UNIFESP

UNIVERSIDADE FEDERAL DE SÃO PAULO

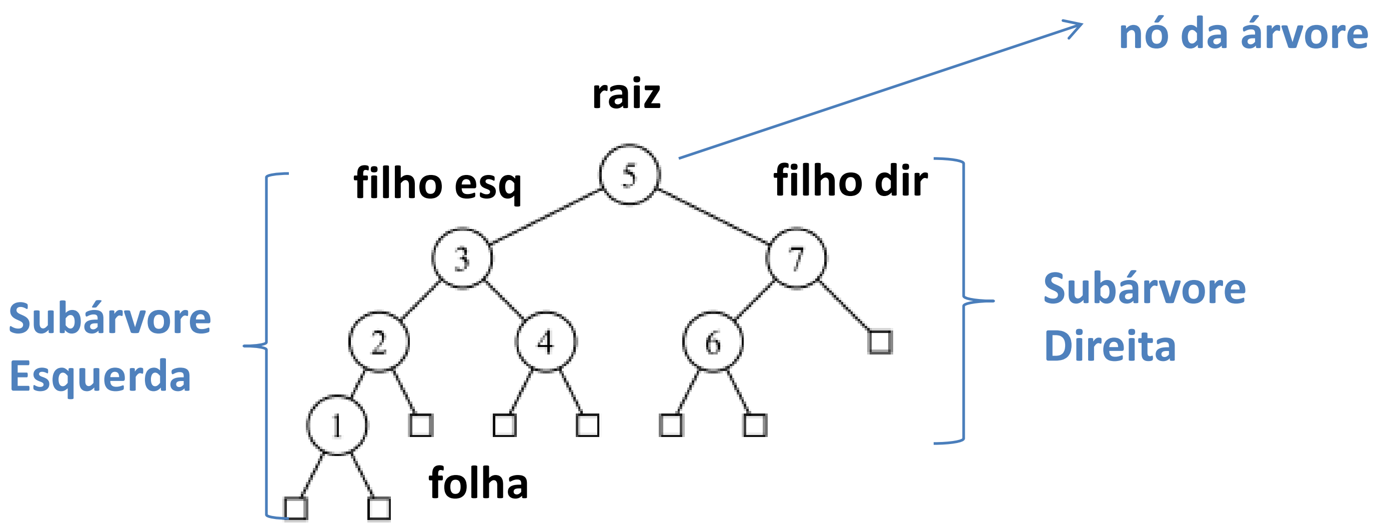
1933

Árvores Binárias e AVL

Prof. Lilian Berton

São José dos Campos, 2018

Árvore binária




```
struct cel {
  int conteudo;
  struct cel *esq;
  struct cel *dir;
};
typedef cel *Arvore;
```

O campo conteúdo armazena a informação
O campo esq contém o endereço de um nó ou NULL
O campo dir contém o endereço de um nó ou NULL

Árvore binária - propriedades

- Uma árvore binária é um conjunto A de nós tal que:

1. Os filhos de cada elemento de A pertencem a A
2. Todo elemento de A tem no máximo um pai
3. O único elemento de A que não tem pai é a raiz
4. Os filhos direita e esquerda de A são distintos
5. Não há ciclos em A

Árvore binária

- Um **caminho** em uma árvore é uma sequência de nós. Dizemos que Y_0 é a origem, Y_k é o término e k é o comprimento do caminho.

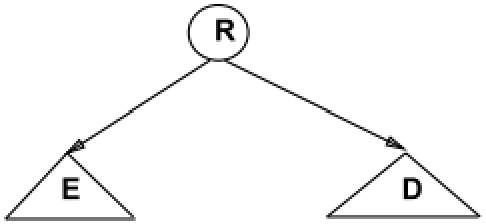
- **Endereço** de uma árvore: o endereço da árvore é a sua raiz. O endereço da árvore vazia é NULL.

- Uma árvore é composta de subárvores.

- **Recursão**: Para toda árvore binária, vale uma das alternativas:
- A é NULL ou
- A-> esq e A->dir são árvores

Árvore binária de pesquisa

- Acesso direto e sequencial eficientes.
- Facilidade de inserção e retirada de registros.
- Boa taxa de utilização de memória.
- Todos os registros com chaves menores estão na subárvore à esquerda. Todos os registros com chaves maiores estão na subárvore à direita.



E.chave < X.chave < D.chave

Pesquisa em uma árvore

1. Compare-a com a chave que está na raiz.

2. Se x é menor, vá para a subárvore esquerda.

3. Se x é maior, vá para a subárvore direita.

4. Repita o processo recursivamente, até que a chave procurada seja encontrada ou um nó folha é atingido.

5. Se a pesquisa tiver sucesso então o conteúdo do registro retorna no próprio registro x.

Árvore binária de pesquisa

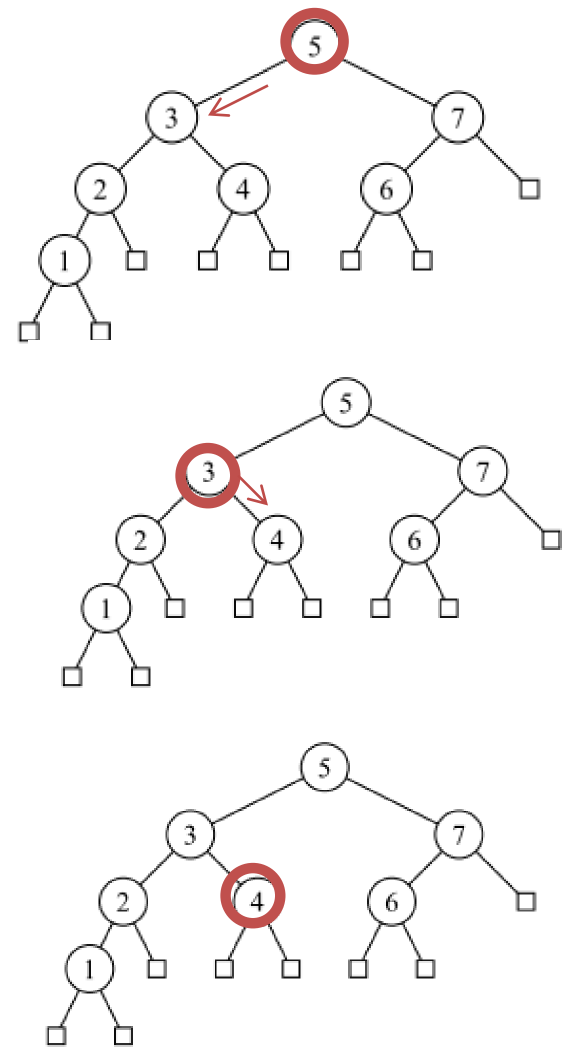
Versão recursiva

```
no *busca (Arvore A, int k) {
    if (A == NULL || A->chave == k)
        return A;
    if(A->chave > k)
        return busca(A->esq, k);
    else
        return busca (A->dir,k);
}
```

Versão iterativa

```
no *busca (Arvore A, int k) {
    while (A != NULL || A->chave != k)
        if(A->chave > k)
            A = A->esq;
        else
            A = A->dir;
    }
    return A;
}
```

