Algoritmos e Estruturas de Dados

árvores

2010-2011

Carlos Lisboa Bento

Árvores

conceitos

LISTAS LINEARES

Acesso aos seus elementos é feito sequencialmente
(num ou nos dois sentidos consoante se trate de listas simples ou duplas).

INCONVENIENTE: por exemplo na pesquisa de informação.

ÁRVORES / ÁRVORES BINÁRIAS

Uma possível solução…

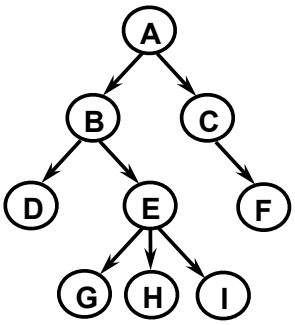
Árvores
conceitos

Árvore:

Um nó c é a RAIZ.

Todos os nós excepto c têm exactamente uma ligação para eles a partir de um nó p. p é o pai de c, c é o filho de p.

Existe um caminho único da raiz c para cada nó. O número de ligações que têm de ser seguidas é o comprimento do caminho.

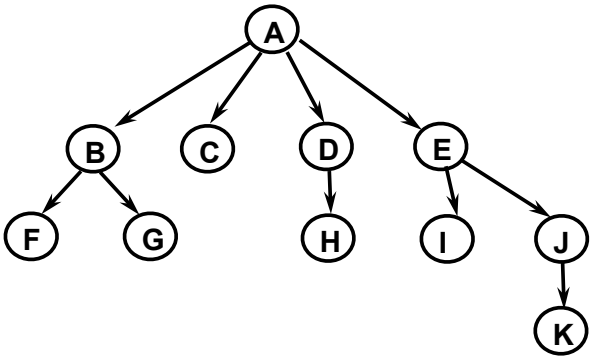


Árvores
conceitos

Altura de um nó (h) - comprimento do caminho mais longo entre esse nó e as folhas da árvore (incluindo o próprio).

Profundidade de um nó (d) - comprimento do caminho da raiz até esse nó.

Tamanho de um nó (s) - número de descendentes desse nó mais ele próprio



Nó	Altura	Profund.	Tam.
A	4	0	11
B	2	1	3
C	1	1	1
D	2	1	2
E	3	1	4
F	1	2	1
G	1	2	1
H	1	2	1
I	1	2	1
J	2	2	2
K	1	3	1

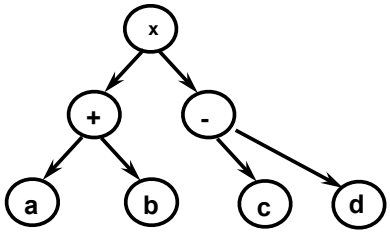
Árvores
aplicações

Apl #1

Organização de dados, por exemplo que contêm vários níveis de informação (por exemplo, um edifício tem salas, uma sala tem objectos, os objectos podem ter outros objectos…)

Apl #2

Árvores de expressões. O valor de um nó é o resultado de aplicar o operador representado nesse nó utilizando como operandos os filhos desse nó



Apl #3

Organizações taxonómicas de dados (por exemplo taxonomia de Lineu, "família, género, espécie…")

Árvores Binárias
conceitos

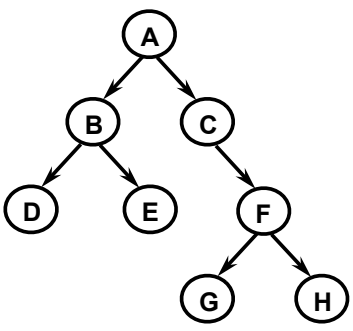
Árvore binária:

conjunto finito de elementos que

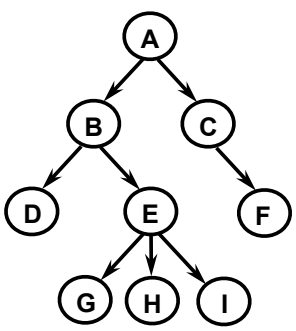
ou está vazio

ou contém um elemento chamado raiz da árvore ligado a dois conjuntos disjuntos em que cada um é, por sua vez uma árvore binária.

