvore AVL

- Seqüência inserida: <u>50, 30, 20, 15,</u> 10, 12, 18, 17, 25, 24.



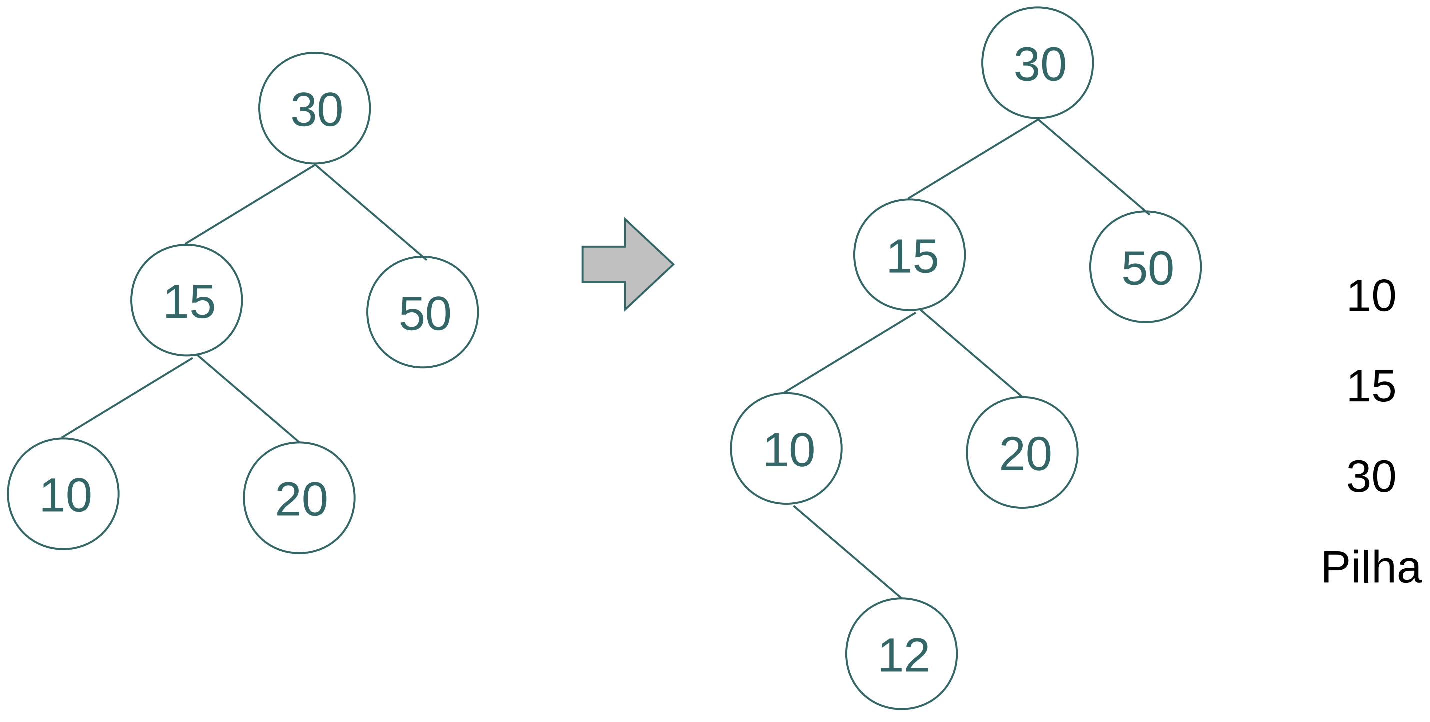
Árvore AVL

- Seqüência inserida: <u>50, 30, 20, 15, 10,</u> 12, 18, 17, 25, 24.



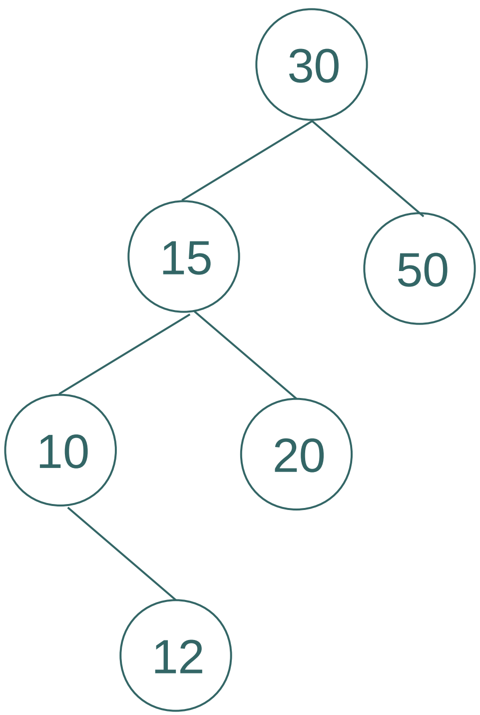
Árvore AVL

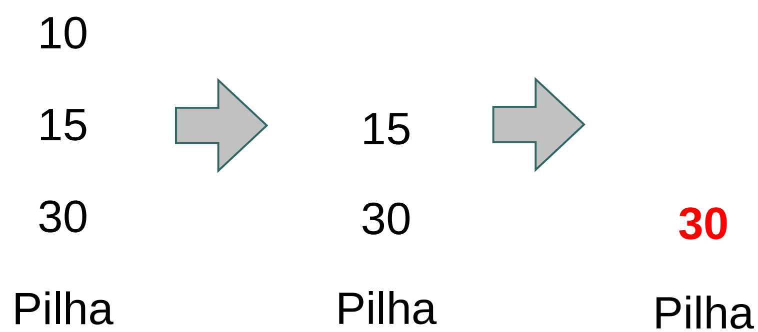
- Seqüência inserida: <u>50, 30, 20, 15, 10, 12</u>, 18, 17, 25, 24.