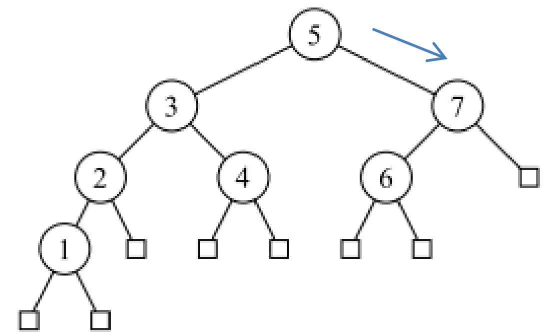
Busca o elemento 4

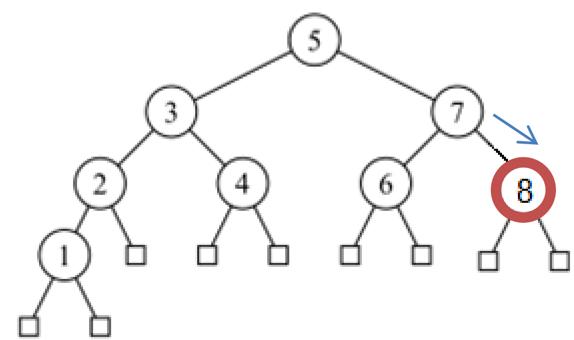


Inserção em uma árvore binária de busca

```
//recebe uma árvore e uma chave k, cria um novo nó e insere como nó folha na árvore
Arvore insere (Arvore A, int k) {
      no *novo;
      novo = (no*) malloc (sizeof(no));
      novo->chave = k;
      novo->esq = novo->dir = NULL;

      no *antep, *ultimo;
      if (A == NULL) return novo;
      ultimo = A;
      while(ultimo != NULL) {
            antep = ultimo;
            if (ultimo ->chave > k) ultimo = ultimo ->esq;
            else ultimo = ultimo ->dir;
      }
      if(antep ->chave > k) antep ->esq = novo;
      else antep ->dir = novo;
      return A;
}
```

Insere o elemento 8



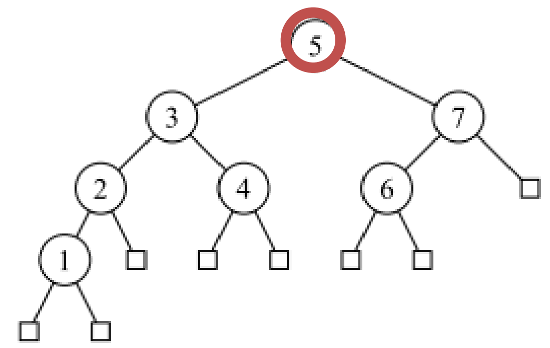


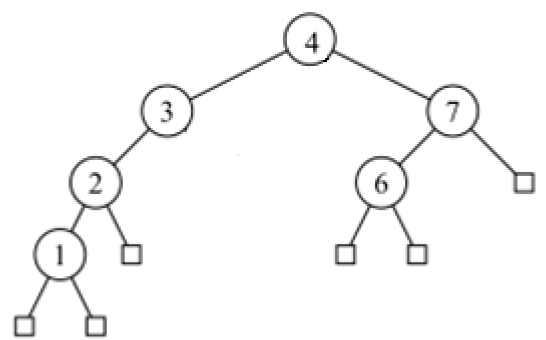
Sempre insere como nó folha

Remoção da raiz em uma árvore binária de busca

```
//recebe uma árvore, remove a raiz e rearranja a árvore
Arvore removeRaiz (Arvore A) {
    no *p,*q;
    if(A->esq == NULL) q = A->dir;  //se tiver apenas um filho
    else {
        p = A; q = A->esq;
        while (q->dir != NULL) {
            p = q; q = q->dir;
        }
        if(p ! = A) {
            p->dir  = q->esq;
            q->esq = A->esq;
        }
        q->dir = A->dir;
    }
    free(A);
    return q; //nova raiz
}
```