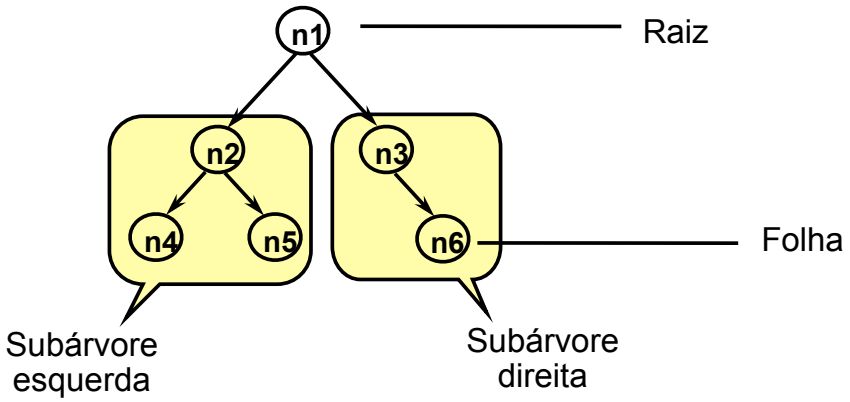
Árvore binária



Árvore não binária



Árvores Binárias
conceitos

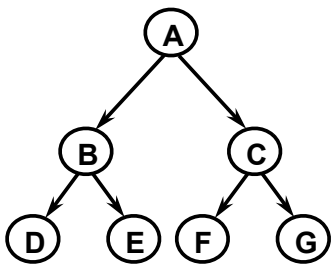


- Dois nós são irmãos se são os filhos direito e esquerdo do mesmo pai;
- Um nó que não tem filhos é uma folha;
- No exemplo acima, n4 e n5 são descendentes de n2 e este é ascendente de ambos.

Árvores Binárias
conceitos

- Nível de uma árvore binária:
  - Nível da raiz = 0;
  - Nível de qualquer outro nó = nível do pai + 1.

- Árvore binária completa de nível n é uma árvore em que cada nó de nível n é uma folha e em que todos os nós de nível menor do que n têm subárvores direita e esquerda não vazias



Árvore binária completa de nível 2

Árvores Binárias
conceitos

Travessia de uma árvore

- **Em muitos problemas é necessário percorrer uma árvore binária, atravessando todos os seus nós e enumerando-os;**
- **Numa árvore não há uma ordem "natural" de percurso, como, por exemplo, nas listas ligadas;**
- **Designa-se por *visitar um nó* a acção de atingir um dado nó de uma árvore e efectuar (ou não) uma qualquer operação com a informação nele armazenada;**
- **Vamos definir três métodos para atravessar uma árvore:**
  - ***pré-ordem*, *ordem*, *pos-ordem***
- **Utiliza-se uma definição recursiva, pelo que percorrer uma árvore implicará percorrer a raiz e percorrer as suas subárvores esquerda e direita.**

Árvores Binárias
conceitos

Travessia de uma árvore (cont.)

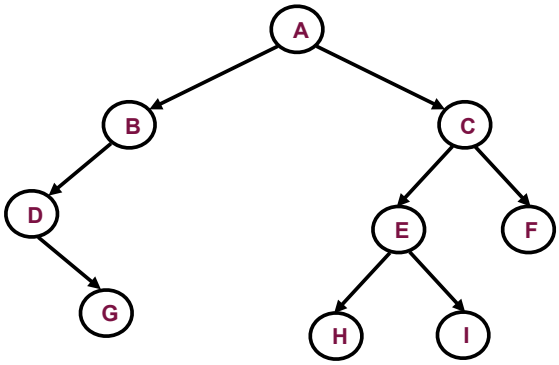
- Em pré-ordem (ou pré-fixado)
  1) Visitar a raiz;
  2) Atravessar a subárvore esquerda em pré-ordem;
  3) Atravessar a subárvore direita em pré-ordem.

- Em ordem (ou central)
  1) Atravessar a subárvore esquerda em ordem;
  2) Visitar a raiz;
  3) Atravessar a subárvore direita em ordem.

- Em pos-ordem (ou pos-fixado)
  1) Atravessar a subárvore esquerda em pos-ordem;
  2) Atravessar a subárvore direita em pos-ordem.
  3) Visitar a raiz;

Árvores Binárias
conceitos

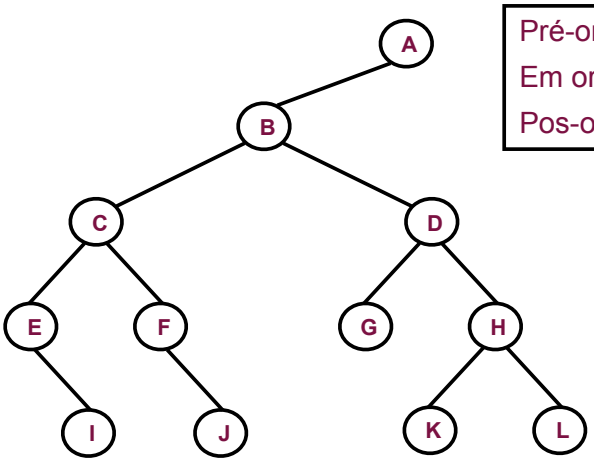
Travessia de uma árvore (exemplo)



Pré-ordem: **A B D G C E H I F**
Em ordem: **D G B A H E I C F**
Pos-ordem: **G D B H I E F C A**

Árvores Binárias
conceitos

Travessia de uma árvore (outro exemplo...)



