Newton

Quem se prepara, não para.

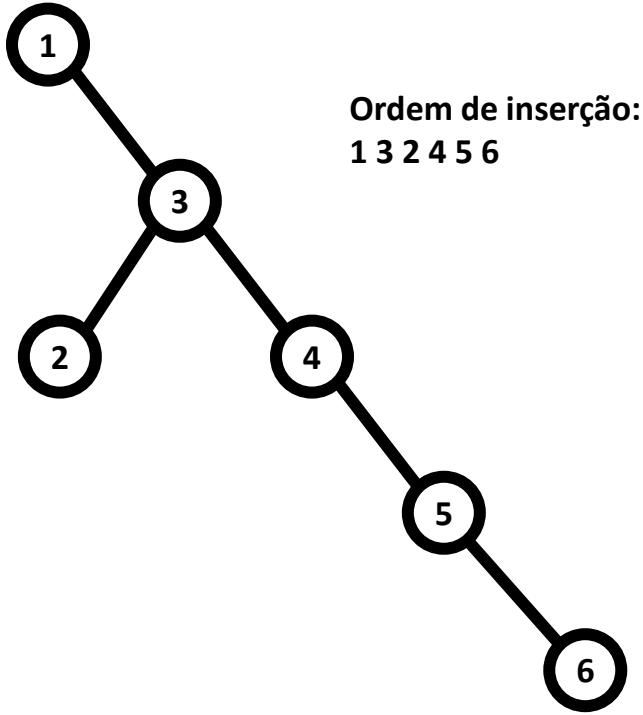
Estrutura de Dados

Professora: Michelle Hanne Soares de Andrade
michelle.andrade@newtonpaiva.br

Árvores Binárias de Pesquisa

Newton
Quem se prepara, não para.

- Pior caso para uma busca é O(n)



Ordem de inserção:
1 3 2 4 5 6

Árvores Binárias com Balanceamento

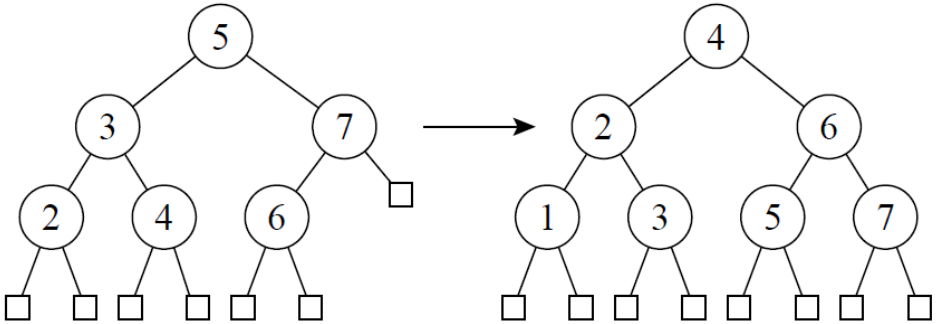
Árvore completamente balanceada ⇒ Uma árvore binária está balanceada se e somente se para todo nó, as alturas de suas duas sub-árvores diferem de no máximo 1.

Objetivo: Funções para pesquisar, inserir e retirar eficientes
$O(\lg(n))$

Problema: manter árvore completamente balanceada após cada inserção é muito caro

Árvores Binárias com Balanceamento

- Exemplo: Para inserir a chave 1 na árvore à esquerda e manter a árvore completamente balanceada precisamos movimentar todos os nós



Árvores “Quase Balanceadas”

Newton
Quem se prepara, não para.

- Solução intermediária que mantém a árvore “quase balanceada”, em vez de tentar manter a árvore completamente balanceada
- Manter a altura da árvore pequena diante de inserções e remoções arbitrárias.
- Ex: Árvores AA, AVL, Red-Black (ou SBB), Splay, Treap, etc.

Árvore SBB

Eliminação da assimetria nas árvores B binárias: árvores B binárias simétricas (*Symmetric Binary B-trees – SBB*).

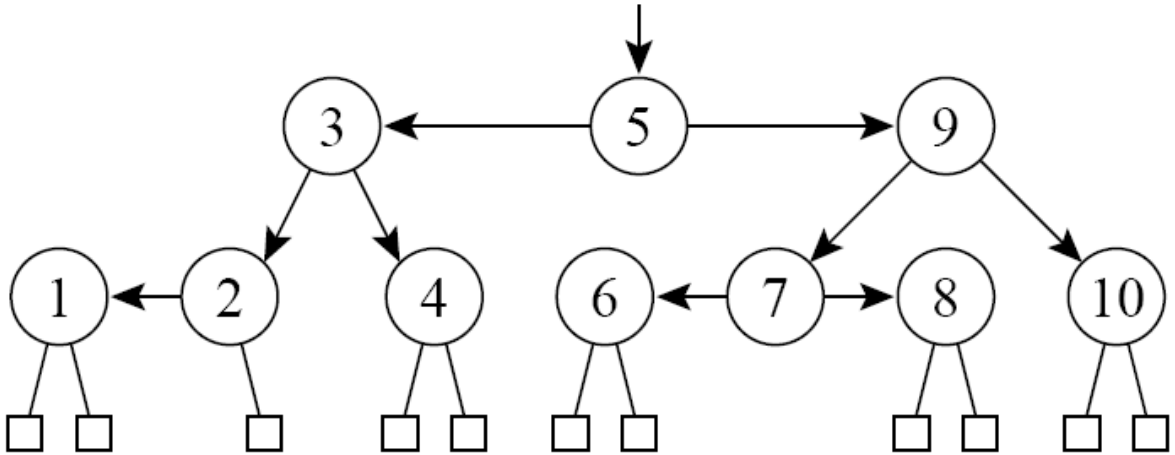
- Árvore SBB é uma **árvore binária com 2 tipos de apontadores:**