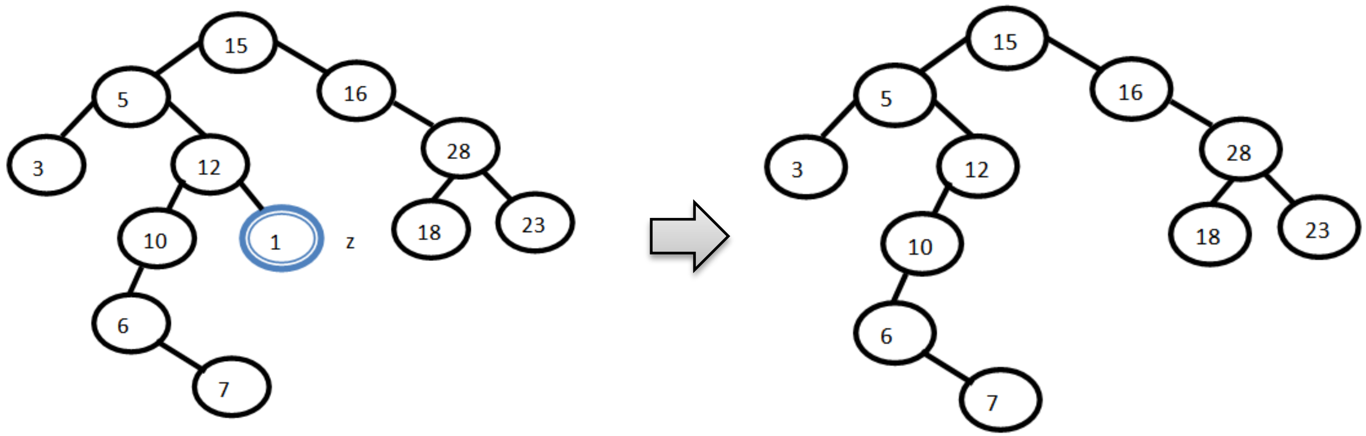
Remove a raiz

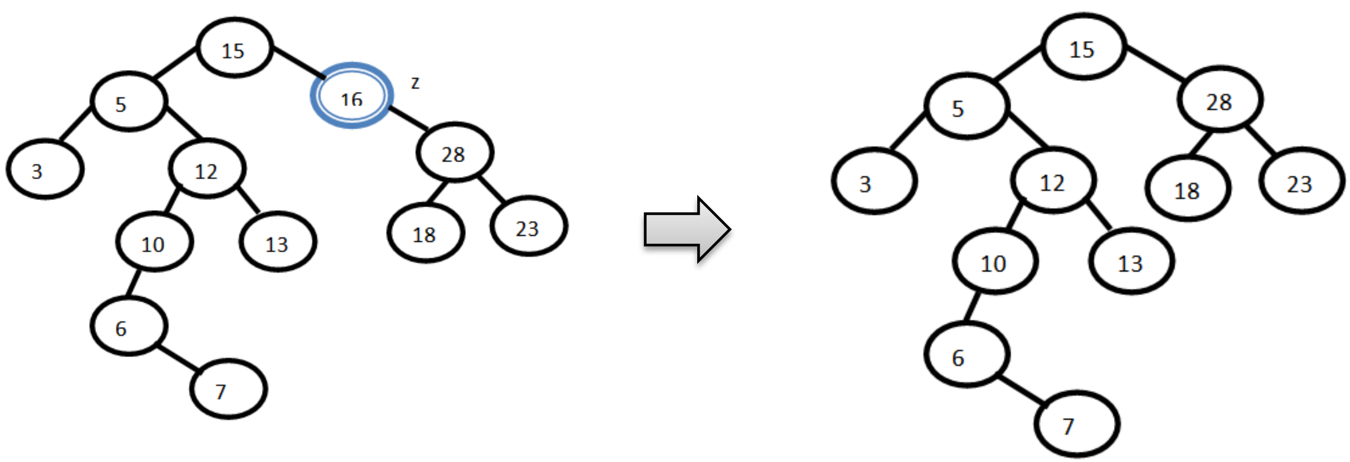




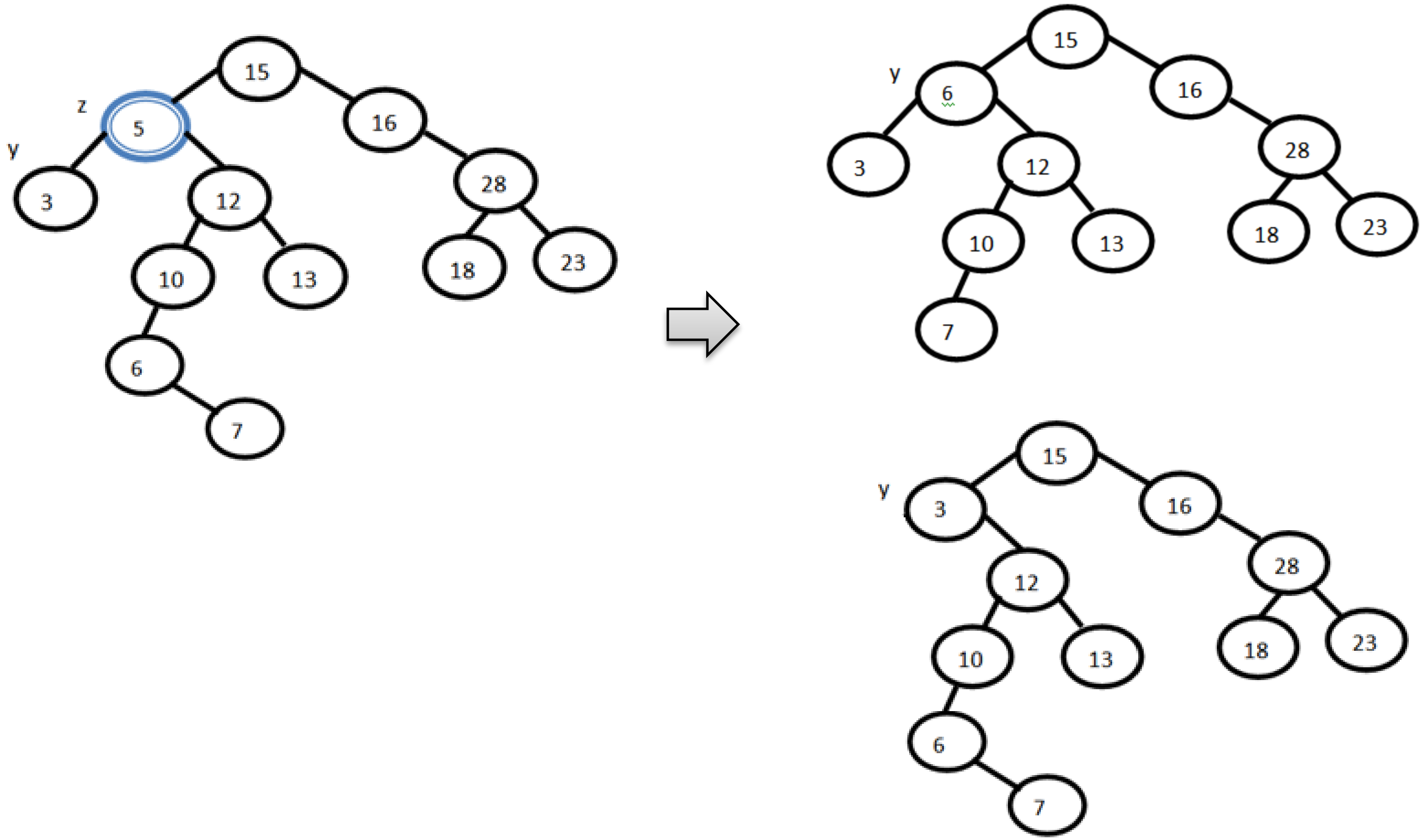
Substituiu pela chave mais a direita na subarvore esquerda
Também poderia substituir pela chave mais a esquerda na subarvore direita.

Exemplo remoção





Exemplo remoção

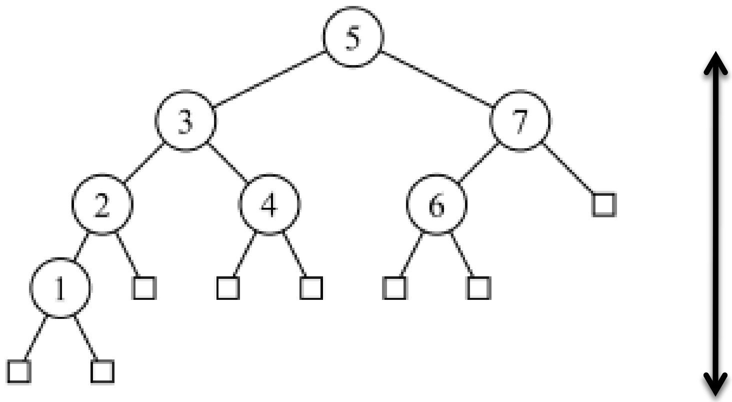


Buscar o registro mais à direita na subárvore esquerda
Ou o registro mais à esquerda na subárvore direita.

Complexidade

- As operações busca, inserção e remoção podem ser executadas em tempo O(h) em uma árvore de pesquisa binária de altura h.

Altura de uma árvore



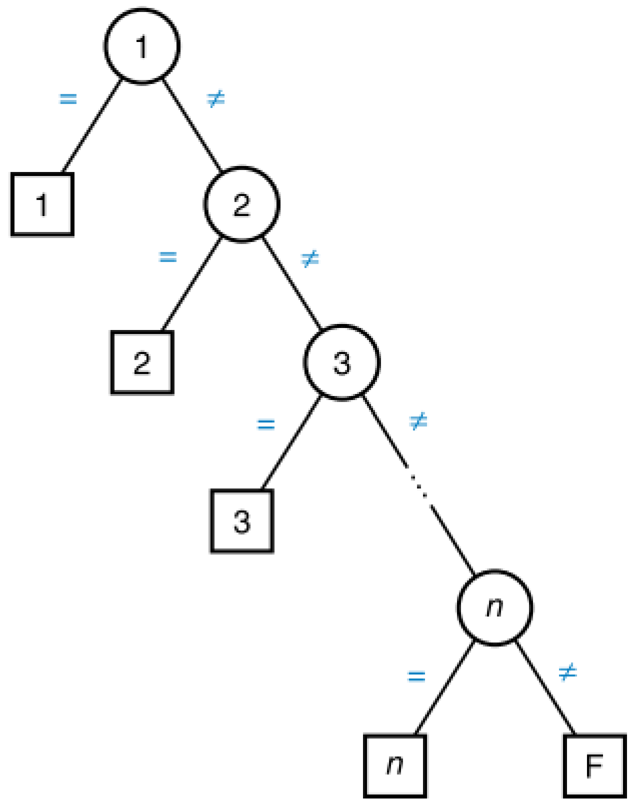
A **altura de uma árvore** é o comprimento do caminho mais longo deste nó raiz até um nó folha.

$\log_2 n < h < n$

```
int altura (Arvore A) {
if (A == NULL) {
    return -1;
} else {
    int he = altura (A->esq);
    int hd = altura (A->dir);
    if(he < hd) return hd + 1;
    else return he+1;
}
```

Árvore balanceada: se as subarvores esquerda e direita de cada nó possuem a mesma altura.

Desvantagens



- Se inserirmos elementos em ordem crescente ou decrescente a árvore torna-se uma lista.

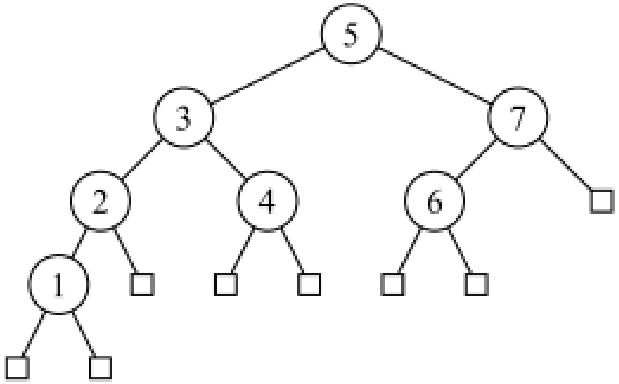
- Custo para buscar um elemento pode ser O(n).