Pré-ordem: **A B C E I F J D G H K L**
Em ordem: **E I C F J B G D K H L A**
Pos-ordem: **I E J F C G K L H D B A**

Árvores Binárias
implementação

```
final class BinaryNode
{
   BinaryNode( )
   { this( null ); }

   BinaryNode( Object theElement )
   {  this( theElement, null, null ); }

   BinaryNode( Object theElement,
                  BinaryNode lt, BinaryNode rt )
   {
      element = theElement;
      left    = lt;
      right   = rt;
   }

  static int size( BinaryNode t )
   {
      if( t == null )
         return 0;
      else
         return 1 + size( t.left ) + size( t.right );
   }

   static int height( BinaryNode t )
   {
      if( t == null )
         return -1;
      else
         return 1 +
           Math.max( height( t.left ), height( t.right ) );
   }
```

```
void printPreOrder( )
{
   System.out.println( element );       // Node
   if( left != null )
      left.printPreOrder( );          // Left
   if( right != null )
      right.printPreOrder( );         // Right
}


void printPostOrder( )
{
   if( left != null )
      left.printPostOrder( );         // Left
   if( right != null )
      right.printPostOrder( );        // Right
   System.out.println( element );       // Node
}

void printInOrder( )
{
   if( left != null )
      left.printInOrder( );           // Left
   System.out.println( element );       // Node
   if( right != null )
      right.printInOrder( );          // Right
}
```


Árvores Binárias
implementação

```
  BinaryNode duplicate( )
  {
     BinaryNode root = new BinaryNode( element );

     if( left != null )           // If there's a left subtree
        root.left = left.duplicate( );    // Duplicate; attach
     if( right != null )          // If there's a right subtree
        root.right = right.duplicate( );  // Duplicate; attach
     return root;                    // Return resulting tree
  }


  // Friendly data; accessible by other package routines
  Object     element;
  BinaryNode left;
  BinaryNode right;
}
```

Aqui a referência para o nó não é passada como um parâmetro - é usado element, left e right que são declarados friendly na classe BinaryNode (comparar com o método size)

Recordar ainda que este método pertence à classe BinaryNode

Árvores Binárias
implementação

… ainda sobre os métodos de travessia definidos nesta classe

```
public void printPreOrder( )
   {
      if( root != null )
         root.printPreOrder( );
   }


------
void printPreOrder( )
   {
      System.out.println( element );
      if( left != null )
         left.printPreOrder( );
      if( right != null )
         right.printPreOrder( );
   }
--------
```

```
public void printInOrder( )
   {
      if( root != null )
         root.printInOrder( );
   }


-----
void printInOrder( )
   {
      if( left != null )
         left.printInOrder( );
      System.out.println( element );
      if( right != null )
         right.printInOrder( );
   }
------
```

```
public void printPostOrder( )
   {
      if( root != null )
         root.printPostOrder( );
   }


-------
void printPostOrder( )
   {
      if( left != null )
         left.printPostOrder( );
      if( right != null )
         right.printPostOrder( );
      System.out.println( element );
   }
-------
```

quantas chamadas dos métodos respectivos?

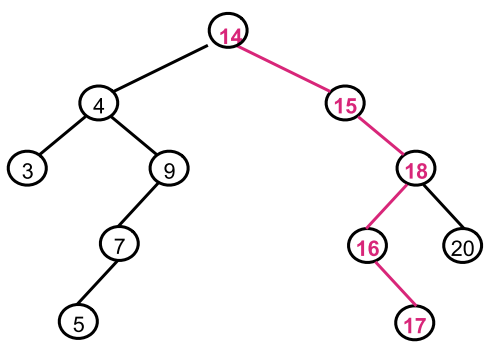
número de chamadas igual ao número de nós **O(N)**

Árvores Binárias de Pesquisa
conceitos

A definição de árvore apresentada anteriormente não apresenta qualquer restrição acerca do posicionamento dos nós dentro da árvore.

EM CONSEQUÊNCIA: a pesquisa de um qualquer elemento teria que passar por todos os nós da árvore (tal como no caso das listas ligadas).