Árvore AVL

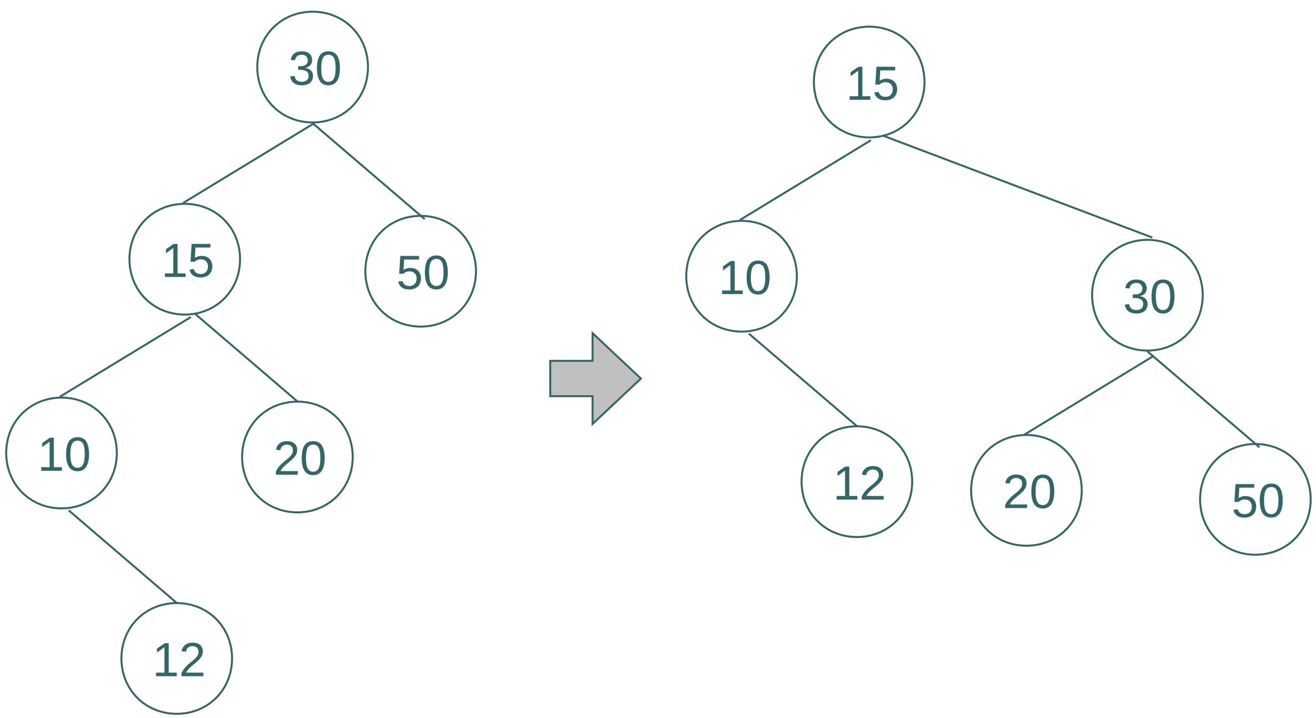
- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.