Varredura e impressão da árvore

- Os nós de uma árvore podem ser visitados de ordens diferentes:

1. **Esquerda-raiz-direita ERD (em ordem)**
2. **Esquerda-direita-raiz RED (pré ordem)**
3. **Raiz-esquerda-direita EDR (pós ordem)**

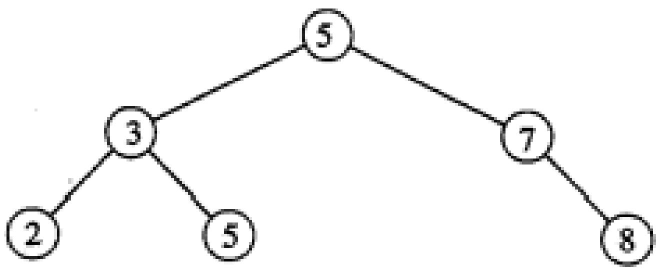
```
void ERD(Arvore A) {
    if (A != NULL) {
        ERD(A->esq);
        printf("%d", A->conteúdo);
        ERD(A->dir);
    }
}
```



1, 2, 3, 4, 5, 6, 7

Versão recursiva!

Como seria o algoritmo para impressão de árvore em pré-ordem e pós-ordem?




```
void emOrdem(struct No *pNo) {
    if(pNo != NULL) {
        emOrdem(pNo→pEsquerda);
        visita(pNo);
        emOrdem(pNo→pDireita);
    }
}
```

2, 3, 5, 5, 7, 8

```
void preOrdem(Struct No *pNo){
    if(pNo != NULL){
        visita(pNo);
        preOrdem(pNo→pEsquerda);
        preOrdem(pNo→pDireita);
    }
}
```

5, 3, 2, 5, 7, 8

```
void posOrdem(Struct No *pNo){
    if(pNo != NULL){
        posOrdem(pNo→pEsquerda);
        posOrdem(pNo→pDireita);
        visita(pNo);
    }
}
```

2, 5, 3, 8, 7, 5

Percurso em árvore binária é O(n)

Teorema 12.1

Se x é a raiz de uma subárvore de n nós, então a chamada INORDER-TREE-WALK(x) demora o tempo $\Theta(n)$.

Prova Seja $T(n)$ o tempo tomado por INORDER-TREE-WALK quando ele é chamado na raiz de uma subárvore de n nós. INORDER-TREE-WALK demora um espaço de tempo pequeno e constante em uma subárvore vazia (para o teste $x \neq$ NIL) e então $T(0) = c$ para alguma constante positiva c.

Para $n > 0$, suponha que INORDER-TREE-WALK seja chamado em um nó x cuja subárvore esquerda tem k nós e cuja subárvore direita tem $n - k - 1$ nós. O tempo para executar INORDER-TREE-WALK(x) é $T(n) = T(k) + T(n-k-1) + d$ para alguma constante positiva d que reflete o tempo para executar INORDER-TREE-WALK(x), excetuando-se o tempo gasto em chamadas recursivas.

Usamos o método de substituição para mostrar que $T(n) = \Theta(n)$, provando que $T(n) = (c + d)n + c$. Para $n = 0$, temos $(c + d) \cdot 0 + c = c = T(0)$. Para $n > 0$, temos

$$
\begin{aligned}
T(n) &= T(k) + T(n-k-1) + d \\
&= ((c+d)k + c) + ((c+d)(n-k-1) + c) + d \\
&= (c+d)n + c - (c+d) + c + d \\
&= (c+d)n + c\,,
\end{aligned}
$$

o que completa a prova.

Obtido em Cormen et al.

Desempenho de uma árvore

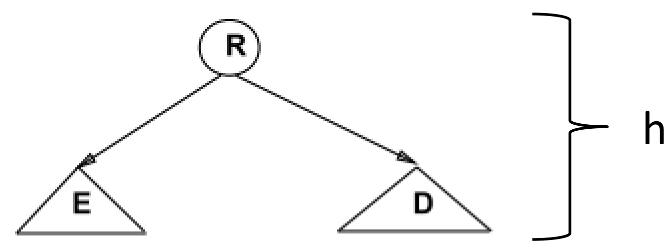
- O consumo de tempo da árvore (busca, inserção e remoção) é no pior caso proporcional a altura da árvore.

- Faz-se necessário manter a árvore balanceada, pois elas tem altura próxima a $\log_2 n$, sendo n o número de nós.

- Ex de árvore balanceada: AVL, vermelho-preto

Árvore AVL

- Esta estrutura foi criada em 1962 pelos soviéticos Adelson Velsky e Landis. Uma árvore binária é denominada AVL quando, para qualquer nó, as alturas de suas duas subárvores, esquerda e direita, diferem em módulo de até uma unidade.



- $|h(D) – h(E)| \leq 1$

- Todos os nós da AVL devem ter fator de balanceamento = -1, 0 ou 1.

- A busca, a inserção e remoção têm custo O(log n).

Busca, inserção e remoção em árvore AVL

- A busca, inserção e remoção é a mesma utilizada na **árvore binária**.

- **Busca**: seja x o nó sendo verificado. Caso x seja nulo então a busca não foi bem sucedida (o elemento não está na árvore ou árvore vazia). Verificar se a chave k é igual a x[chave] (valor chave armazenado no nó x), então a busca foi bem sucedida. Caso contrário, se k < chave[x] então a busca segue pela subárvore esquerda; caso contrário, a busca segue pela subárvore direita.

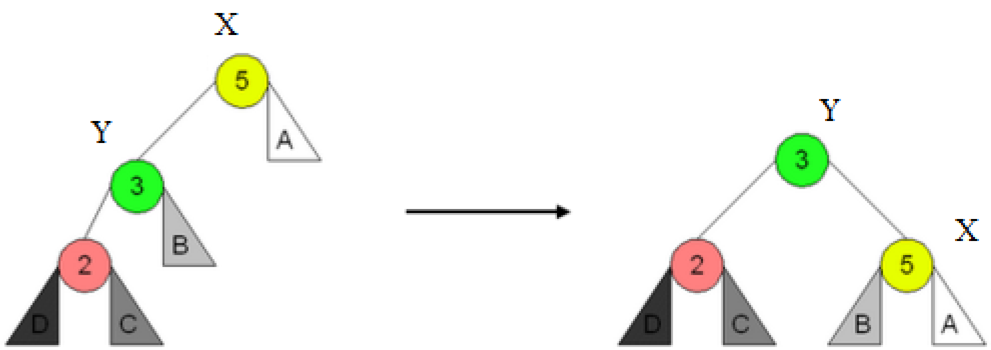
- **Inserção**: após a busca o local correto para a inserção do nó **k** será em uma subárvore vazia de uma folha da árvore.

- **Remoção**: após a busca verificar se o nó a ser excluído é folha, possui um filho ou nenhum filho e tratar cada caso.

- Depois de inserido/removido o nó, **a altura do nó pai e de todos os nós acima deve ser atualizada.** Em seguida o algoritmo de rotação simples ou dupla deve ser acionado para o primeiro nó pai desregulado.