  verticais e horizontais, tal que:

  1 – todos os caminhos da raiz até cada nó externo possuem o mesmo número de apontadores verticais , e

  2 – não podem existir dois apontadores horizontais sucessivos.

Árvore SBB



Transformações para Manutenção da Propriedade SBB

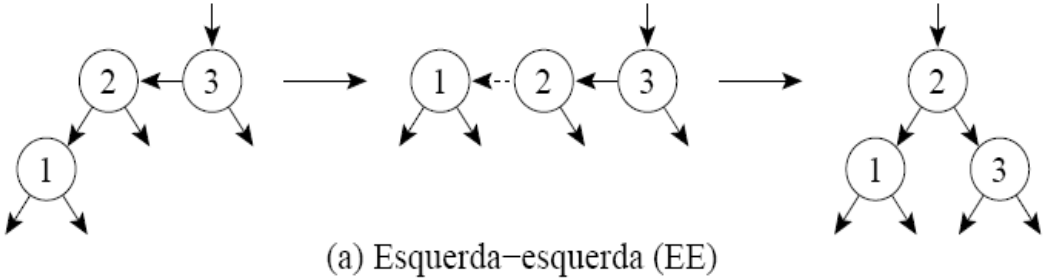
- O algoritmo para árvores SBB usa transformações locais no caminho de **inserção ou retirada** para preservar o balanceamento.
- A chave a ser inserida ou retirada é sempre inserida ou **retirada após o apontador vertical mais baixo na árvore**
- Dependendo da situação anterior à inserção ou retirada, podem, aparecer dois apontadores horizontais sucessivos
- Neste caso: é necessário realizar uma transformação

Transformações para Manutenção da Propriedade SBB

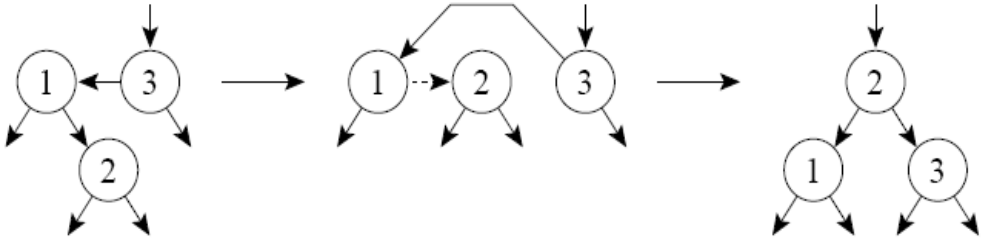
- Árvore binária de busca.
- Os ponteiros à esquerda ou direita podem ser do tipo horizontal ou vertical.
- Todos os caminhos da raiz até cada nó externo possuem o mesmo número de apontadores verticais
- Todos os nós, exceto aqueles no último nível, têm dois filhos
- Não podem existir dois apontadores horizontais sucessivos

Transformações para Manutenção da Propriedade SBB

- Transformações propostas por Bayer R. 1972



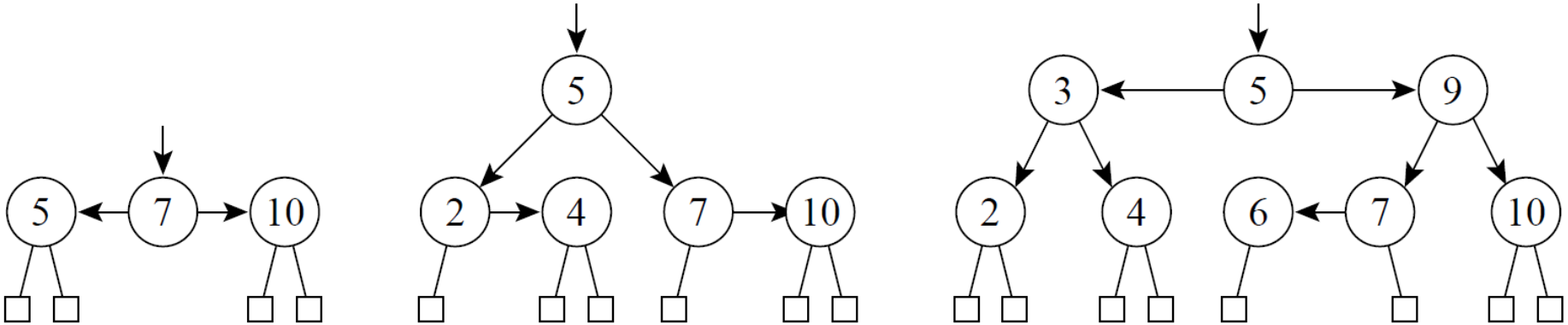
(a) Esquerda−esquerda (EE)