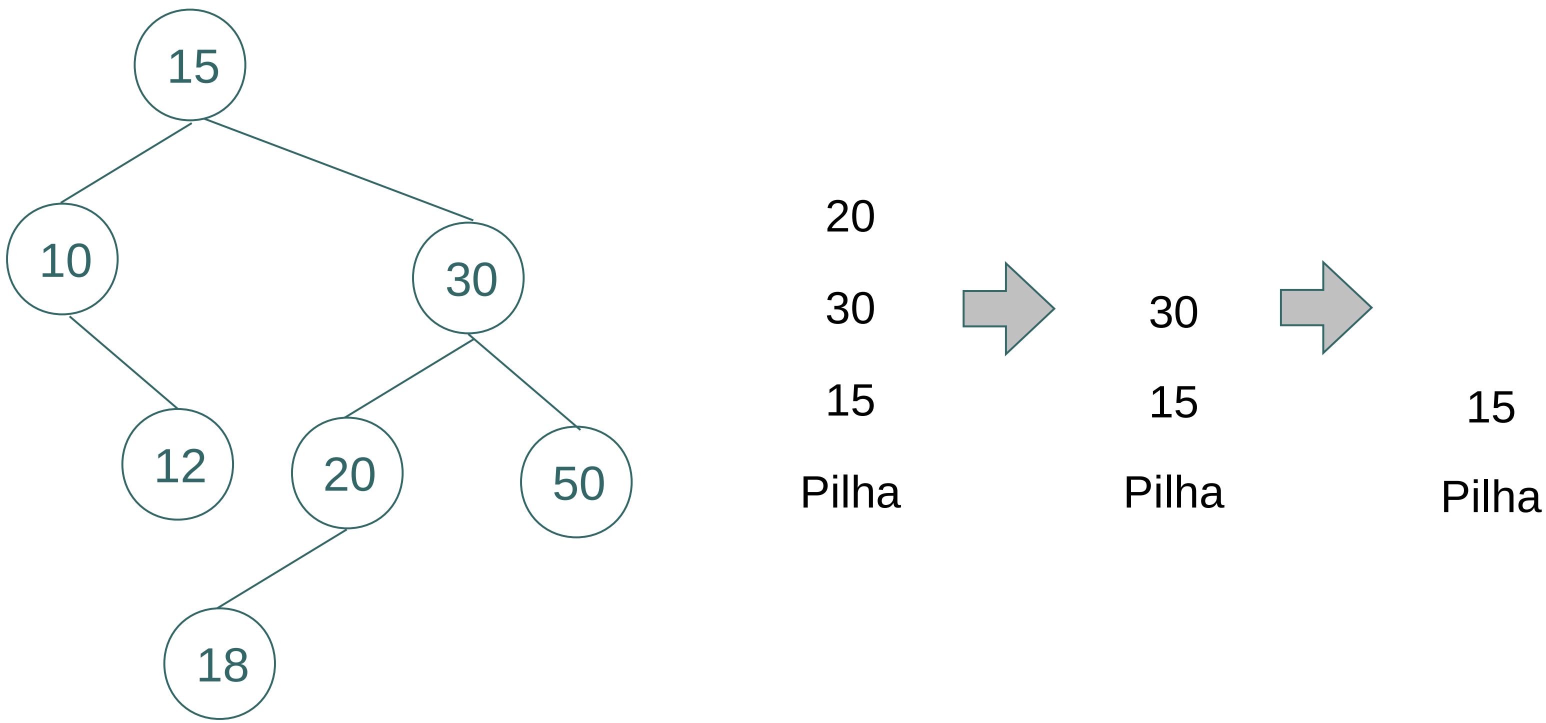
Árvore AVL

- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



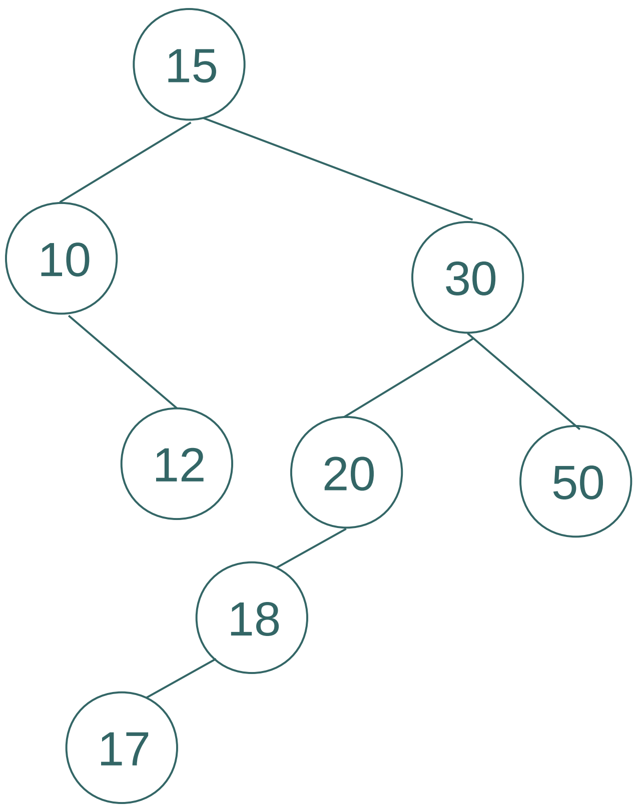
Árvore AVL

- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



Árvore AVL

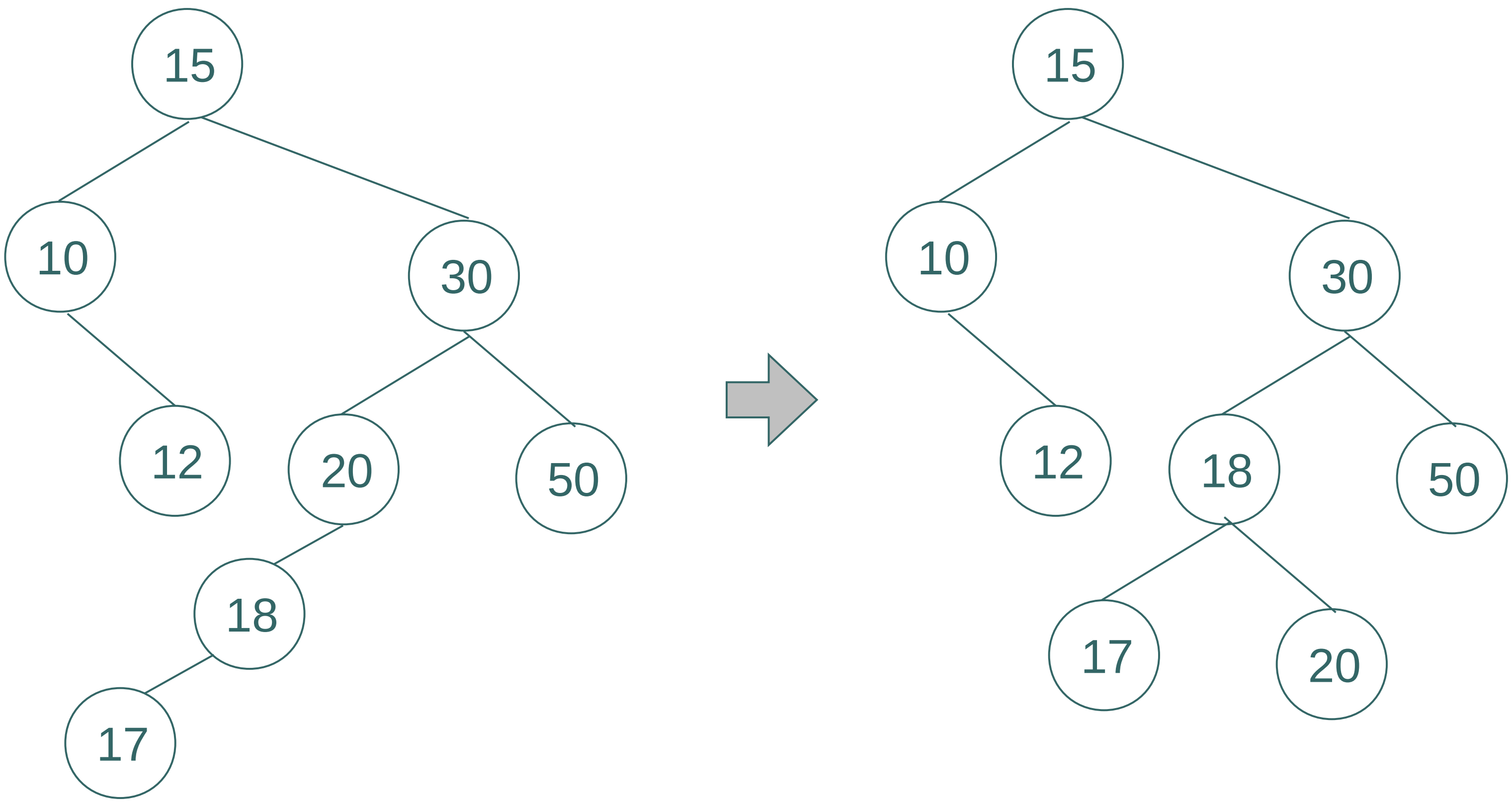
- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



Pilha		Pilha
18		
20		**20**
30	⇨	30
15		15

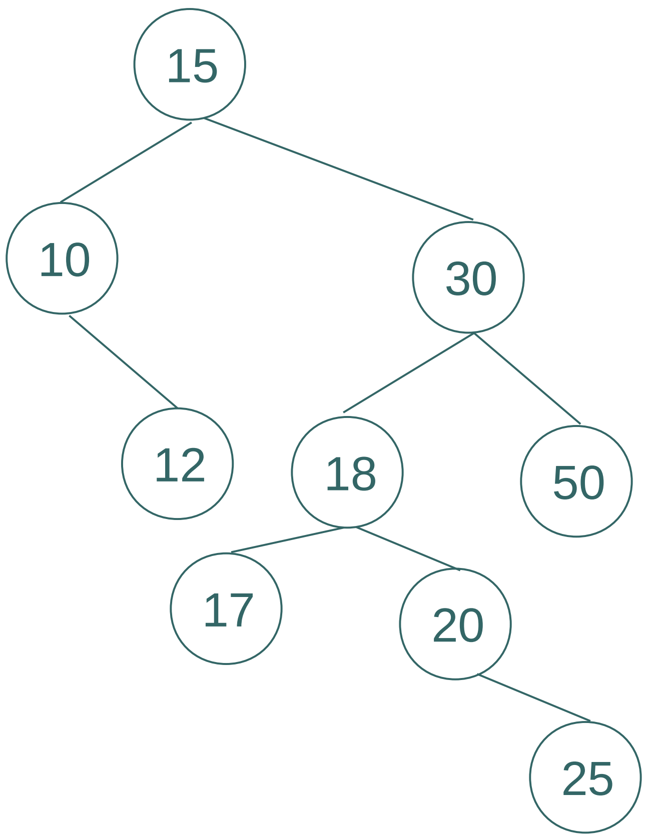
Árvore AVL

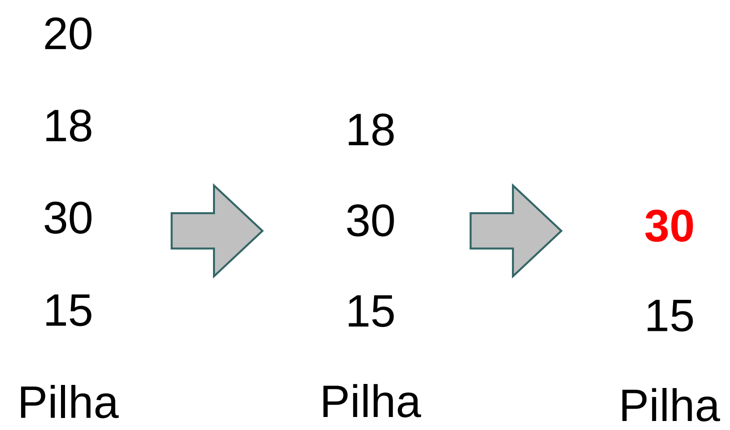
- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