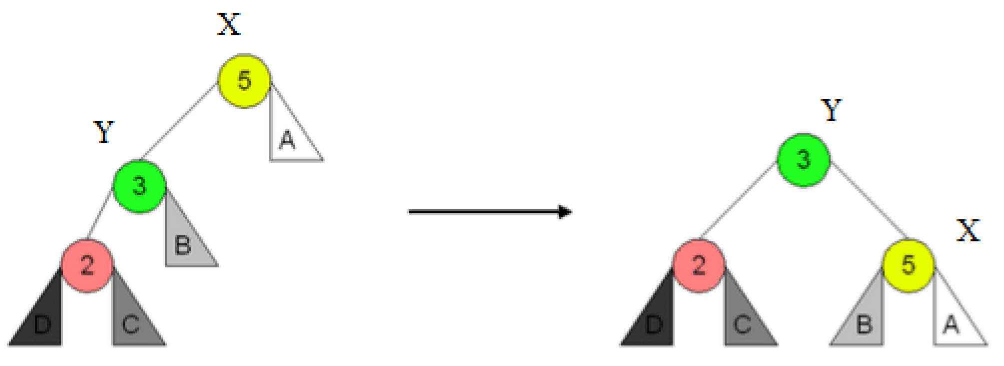
Rotação em AVL

- **Rotação à direita**:
- Deve ser efetuada quando a diferença das alturas **h** dos filhos de X é igual a -2 e dos filhos de Y é -1**:**
  - Seja Y o filho à esquerda de X
  - Torne o filho à direita de Y o filho à esquerda de X.
  - Torne X o filho à direita de Y



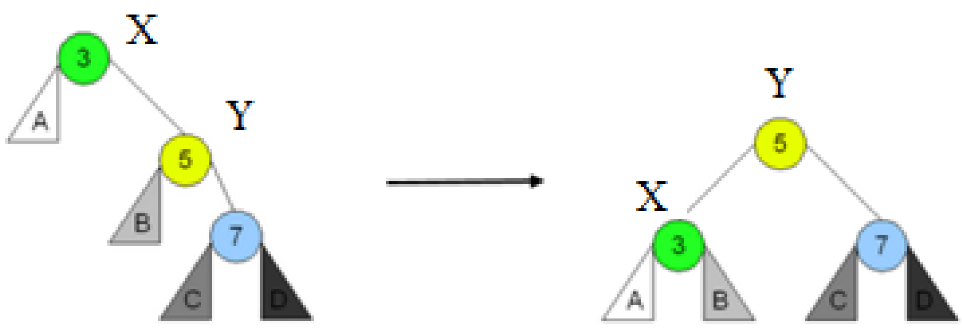
Rotação a direita

```
No* rotacao_direita (No* X) {
    No* Y = X->esq;
    if (Y->dir) X->esq = Y->dir;
    else X->esq = NULL;
    Y->dir = X;
    return Y;
}
```



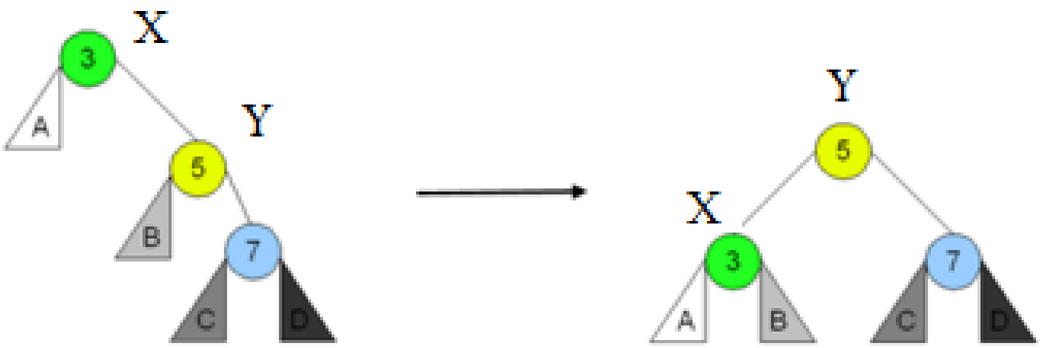
Rotação em AVL

- **Rotação à esquerda:**
- Deve ser efetuada quando a diferença das alturas **h** dos filhos de X é igual a 2 e dos filhos de Y é 1:
  - Seja Y o filho à direita de X
  - Torne o filho à esquerda de Y o filho à direita de X.
  - Torne X filho à esquerda de Y



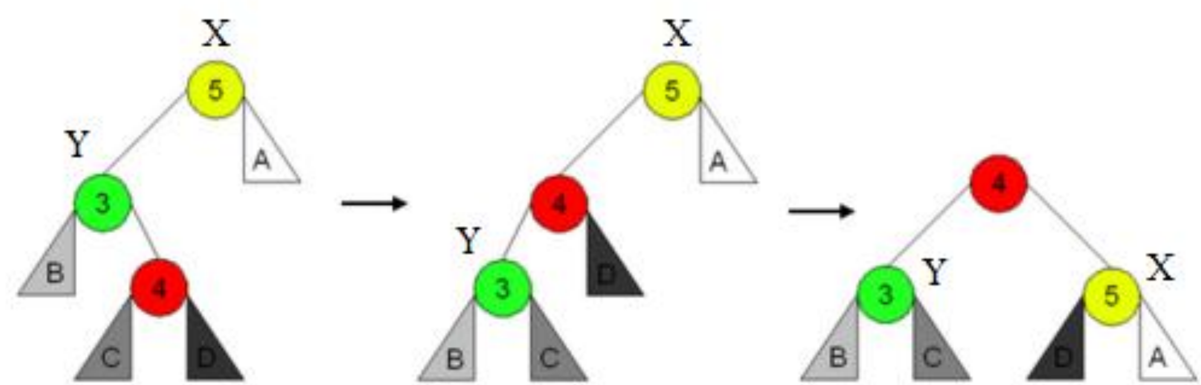
Rotação a esquerda

```
No* rotacao_esquerda (No* X) {
    No* Y = X->dir;
    if (Y->esq) X->dir= Y->esq;
    else X->dir= NULL;
    Y->esq= X;
    return Y;
}
```



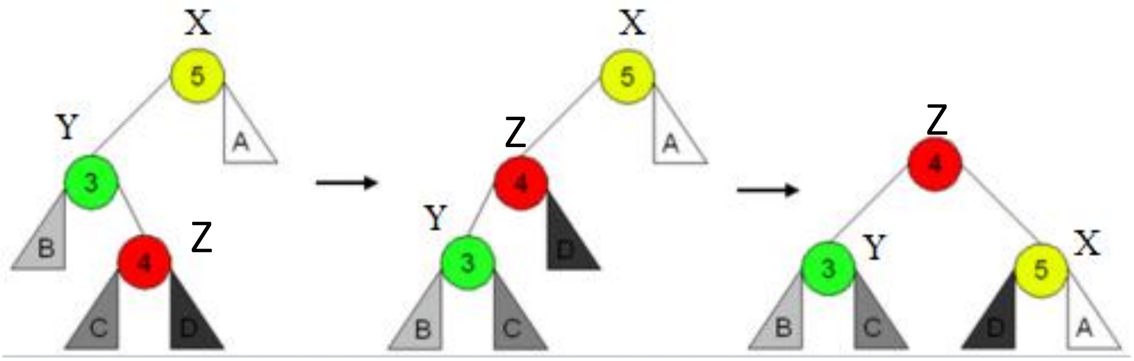
Rotação em AVL

- **Rotação dupla à direita**:
- Deve ser efetuada quando a diferença das alturas **h** dos filhos de X é igual a -2 e dos filhos de Y é 1.
  – Aplica rotação a esquerda em Y
  – Aplica rotação a direita em X.