(b) Esquerda−direita (ED)

Exemplo de Inserção - Árvore SBB

- Inserir chaves 7, 10, 5, 2, 4, 9, 3, 6



Exemplo de Inserção - Árvore SBB

- Inserção de uma sequência de chaves em uma árvore SBB inicialmente vazia.

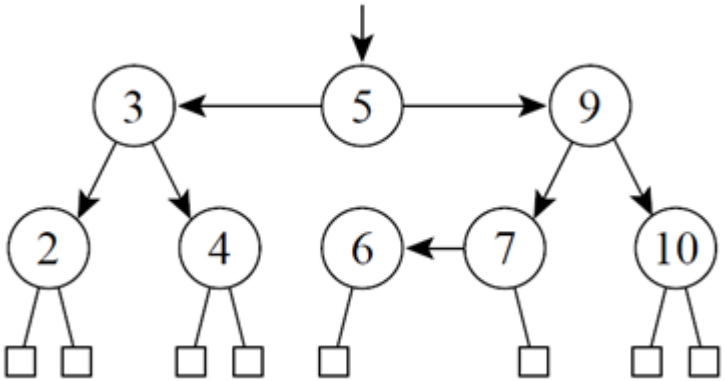
  1 - Árvore à esquerda é obtida após a inserção das chaves 7, 10, 5

  2 - Árvore do meio é obtida após a inserção das chaves 2,4 na árvore anterior

  3 - Árvore à direita é obtida após a inserção das chaves 9, 3, 6 na árvore anterior.

Exemplo de Inserção - Árvore SBB

- Inserir a chave 5.5 na árvore a seguir



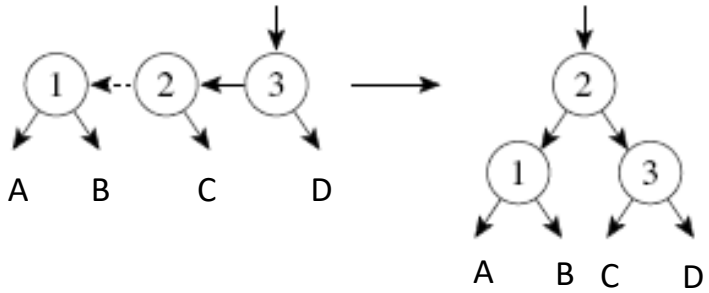
Estrutura de Dados Árvore SBB para Implementar o TAD Dicionário

- Diferenças da árvore sem balanceamento:
  - constantes $Horizontal$ e $Vertical$: representam as inclinações das referências às subárvores;
  - campo $propSBB$: utilizado para verificar quando a propriedade SBB deixa de ser satisfeita
  - campos $incE$ e $incD$: indicam o tipo de referência (horizontal ou vertical) que sai do nó.
- A operação inicializa é implementada pelo construtor da classe $ArvoreSBB$.
- As demais operações são implementadas utilizando métodos privados sobrecarregados.

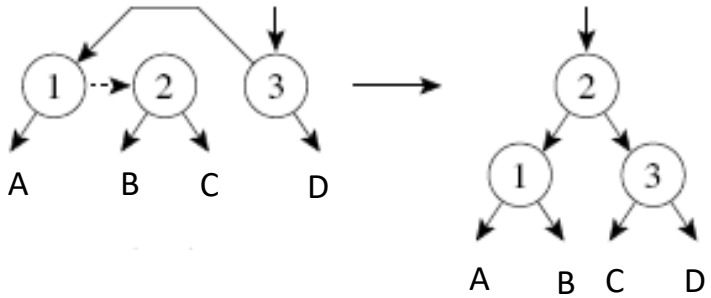
```
public class ArvoreSBB {
  private static class No {
    Item reg; No esq, dir; byte incE, incD;
  }
  private static final byte Horizontal = 0;
  private static final byte Vertical = 1;
  private No raiz; private boolean propSBB;


public ArvoreSBB () {
  this.raiz = null; this.propSBB = true;
}
public Item pesquisa (Item reg) {
  return this.pesquisa (reg, this.raiz); }
public void insere (Item reg) {
  this.raiz = insere (reg, null, this.raiz, true); }
public void retira (Item reg) {
  this.raiz = this.retira (reg, this.raiz); }
```