Árvore AVL

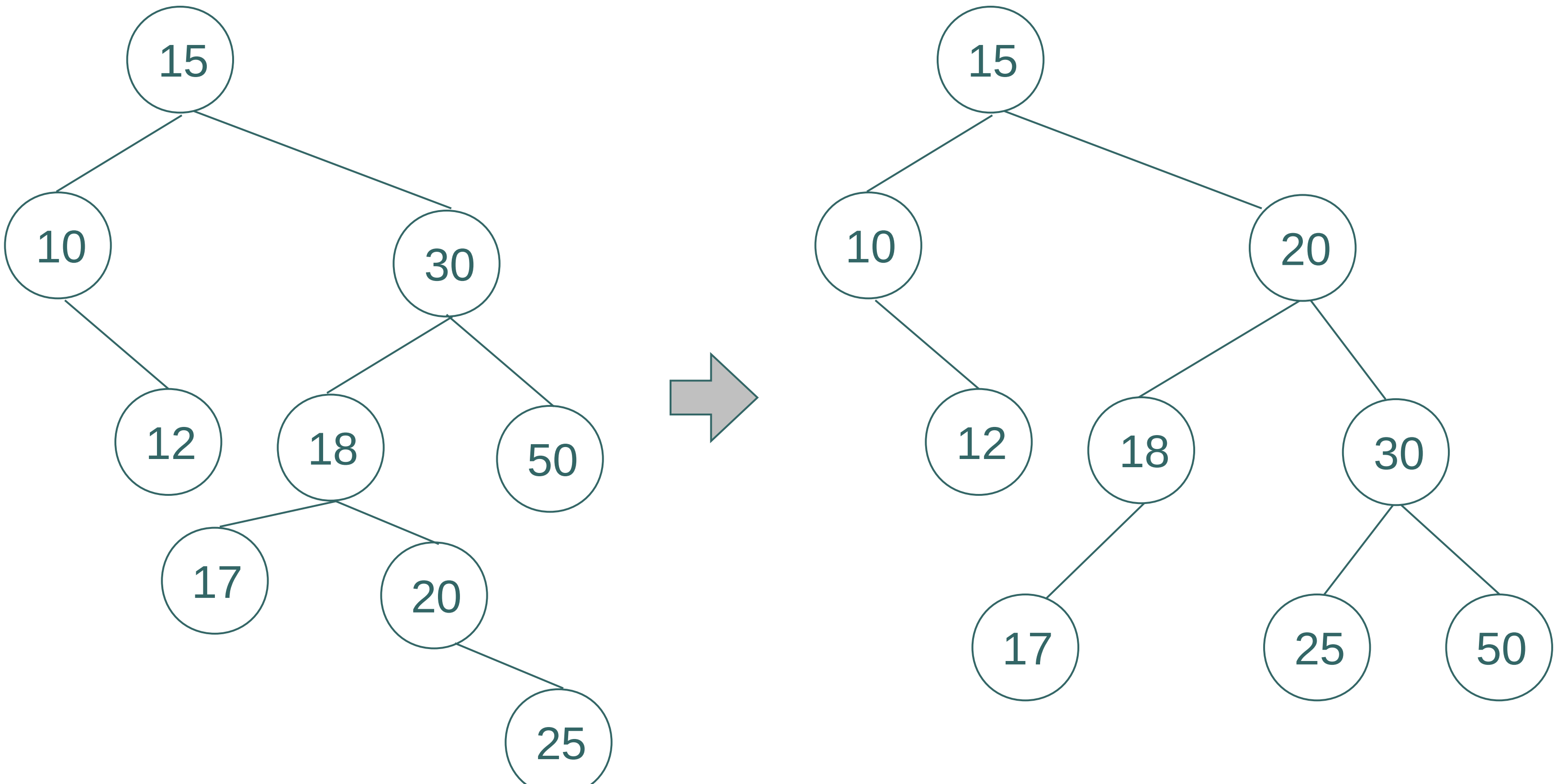
- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.