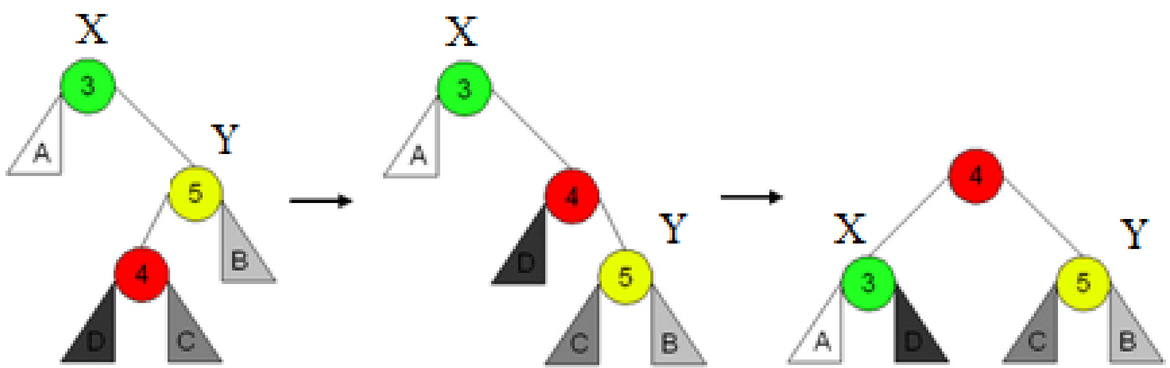
Rotação dupla a direita

```
No* rotacao_dupla_direita (No* X) {
    No* Y = X->esq;
    No* Z = Y->dir;
    if (Z->esq) Y->dir = Z->esq;
    else Y->dir= NULL;

    if(Z->dir) X->esq = Z->dir;
    else X->esq = NULL;

    Z->esq = Y;
    Z->dir = X;
    return Z;
}
```



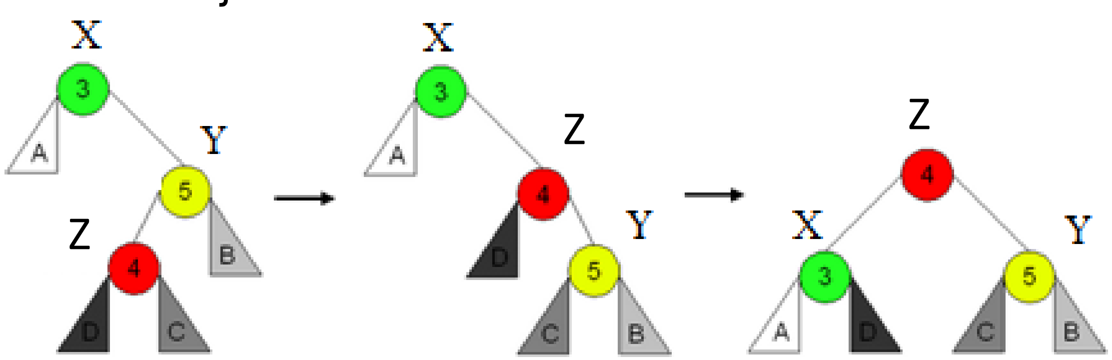
Rotação em AVL

- **Rotação dupla à esquerda**:
- Deve ser efetuada quando a diferença das alturas **h** dos filhos de X é igual a 2 e dos filhos de Y é -1.
  - Aplica rotação a direita em Y
  - Aplica rotação a esquerda em X.



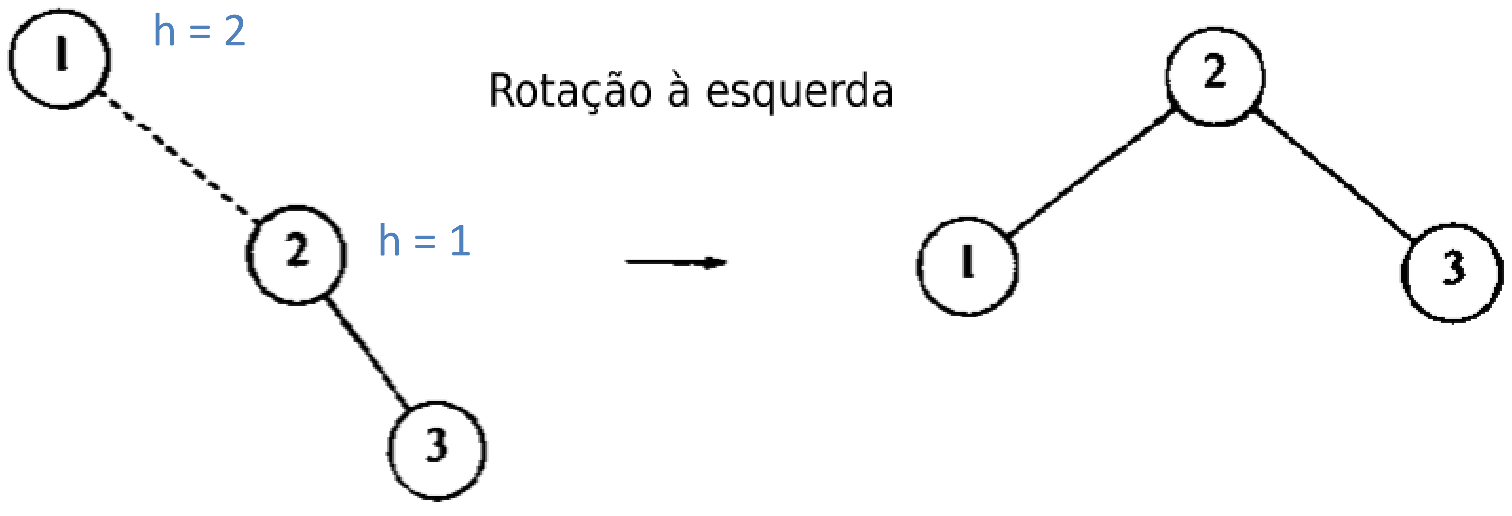
Rotação dupla a esquerda

```
No* rotacao_dupla_esquerda (No* X) {
    No* Y = X->dir;
    No* Z = Y->esq;
    if (Z->dir) Y->esq= Z->dir;
    else Y->esq= NULL;

    if(Z->esq) X->dir = Z->esq;
    else X->dir = NULL;

    Z->esq = X;
    Z->dir = Y;
    return Z;
}
```



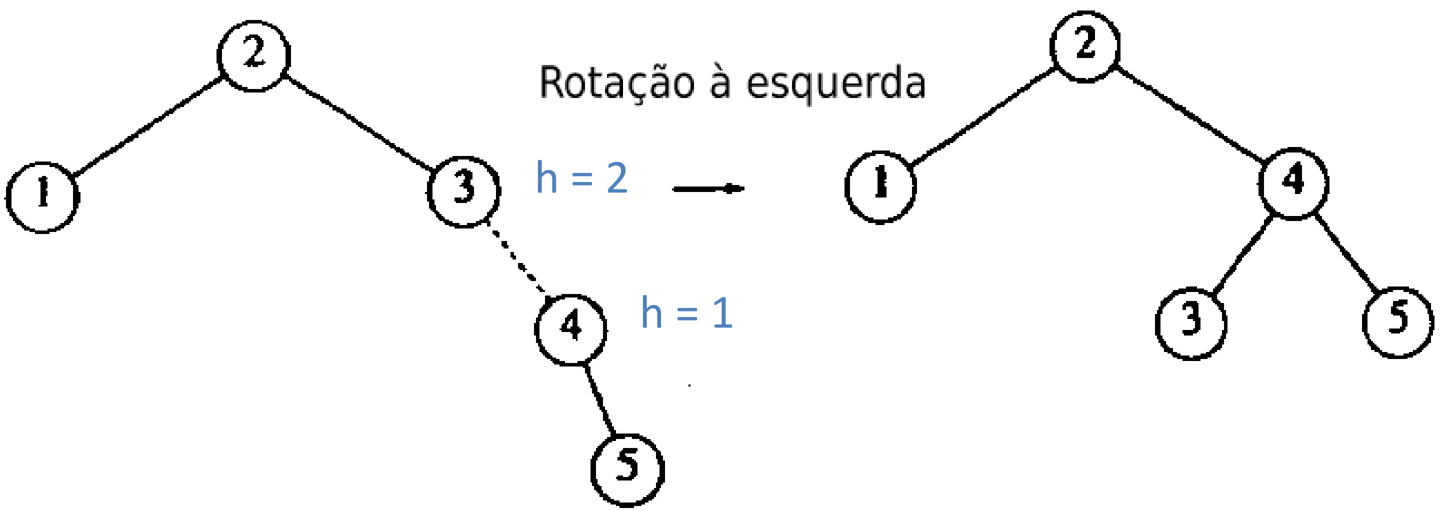
Exemplo AVL

- Inserção de 1, 2 e 3. Ao inserir 3, o nó raiz fica desbalanceado (+2)