Procedimentos Auxiliares para Árvores SBB

```
private No ee (No ap) {
  No ap1 = ap.esq; ap.esq = ap1.dir; ap1.dir = ap;
  ap1.incE = Vertical; ap.incE = Vertical; ap = ap1;
  return ap;
}
private No dd (No ap) {
  No ap1 = ap.dir; ap.dir = ap1.esq; ap1.esq = ap;
  ap1.incD = Vertical; ap.incD = Vertical; ap = ap1;
  return ap;
}
```



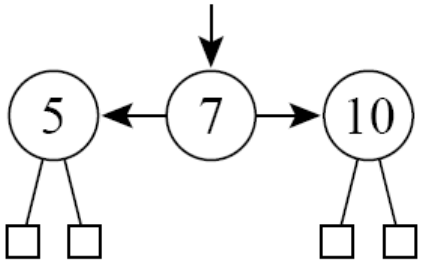
```
private No ed (No ap) {
  No ap1 = ap.esq; No ap2 = ap1.dir; ap1.incD = Vertical;
  ap.incE = Vertical; ap1.dir = ap2.esq; ap2.esq = ap1;
  ap.esq = ap2.dir; ap2.dir = ap; ap = ap2;
  return ap;
}

private No de (No ap) {
  No ap1 = ap.dir; No ap2 = ap1.esq; ap1.incE = Vertical;
  ap.incD = Vertical; ap1.esq = ap2.dir; ap2.dir = ap1;
  ap.dir = ap2.esq; ap2.esq = ap; ap = ap2;
  return ap;
}
```



Procedimentos para Inserir na Árvores SBB

- Quando a inserção é realizada:
  1. Um nó folha é inserido a partir de uma ligação vertical;
  2. Como a altura da árvore é aumentada de 1 a ligação vertical é transformada em uma ligação horizontal

```
private No insere (Item reg, No pai, No filho, boolean filhoEsq) {
  if (filho == null) {
    filho = new No (); filho.reg = reg;
    filho.incE = Vertical; filho.incD = Vertical;
    filho.esq = null; filho.dir = null;
    if (pai != null)
      if (filhoEsq) pai.incE = Horizontal; else pai.incD = Horizontal;
    this.propSBB = false;
  }
```


5
7
10

- Quando a inserção é realizada:
  3. A segunda propriedade é verificada para que as transformações sejam realizadas caso necessário;

```
else if (reg.compara (filho.reg) < 0) {
  filho.esq = insere (reg, filho, filho.esq, true);
  if (!this.propSBB)
    if (filho.incE == Horizontal) {
      if (filho.esq.incE == Horizontal) {
        filho = this.ee (filho); // transformação esquerda-esquerda
        if (pai != null)
          if (filhoEsq) pai.incE=Horizontal; else pai.incD=Horizontal;
      }
    else if (filho.esq.incD == Horizontal) {
      filho = this.ed (filho); // transformação esquerda-direita
          if (pai != null)
          if (filhoEsq) pai.incE=Horizontal;
          else pai.incD=Horizontal;
      }
    }
    else this.propSBB = true;
}
```

```
    else if (reg.compara(filho.reg) > 0) {
      filho.dir = insere(reg, filho, filho.dir, false);
      if (!this.propSBB)
        if (filho.incD == Horizontal) {
          if (filho.dir.incD == Horizontal) {
            filho = this.dd(filho); // transformação direita-direita
            if (pai != null)
              if (filhoEsq) pai.incE=Horizontal; else pai.incD=Horizontal;
          }
          else if (filho.dir.incE == Horizontal) {
            filho = this.de(filho); // transformação direita-esquerda
            if (pai != null)
              if (filhoEsq) pai.incE=Horizontal; else pai.incD=Horizontal;
          }
        }
        else this.propSBB = true;
    }
    else {
      System.out.println("Erro: Registro ja existente");
      this.propSBB = true;
    }
    return filho;
  }
```

- A verificação é feita recursivamente, em todos os nodos no caminho da inserção;

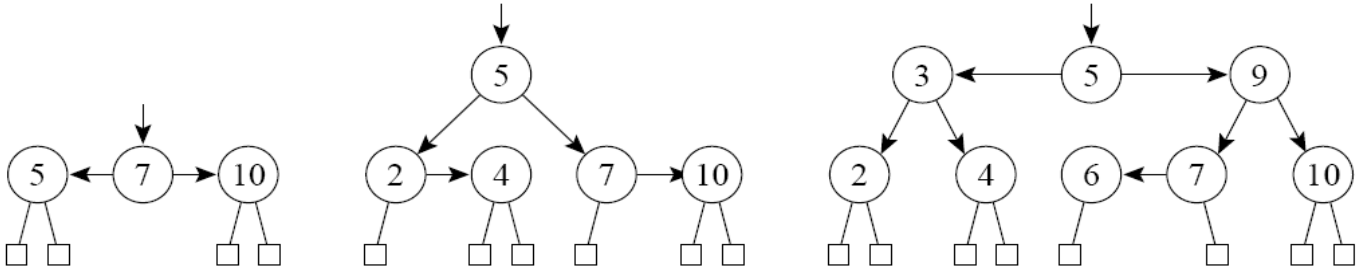
Exemplo

- Inserção de uma sequência de chaves em uma árvore SBB inicialmente vazia.