SOLUÇÃO: impôr que, para qualquer nó da árvore, todos os elementos da sua subárvore esquerda sejam menores que ele e que todos os elemento da sua subárvore direita sejam maiores que ele



Por exemplo, para pesquisar o elemento 17 na árvore da figura seria apenas necessário percorrer os elementos assinalados:

Operações find, insert e remove c/ compl. O(log N) SE FIZERMOS ALGUMAS ALTERAÇÕES, SENÃO temos O(N) !!!

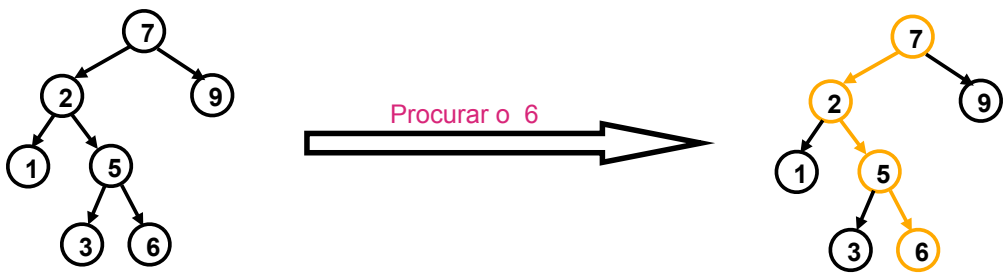
Árvores Binárias de Pesquisa
conceitos

```
package DataStructures;

import Supporting.*;
import Exceptions.*;
import Supporting.Comparable;

// ******************PUBLIC OPERATIONS*********************
// void insert( x )       --> Insert x
// void remove( x )       --> Remove x
// void removeMin( )      --> Remove smallest item
// Comparable find( x )   --> Return item that matches x
// Comparable findMin( )  --> Return smallest item
// Comparable findMax( )  --> Return largest item
// boolean isEmpty( )     --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// void printTree( )      --> Print tree in sorted order
// ******************ERRORS********************************
// Most routines throw ItemNotFound on
// various degenerate conditions
// insert throws DuplicateItem if item is already
// in the tree

// @author Mark Allen Weiss
```

```
public interface SearchTree
{
   void      insert( Comparable x )
                   throws DuplicateItem;

   void      remove( Comparable x )
                   throws ItemNotFound;

   void      removeMin( )
                   throws ItemNotFound;

   Comparable findMin( )
                   throws ItemNotFound;

   Comparable findMax( )
                   throws ItemNotFound;

   Comparable find( Comparable x )
                   throws ItemNotFound;

   void      makeEmpty( );

   boolean   isEmpty( );

   void      printTree( );
}
```