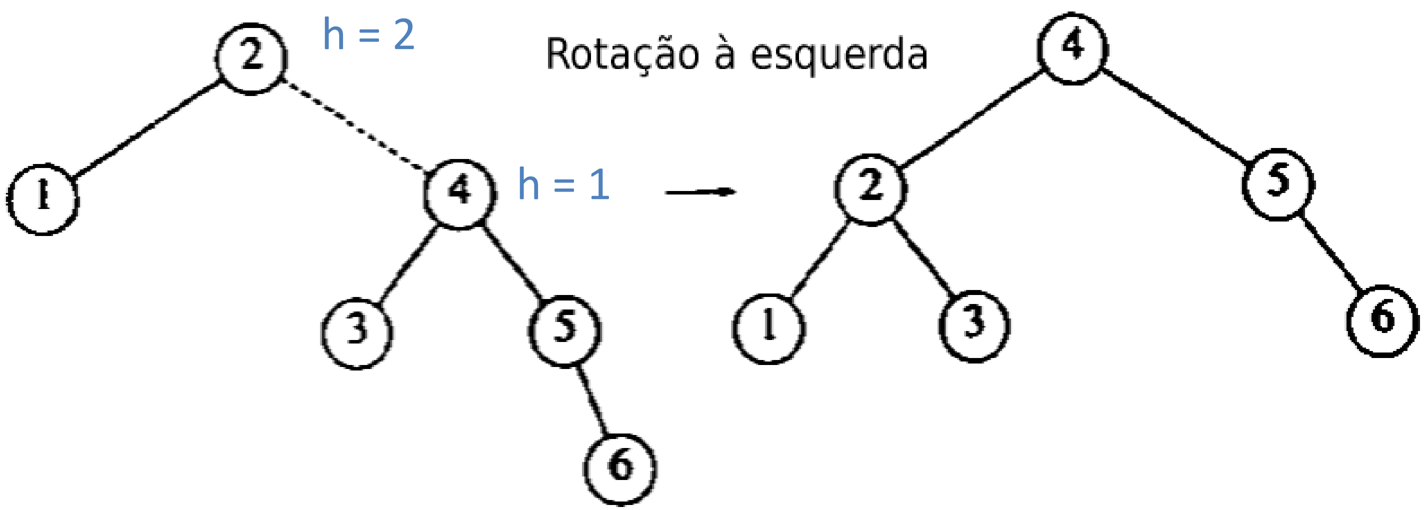
Exemplo AVL

- Inserção do 4 e 5.
  - 4: sem problemas
  - 5: desbalanceamento do nó 3 (+2)



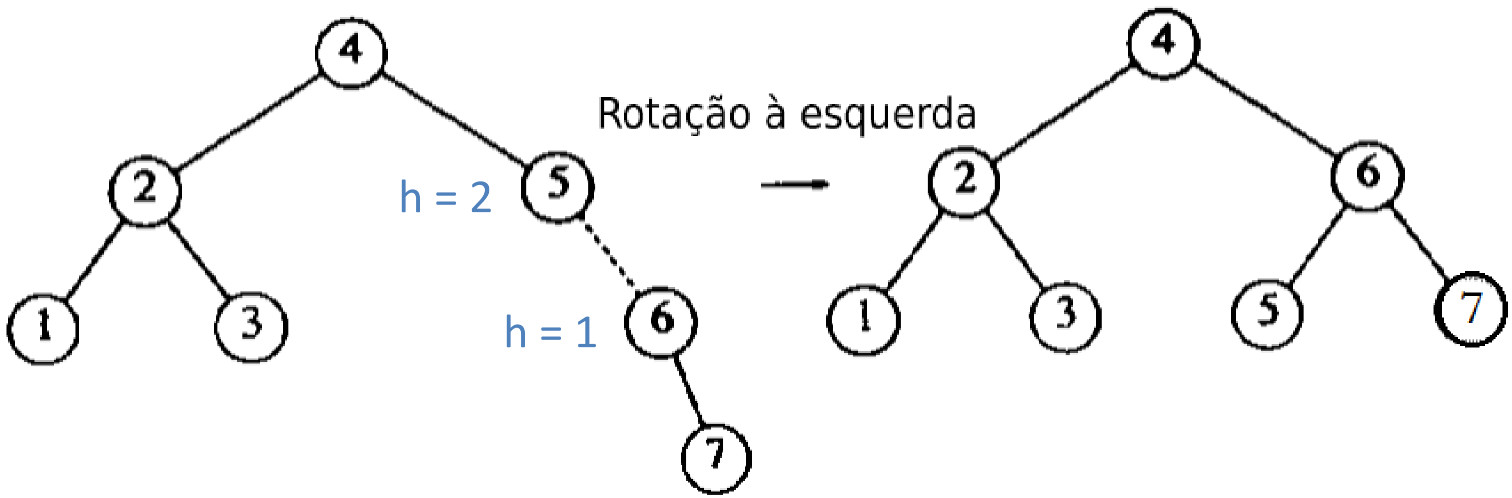
Exemplo AVL

- Inserção do 6
- Nó 2 fica desbalanceado (+2)



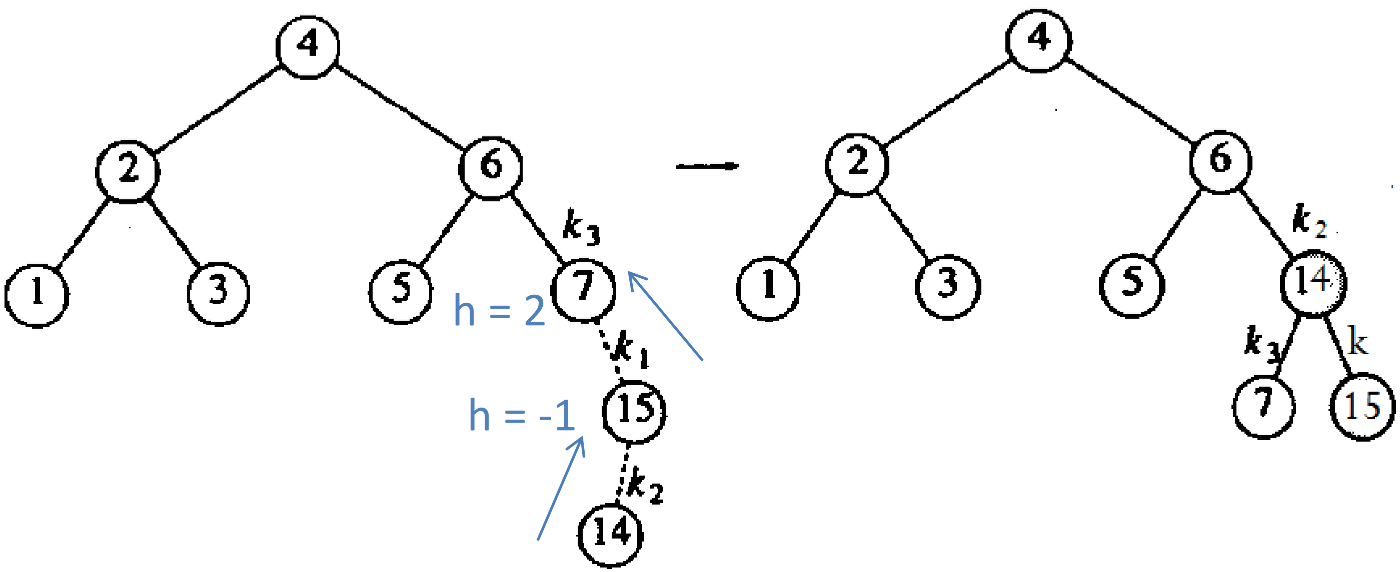
Exemplo AVL

- Inserção do nó 7
- Nó 5 fica desbalanceado (+2)