1 - Árvore à esquerda é obtida após a inserção das chaves 7, 10, 5

2 - Árvore do meio é obtida após a inserção das chaves 2,4 na árvore anterior

3 - Árvore à direita é obtida após a inserção das chaves 9, 3, 6 na árvore anterior.



Procedimento de Retirada da Árvore SBB

- Retira usa 3 procedimentos internos:
  - *EsqCurto*, *DirCurto* e *Antecessor*
- EsqCurto (DirCurto):
  - Chamado quando um nó folha que é referenciado por um apontador vertical é retirado da subárvore à direita (esquerda) tornando-a menor na altura após a retirada;
- Antecessor:
  - Quando o nó a ser retirado possui dois descendentes, o procedimento Antecessor localiza o nó antecessor para ser trocado com o nó a ser retirado

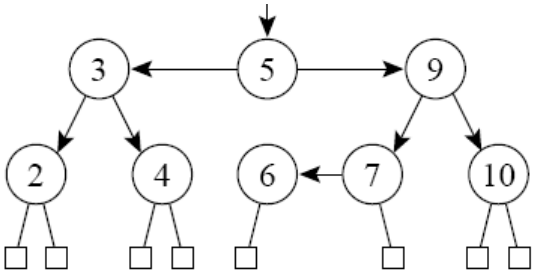
- A retirada de um elemento da árvore SBB pode ser dividida em 3 casos:

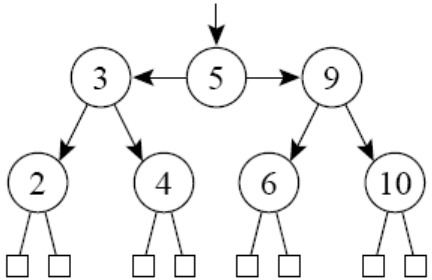
  I. Quando o nó a ser retirado possui **somente** um **filho**, ele é substituído por esse **filho** pois os dois nós são **externos** e estão no mesmo nível – não vai haver mudança no balanceamento da árvore;

```
else { // encontrou o registro
  this.propSBB = false;
(1)if (ap.dir == null) {
(3)  ap = ap.esq;
(2)  if (ap != null) this.propSBB = true;
  }
  else if (ap.esq == null) {
    ap = ap.dir;
    if (ap != null)
      this.propSBB = true;
  }
```

• Se um descendente do nó encontrado for null **(1)** e o outro não **(2)** uma substituição simples é realizada **(3)** e a propriedade da árvore é satisfeita **(2)**;

Ex.: Retirada do nó 7;



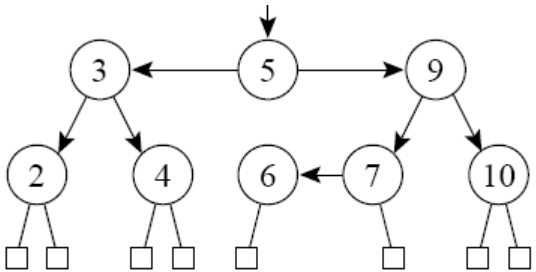


II. Quando é retirada uma folha (**dirCurto/esqCurto** é invocado):

1) Se a ligação feita pelo nó pai é **horizontal (1)** significa que pai e filho estão no mesmo nível - somente a deleção é realizada

```
// Folha direita retirada => árvore curta na altura direita
private No dirCurto (No ap) {
(1) if (ap.incD == Horizontal) {
      ap.incD = Vertical; this.propSBB = true;
    }
```

Ex.: Retirada do nó 6



2) Se a ligação feita pelo nó pai é **vertical** significa que **pai** passa a ser um nodo externo e fica a um nível acima do que deveria estar – é necessário ajuste **retirada da folha direita**

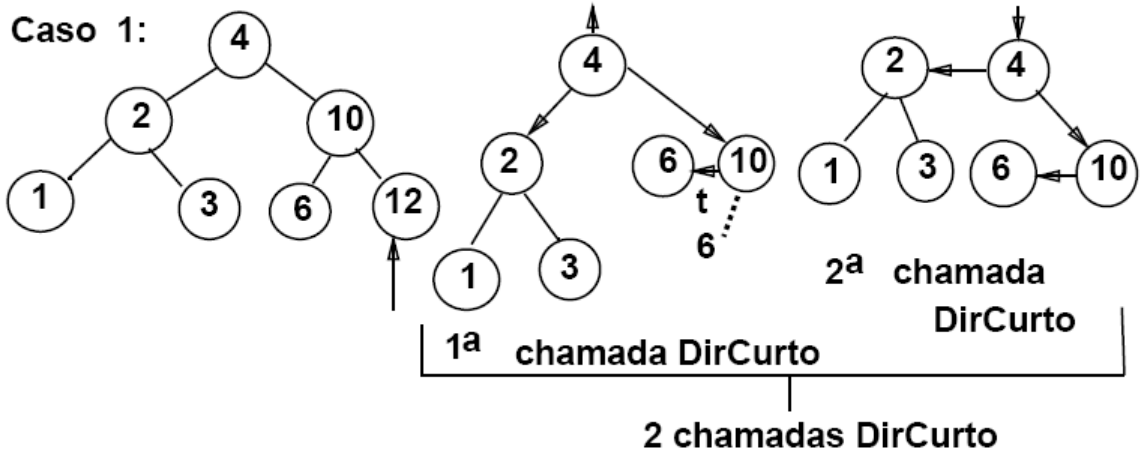
a) se a ligação esquerda (nó pai) for vertical o nó filho (esq) está um nível abaixo do que deveria estar (nó 6 após retirada de 12 no ex. 1).