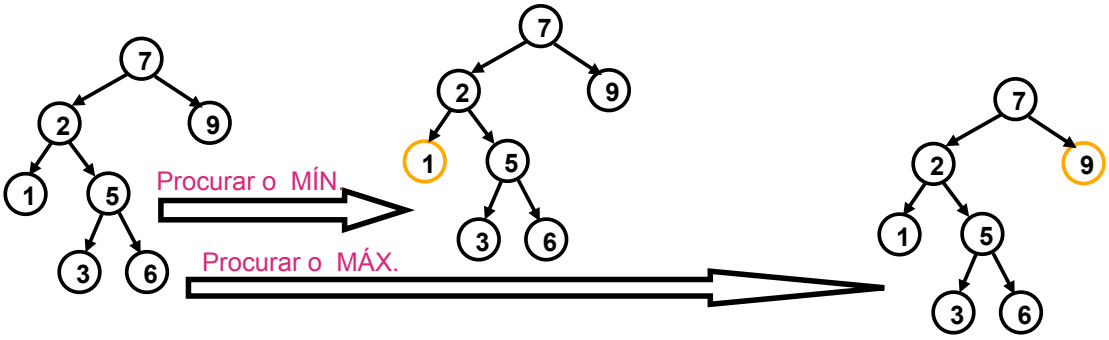

Árvores Binárias de Pesquisa
conceitos

find(x) --> Return item that matches x



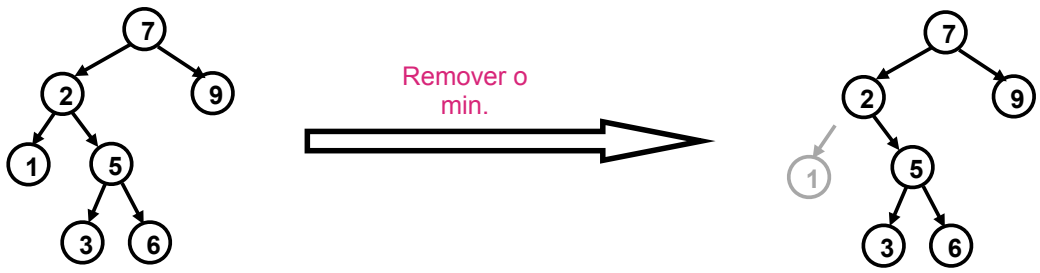
Árvores Binárias de Pesquisa
conceitos

findMin( ) --> Return smallest item
findMax( ) --> Return largest item



Árvores Binárias de Pesquisa
conceitos

void removeMin( ) --> Remove smallest item

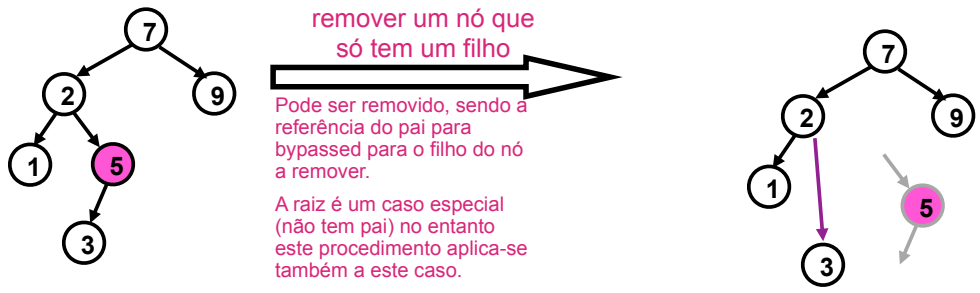


Árvores Binárias de Pesquisa
conceitos

void remove(x) **--> Remove x**

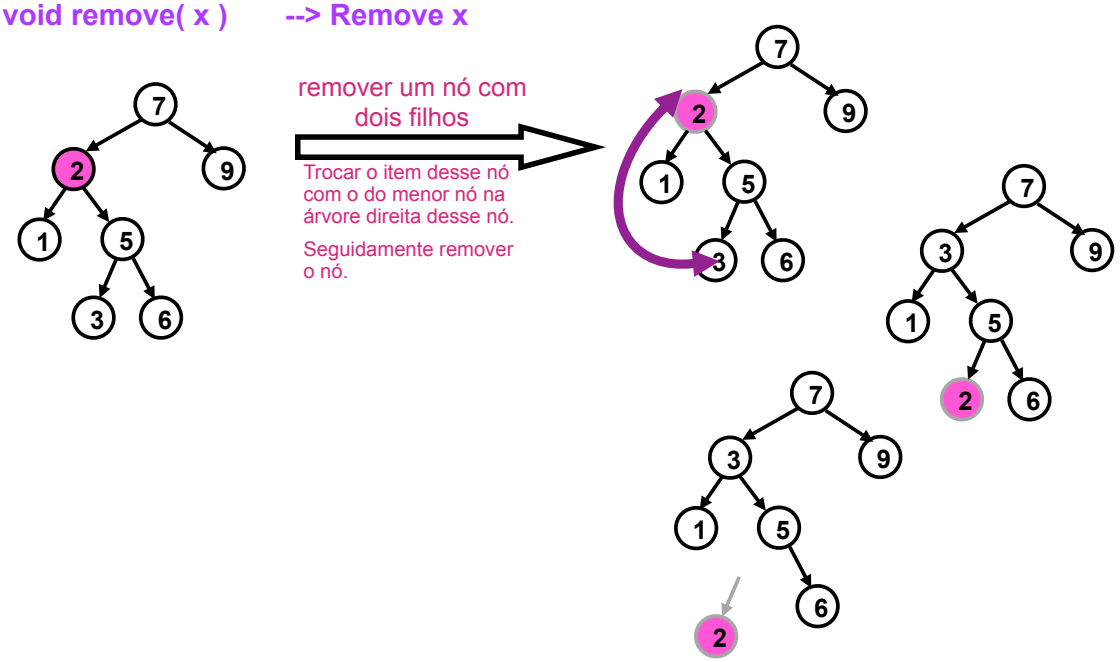
Árvores Binárias de Pesquisa
conceitos

void remove(x) **--> Remove x**



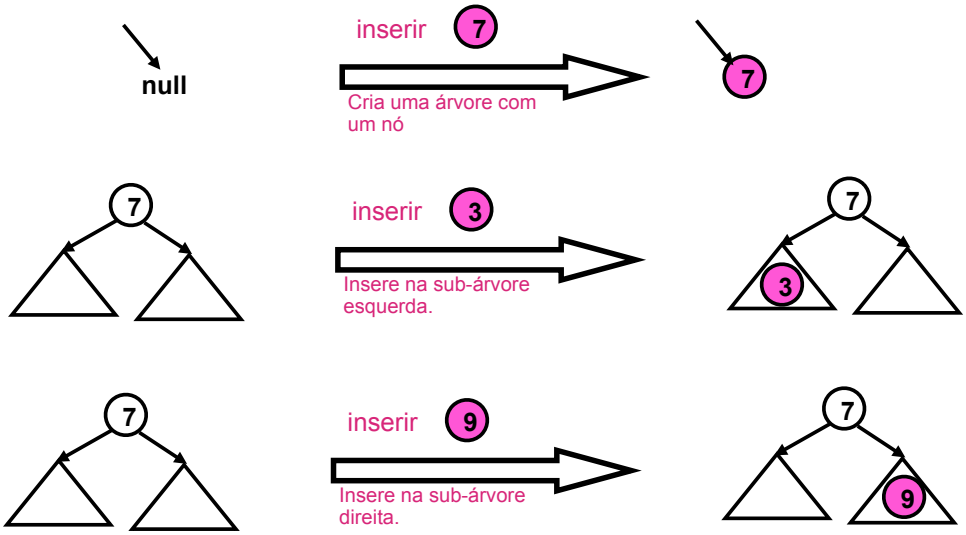
Árvores Binárias de Pesquisa
conceitos

void remove(x) **--> Remove x**



Árvores Binárias de Pesquisa
conceitos

void insert(x) **--> Insert x**



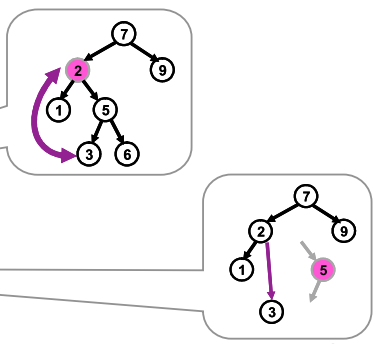
Árvores Binárias de Pesquisa
implementação

```
package DataStructures;

import Supporting.*;
import Exceptions.*;
import Supporting.Comparable;

// @author Mark Allen Weiss

public class BinarySearchTree
                    implements SearchTree
{
   public BinarySearchTree( )
   { root = null; }

   public void insert( Comparable x )
                        throws DuplicateItem
   { root = insert( x, root );  }

   public void remove( Comparable x )
                        throws ItemNotFound
   { root = remove( x, root ); }

   public void removeMin( )
                        throws ItemNotFound
   { root = removeMin( root ); }
```

```
public Comparable findMin( ) throws ItemNotFound
   { return findMin( root ).element; }

public Comparable findMax( ) throws ItemNotFound
   { return findMax( root ).element; }

public Comparable find( Comparable x )
                                    throws ItemNotFound
   { return find( x, root ).element ; }

public void makeEmpty( )
   { root = null; }

public boolean isEmpty( )
   { return root == null; }

public void printTree( )
   {