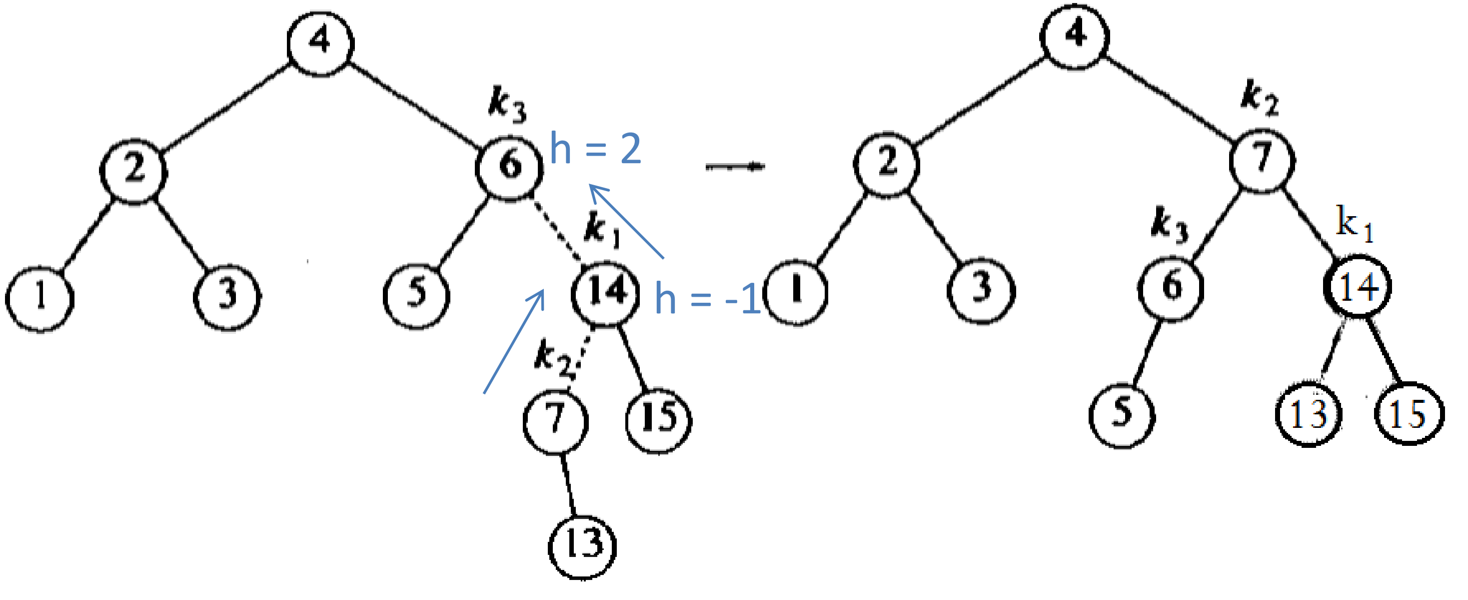
Exemplo AVL

- Inserção de 15 e 14
- 15: sem problemas
- 14: rotação dupla, 14 e 15 à direita e depois 7 e 14 à esquerda.



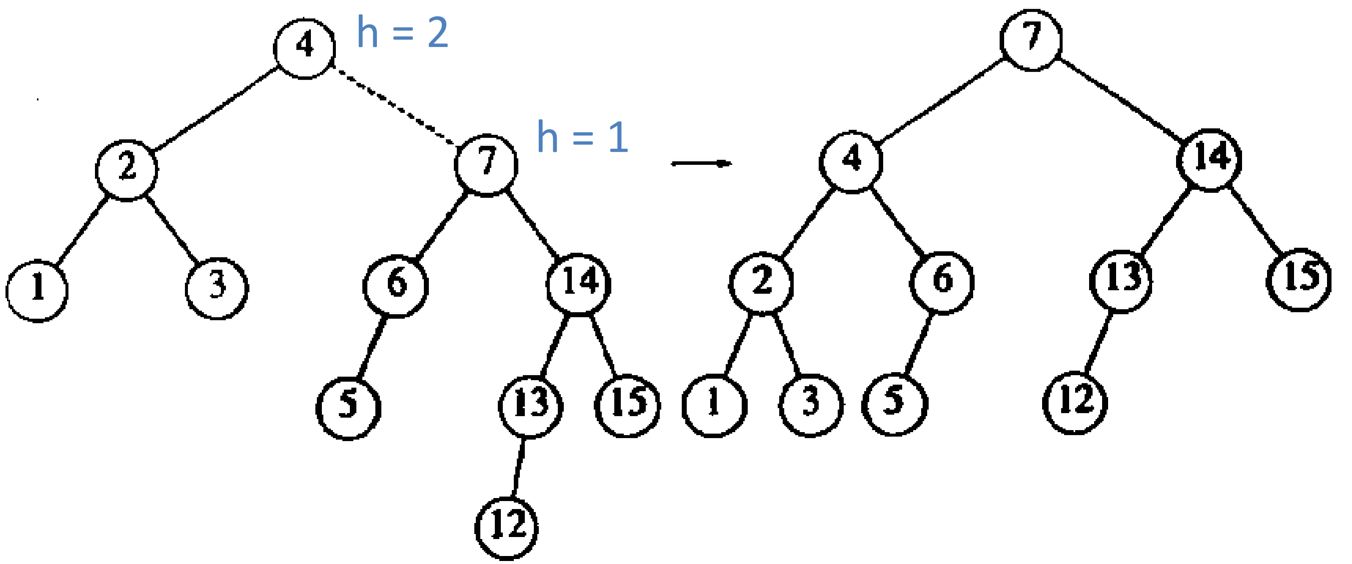
Exemplo AVL

- Inserção do 13
- Rotação do 7 e 14 à direita e rotação de 6 e 7 à esquerda.



Exemplo AVL

- Inserção do 12
- Rotação da raiz à esquerda



Exercício

1) Forneça um algoritmo não recursivo que execute um percurso de árvore em ordem. (Sugestão: usar pilha).

2) Forneça um algoritmo que imprima apenas as folhas de uma árvore.

3) Forneça um algoritmo para remoção de qualquer elemento em uma árvore binária.

4) Faça o desenho da inserção dos seguintes elementos em uma árvore binária: 50, 30, 70, 20, 40, 60, 80, 15, 25, 35, 45, 36 nesta ordem. Em seguida remova o elemento 30, como ficaria a nova árvore?

5) A operação de eliminação é comutativa, ie, a eleminação de x e depois y resulta na mesma árvore que a eliminação de y e depois x? Mostre um contra-exemplo no exercício anterior.

6) Insira as seguintes chaves em uma árvore AVL: {924, 220, 911, 244, 898, 258, 362, 363, 360, 350}.

7) Insira aleatoriamente 10000 elementos em um vetor, vetor binário, árvore binária e árvore AVL. Plote dois gráficos com o tempo:

De inserção dos 10000 elementos em cada estrutura de dados.

De busca do maior e menor elemento em cada estrutura de dados.

Se o plano não funcionou, mude o plano, não o objetivo.

Autor desconhecido