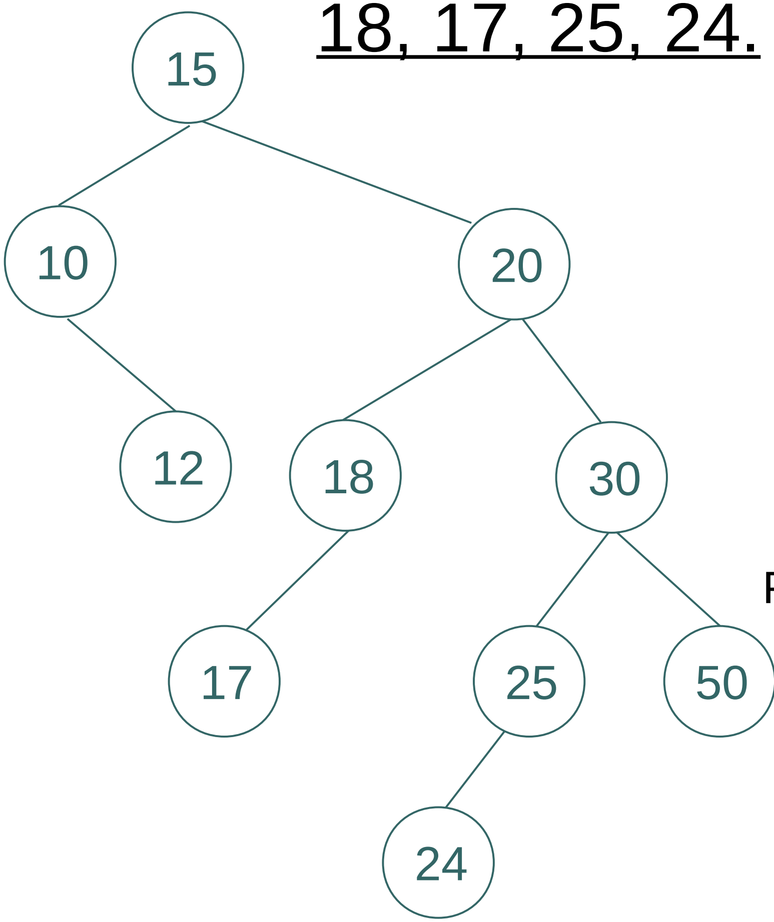
Árvore AVL

- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.



Árvore AVL

- Seqüência inserida: <u>50, 30, 20, 15, 10, 12, 18, 17, 25, 24.</u>



25
30
20
15
Pilha

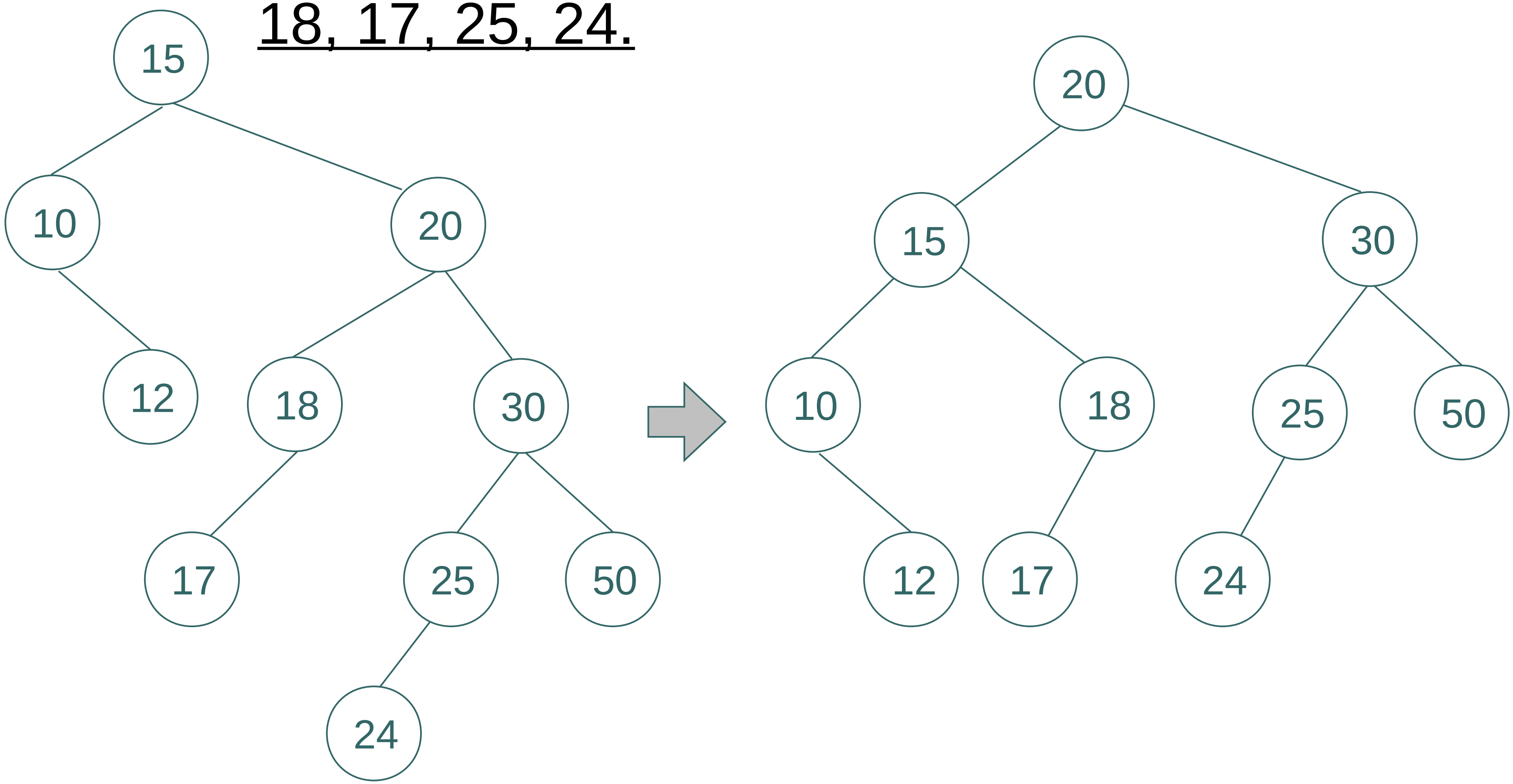
30
20
15
Pilha

20
15
Pilha

15
Pilha

Árvore AVL

- Seqüência inserida: 50, 30, 20, 15, 10, 12, 18, 17, 25, 24.