Eleva-se o nível do nó filho – horizontal **(I)** – e aplica alguma transformação se necessário **(II)** e **(III)**

```
a) else {
(I)   ap.incE = Horizontal;
      if (ap.esq.incD == Horizontal) {
(II)    ap = this.ed (ap); this.propSBB = true;
      }
      else if (ap.esq.incE == Horizontal) {
(III)   ap = this.ee (ap); this.propSBB = true;
      }
   } return ap;
```

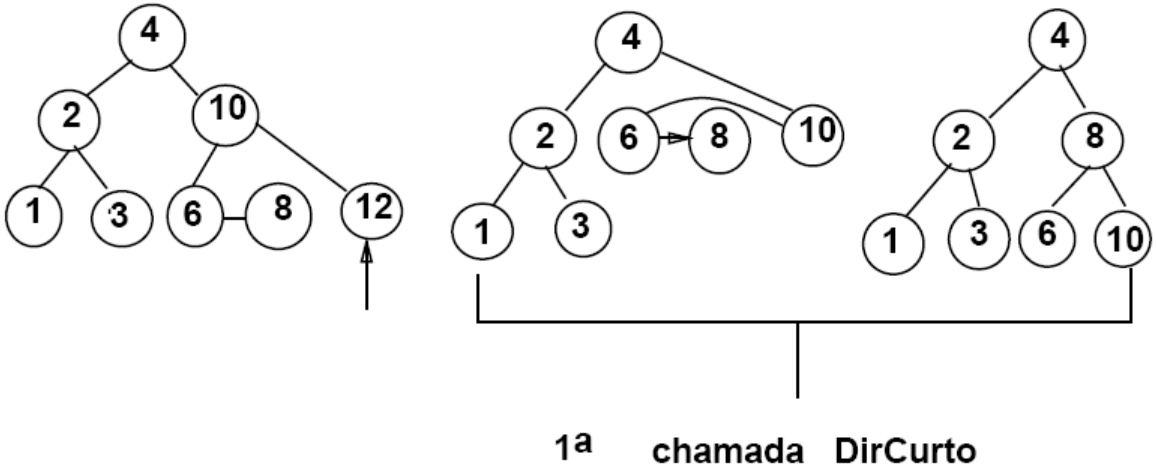
Obs.: o nó **6** do exemplo não pode ter um filho com ligação vertical pois este filho estaria a um nível abaixo da folha que retirada

Obs.: no exemplo **ap** aponta para o nó **10**;



Obs.: caso não seja feita nenhuma transformação DirCurto é invocado novamente pois a árvore ficou desbalanceada

Caso 2:

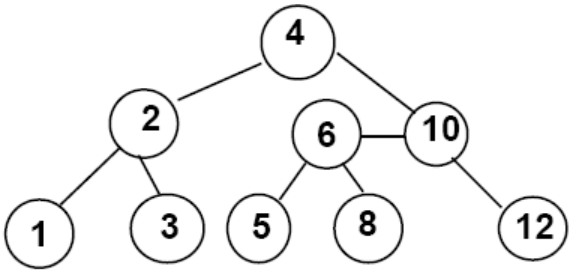


Obs.: caso uma transformação ocorra DirCurto não é invocado pois a árvore continua balanceada

(retirada da **folha direita**)

b) Se a ligação esquerda (nó pai) for **horizontal** significa que o nó **filho (esq)** está no nível em que deveria estar após a retirada da **folha (dir)**;

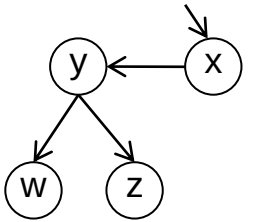
É certo que esse nó filho possui **dois descendentes verticais** e que são nós externos (nó 6);

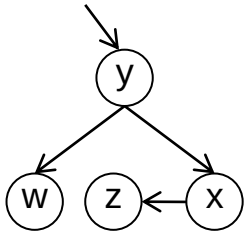


É realizada a modificação dos **nós** mostrada na figura abaixo **(IV)**;

Se alguma transformação for necessária **(V)** mais um nível será acrescentado na subárvore (nó 6 do exemplo 4);