      if( root == null )
         System.out.println( "Empty tree" );
      else
         printTree( root );
   }
```


Árvores Binárias de Pesquisa
implementação

```
protected BinaryNode insert( Comparable x, BinaryNode t ) throws DuplicateItem
   {  if( t == null )
         t = new BinaryNode( x, null, null );
      else if( x.compares( t .element ) < 0 )
         t.left = insert( x, t.left );
      else if( x.compares( t.element ) > 0 )
         t.right = insert( x, t.right );
      else
         throw new DuplicateItem( "SearchTree insert" );
      return t;   }

protected BinaryNode remove( Comparable x, BinaryNode t ) throws ItemNotFound
   {  if( t == null )
         throw new ItemNotFound( "SearchTree remove" );
      if( x.compares( t.element ) < 0 )
         t.left = remove( x, t.left );
      else if( x.compares( t.element ) > 0 )
         t.right = remove( x, t.right );
      else if( t.left != null && t.right != null ) // Two children
      {
         t.element = findMin( t.right ).element;
         t.right = removeMin( t.right );
      }
      else
         t = ( t.left != null ) ? t.left : t.right;
      return t;  }
```



Árvores Binárias de Pesquisa
implementação

```
protected BinaryNode removeMin( BinaryNode t ) throws ItemNotFound
   {
      if( t == null )
         throw new ItemNotFound( "SearchTree removeMin" );
      if( t.left != null )
         t.left = removeMin( t.left );
      else
         t = t.right;
      return t;
   }

protected BinaryNode findMin( BinaryNode t ) throws ItemNotFound
   {
      if( t == null )
         throw new ItemNotFound( "SearchTree findMin" );

      while( t.left != null )
         t = t.left;
      return t;
   }

protected BinaryNode findMax( BinaryNode t ) throws ItemNotFound
   {
      if( t == null )
         throw new ItemNotFound( "SearchTree findMax" );

      while( t.right != null )
         t = t.right;
      return t;
   }
```