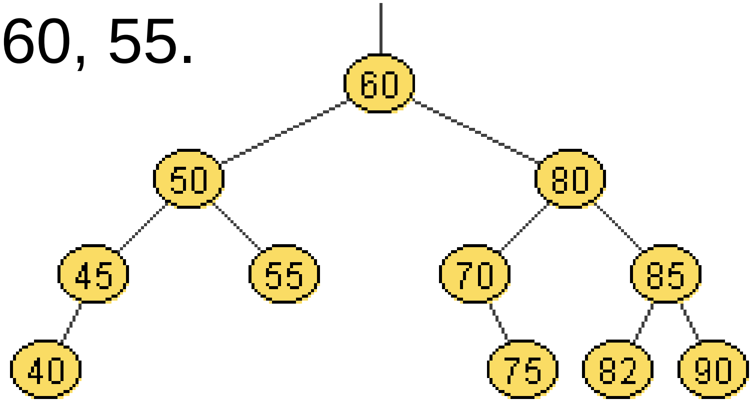


Árvore AVL

- Exercício:
  - Dada a seguinte seqüência de chaves numéricas a serem inseridas em uma árvores AVL, mostre a seqüência de árvores AVL produzidas após a inserção de cada uma destas chaves:
    - 50, 40, 45, 70, 80, 60, 90, 85, 82, 55, 75.

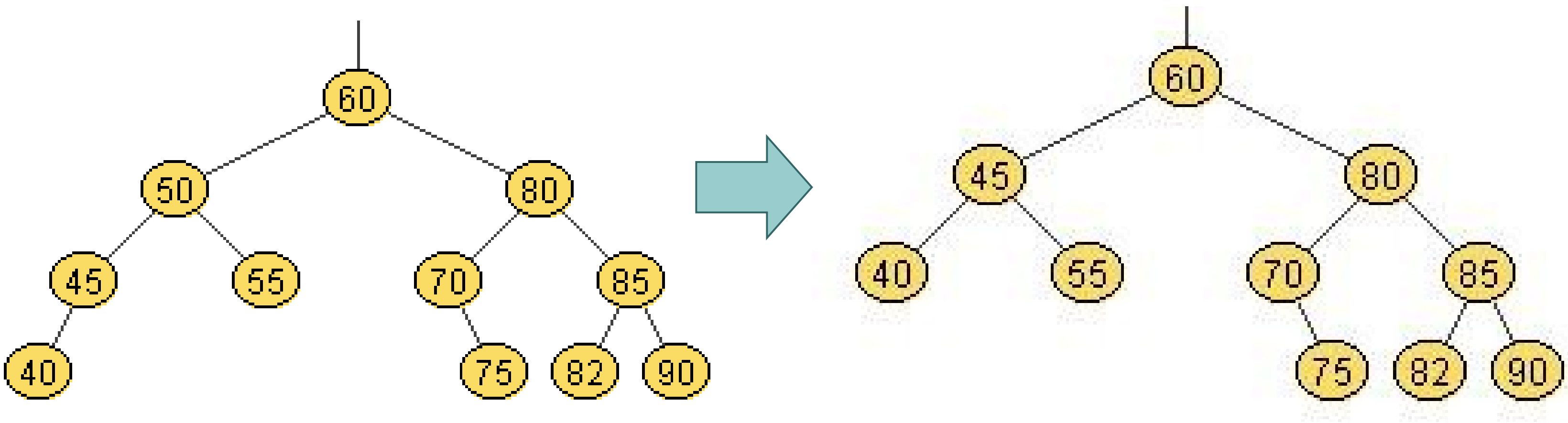
Árvore AVL

- Exercício
  - Dada a seguinte árvores AVL, mostre a seqüência de árvores produzidas após a remoção de cada chave da seguinte seqüência:
    - 50, 60, 55.