A aresta referente à raiz se torna horizontal **(VI)**;

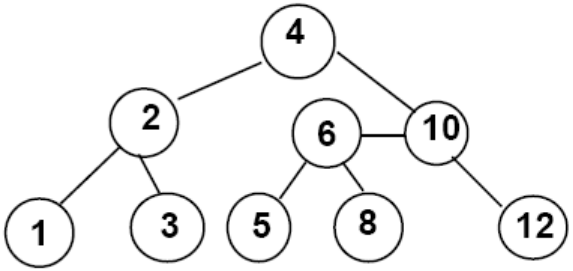


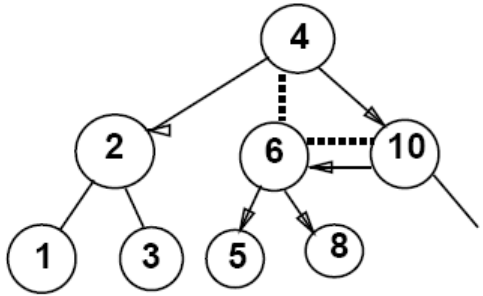


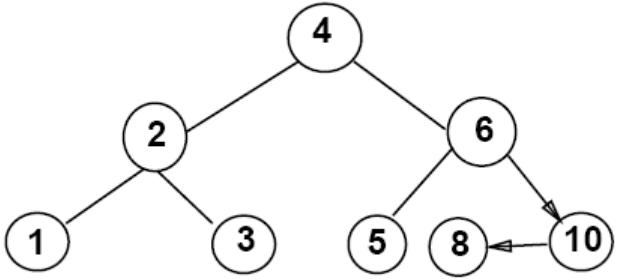

```
b) else if (ap.incE == Horizontal) {
(IV) No ap1 = ap.esq; ap.esq = ap1.dir; ap1.dir = ap; ap = ap1;
(V)  if (ap.dir.esq.incD == Horizontal) {
       ap.dir = this.ed (ap.dir); ap.incD = Horizontal; (VI)
     }
(V)  else if (ap.dir.esq.incE == Horizontal) {
       ap.dir = this.ee (ap.dir); ap.incD = Horizontal; (VI)
     }
     this.propSBB = true;
   }
```

Obs.: após a transformação **ap** está apontando para y;

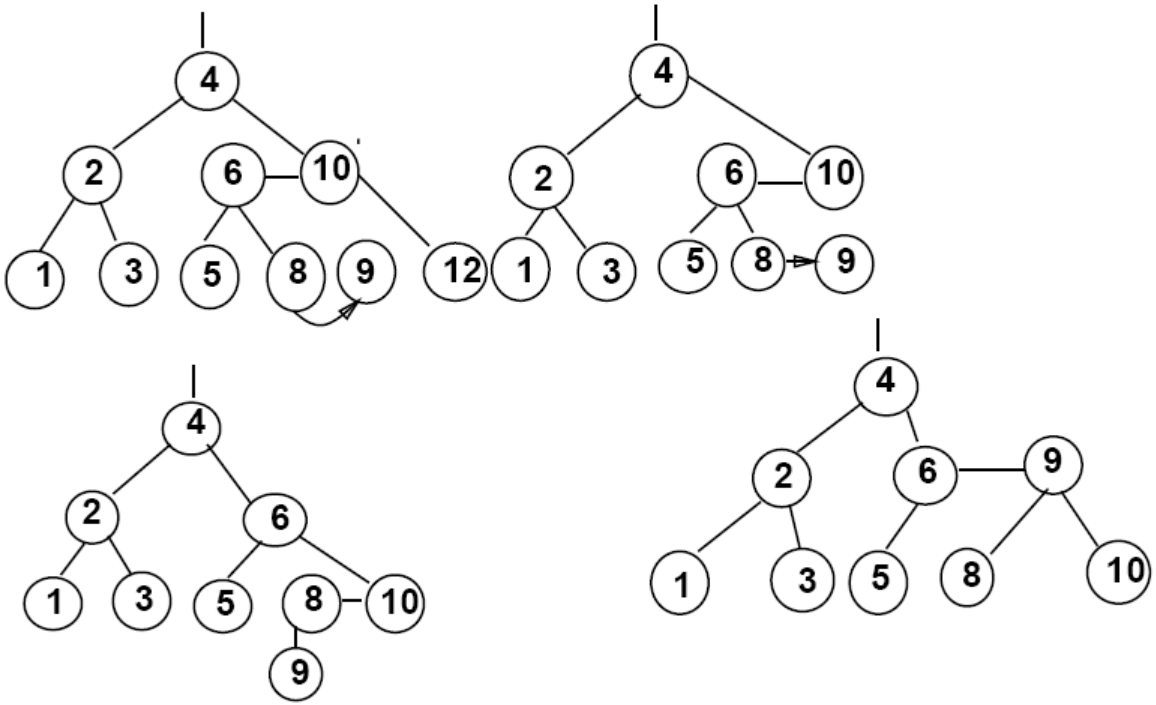
Caso 3:







Caso 4



Obs.: o nó 9 está entre o 8 e o 10;

III. Quando um **nodo interno** é retirado é realizada a sua substituição por outro nó, como na árvore binária de pesquisa tradicional;

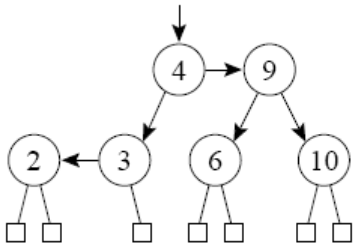
- Se for um nó com somente um filho tratar como no **caso I**;

retirar o **nó 4** e substituí-lo por **3** no exemplo 1;

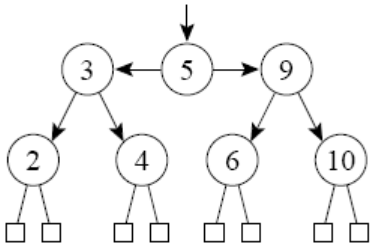
- Se for uma folha tratar como no **caso II**;

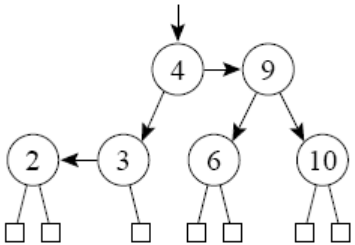
retirar o **nó 5** e substituí-lo por **4** no exemplo 2;

Ex. 1



Ex. 2



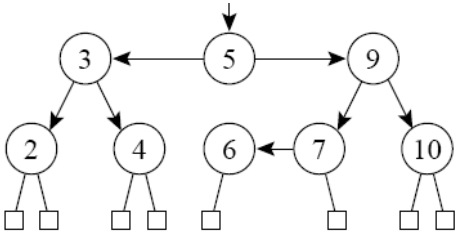


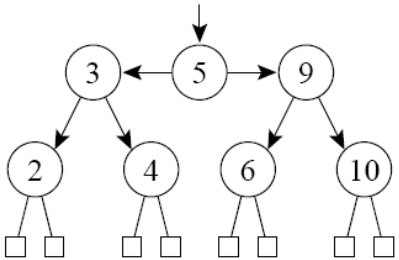
- Resultado obtido quando se retira uma sequência de chaves da árvore SBB mais à direita:

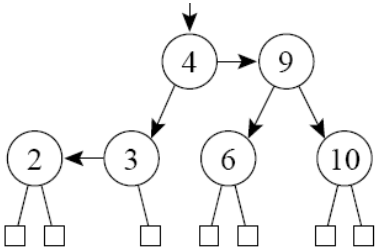
  1- A árvore à esquerda é obtida após a retirada da chave ***7*** da árvore acima;

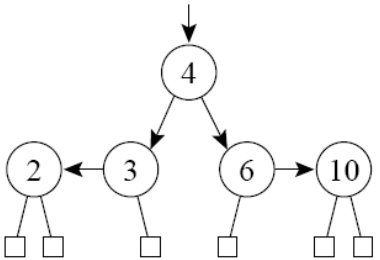
  2- A árvore do meio é obtida após a retirada da chave ***5*** da árvore anterior;

  3- A árvore à direita é obtida após a retirada da chave ***9*** da árvore anterior;









Análise

- Nas árvores SBB é necessário distinguir dois tipos de **alturas**

  1. altura **vertical *h*** : necessária para manter a altura uniforme e obtida através da contagem do número de apontadores ***verticais*** em qualquer caminho entre a raiz e um nó externo

  2. **altura *k*:** representa o número máximo de comparações de chaves obtido através da contagem do número total de apontadores no maior caminho entre a raiz e um nó externo

- A altura ***k*** é maior que a altura ***h*** sempre que existirem apontadores horizontais na árvore;

- Bayer (1972) mostrou que:

$$\log(n+1) \leq k \leq 2\log(n+2) - 2$$

- Custo para manter a propriedade SBB: custo para percorrer o caminho de pesquisa para encontrar a chave, seja para inserí-la ou para retirá-la
  - Logo: o custo e *O(log n)*
- Número de comparações em uma pesquisa com sucesso na árvore SBB:
  - Melhor caso: C(n) = O(1)
  - Pior caso: C(n) = O(log n)
  - Caso médio: C(n) = O(log n)
- **Obs:** Na prática, o caso médio para C(n) é apenas cerca de 2% pior que o C(n) para uma árvore completamente balanceada, conforme mostrado em Ziviani e Tompa (1982)

Referências

- NICOLETTI, Maria do Carmo, HRUSCHKA, Estevam R. Jr.. **Fundamentos da teoria dos grafos para computação**, - 3. ed. - Rio de Janeiro : LTC, 2018.
- ZIVIANI, N. **Projeto de Algoritmos com Implementações em Java e C++** . Consultoria em Java e C++ de F.C. Botelho, Cengage Learning Brasil, ISBN 9788522108213, 2012.