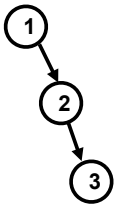

Árvores Binárias de Pesquisa
implementação

```
protected BinaryNode find( Comparable x, BinaryNode t ) throws ItemNotFound
   {
      while( t != null )
         if( x.compares( t.element ) < 0 )
            t = t.left;
         else if( x.compares( t.element ) > 0 )
            t = t.right;
         else
            return t;    // Match

      throw new ItemNotFound( "SearchTree find" );
   }

protected void printTree( BinaryNode t )
   {
      if( t != null )
      {
         printTree( t.left );
         System.out.println( t.element.toString( ) );
         printTree( t.right );
      }
   }

    /** The tree root. */
   protected BinaryNode root;
}
```

Árvores Binárias de Pesquisa
conceitos

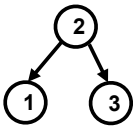
Árvores que podem resultar da inserção de permutações dos elementos 1, 2 e 3:

1 ; 2 ; 3

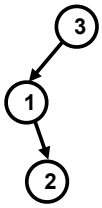


1 ; 3 ; 2

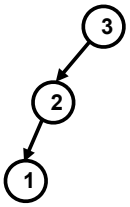
2 ; 1 ; 3



3 ; 1 ; 2



3 ; 2 ; 1



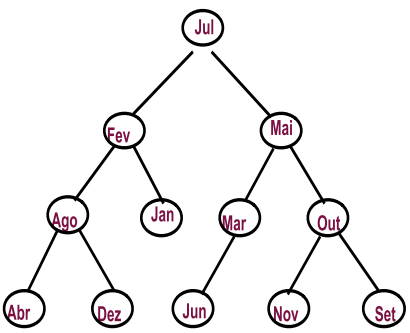
Profund.: log N

Profund.: N-1

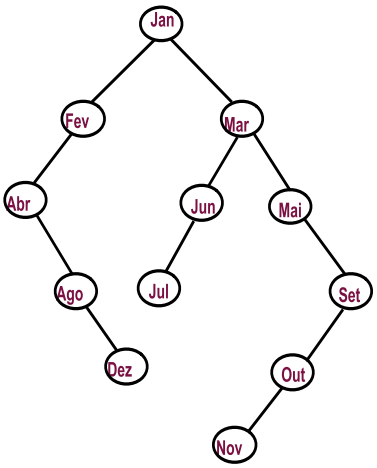
Profund.: N-1

Árvores Binárias de Pesquisa
conceitos

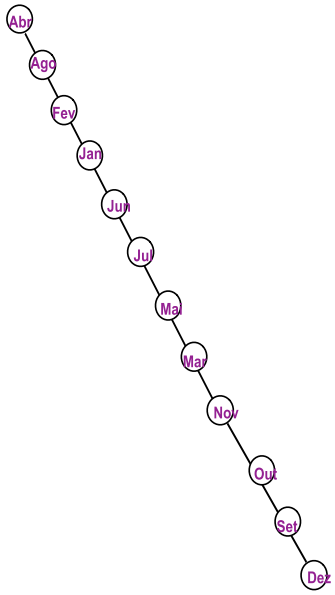
Exemplo de uma ÁRVORE BINÁRIA EQUILIBRADA para os meses do ano:



Exemplo de uma ÁRVORE BINÁRIA NÃO EQUILIBRADA para os meses do ano:



Exemplo de uma ÁRVORE BINÁRIA DEGENERADA (os meses foram inseridos por ordem alfabética):



Árvores Binárias de Pesquisa
conceitos

OBSERVAÇÃO: **as árvores binárias equilibradas têm como propriedade estrutural o garantirem profundidade logaritmica no pior caso.**

DIFICULDADE: **Um método de inserção de elementos numa árvore em que, após cada inserção, a árvore fique perfeitamente equilibrada é muito complexo.**

SOLUÇÃO: **uma possível solução são as árvores AVL.**

Árvores AVL
conceitos

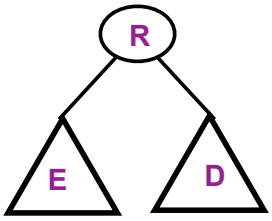
Definição de árvore equilibrada segundo Adelson-Velkii e Landis (árvore AVL):

Uma árvore binária está equilibrada se e só se

para cada elemento a altura das suas subárvores diferir no máximo de uma unidade.