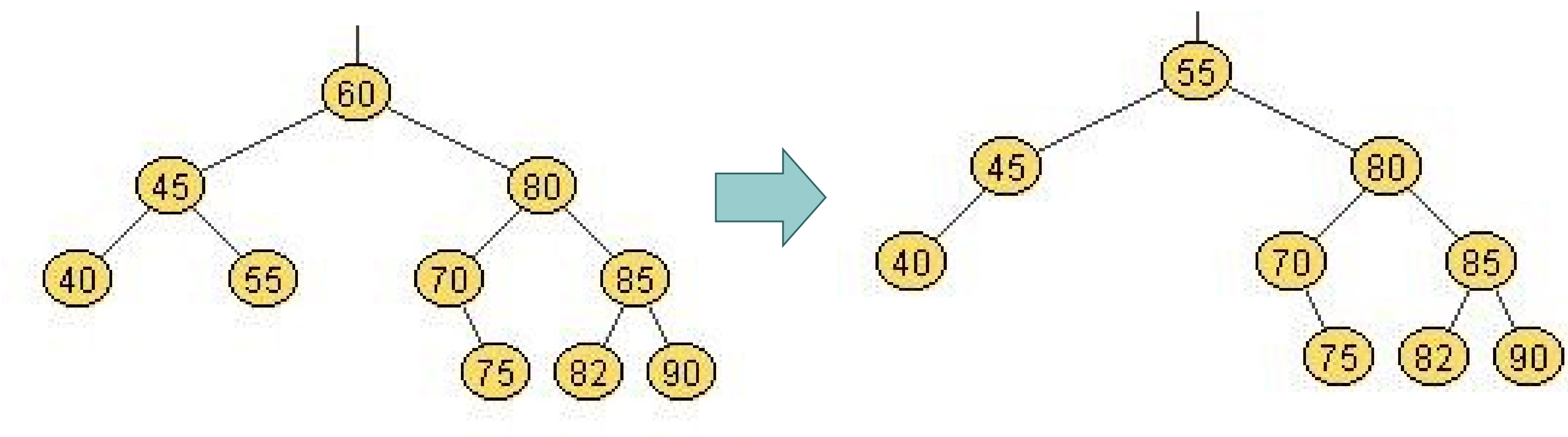
Árvore AVL

- Exclusão do 50:



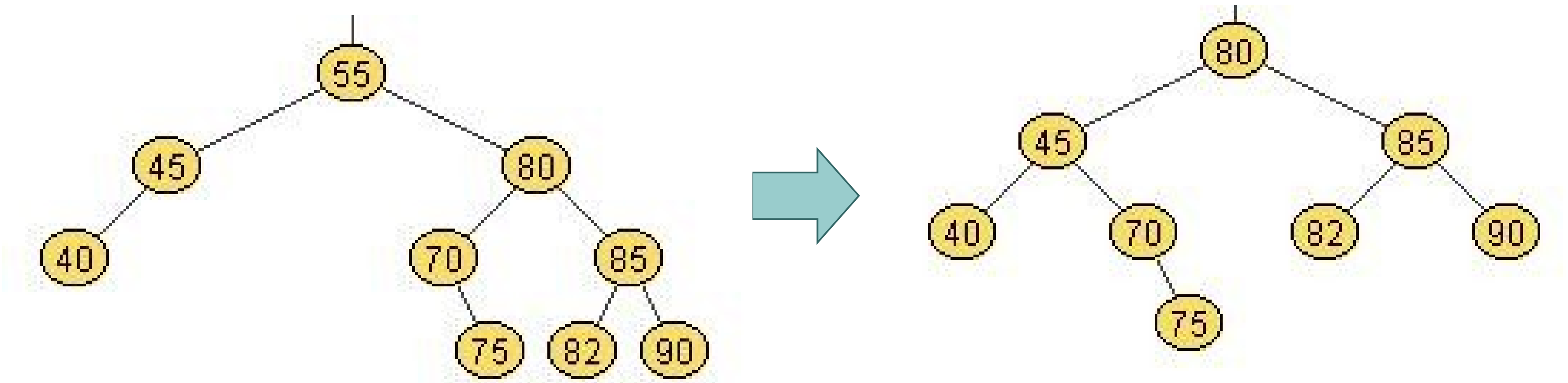
Árvore AVL

- Exclusão do 60:



Árvore AVL

- Exclusão do 55:



Árvore AVL – Algoritmos de Rotação

```
    Nodo* rot_dir(Nodo* N3)
{
        Nodo *N2;

        N2 = N3-> esq;
        if (N2->dir != NULL)
              N3->esq = N2->dir;
        else
              N3->esq = NULL;
        N2->dir = N3;
        return N2;
}
```

```
    Nodo* rot_esq(Nodo* N3)
{
        Nodo *N2;

        N2 = N3-> dir;
        if (N2->esq != NULL)
              N3->dir = N2->esq;
        else
              N3->dir = NULL;
        N2->esq = N3;
        return N2;
}
```

```
    Nodo* rot_dupla_dir(Nodo* N3)
{
        Nodo *N1= N3->esq;
        Nodo *N2= N1->dir;

        if (N2->esq != NULL)
              N1->dir = N2->esq;
        else
              N1->dir = NULL;
        if (N2->dir != NULL)
              N3->esq = N2->dir;
        else
              N3->esq = NULL;
        N2->esq = N1;
        N2->dir = N3;
        return N2;
}
```

```
    Nodo* rot_dupla_esq(Nodo* N3)
{
        Nodo *N1= N3->dir;
        Nodo *N2= N1->esq;

        if (N2->dir != NULL)
              N1->esq = N2->dir;
        else
              N1->esq = NULL;
        if (N2->esq != NULL)
              N3->dir = N2->esq;
        else
              N3->dir = NULL;
        N2->dir = N1;
        N2->esq = N3;
        return N2;
}
```

Bibliografia

- Azeredo, Paulo. Notas de aula de Algoritmos e Estruturas de Dados. INF/UFRGS, 2000;
- Celes, Waldemar et al. Introdução a Estruturas de Dados. Editora Campus, 2004;
- Wirth, Niklaus. Algoritmos e Estruturas de Dados. Editora PHB;
- Ziviani, Nivio. Projeto de Algoritmos. Editora Pioneira.