Árvores AVL
conceitos

Caso da inserção de um elemento na subárvore esquerda



$h_e = h_d$: E e D ficam com alturas diferentes mas o critério AVL ainda é válido

$h_e < h_d$: E e D ficam com alturas iguais

$h_e > h_d$: O critério AVL deixa de ser válido; é preciso reestruturar a árvore

Factor de equilíbrio de um elemento t de uma árvore binária: $FE(t) = h_e - h_d$

Numa árvore AVL: $FE(t) = -1, 0, +1$

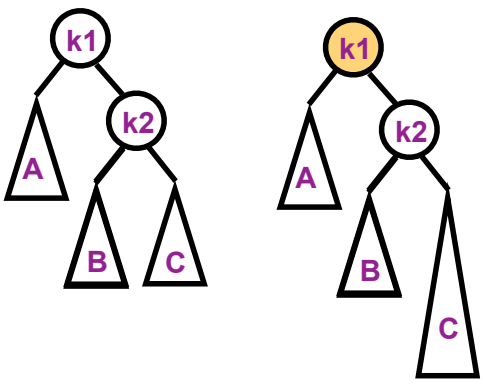
Para reequilibrar uma árvore são efectuadas rotações em torno do antecessor mais próximo do elemento inserido que tenha FE = ± 2

Árvores AVL
conceitos

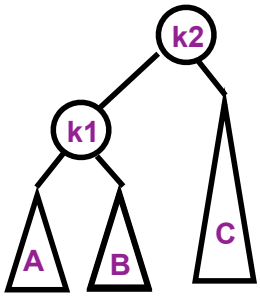
Nas árvores AVL são feitas 4 tipo de rotações que vão resolver 4 situações de inserção.

Situação 1:
DD - O novo elemento Y é inserido na subárvore Direita da subárvore Direita de k1.

Inseriu Y



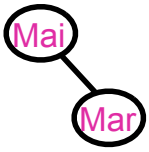
Faz rotação simples com filho k2 à direita



Árvores AVL
conceitos

Exemplo de inserção AVL

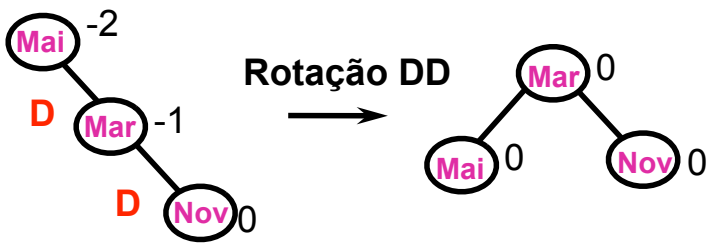
Meses a inserir na árvore:
Mai, Mar, Nov, Ago, Abr, Jan, Dez, Jun, Fev

Elemento a inserir	Depois da inserção	Depois de equilibrada
Mai		—
Mar		—

Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

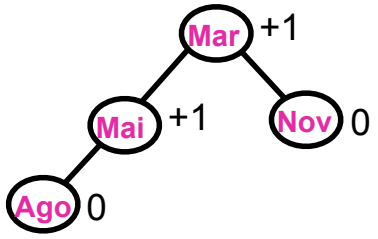
Meses a inserir na árvore:
Mai, Mar, Nov, Ago, Abr, Jan, Dez, Jun, Fev

Elemento a inserir	Depois da inserção	Depois de equilibrada
Nov		

Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

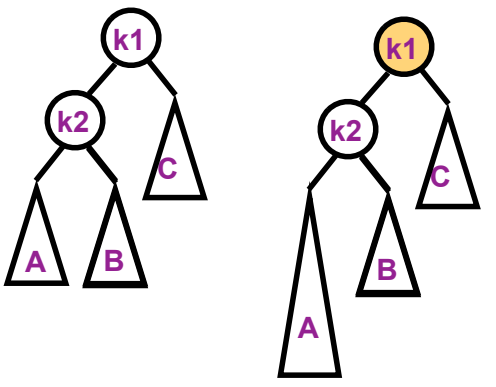
Meses a inserir na árvore:
~~Mai~~, ~~Mar~~, ~~Nov~~, Ago, Abr, Jan, Dez, Jun, Fev

Elemento a inserir	Depois da inserção	Depois de equilibrada
Ago		—

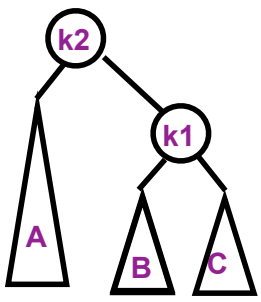
Árvores AVL
conceitos

Situação 2:
EE - O novo elemento Y é inserido na subárvore Esquerda da subárvore Esquerda de k1.

Inseriu Y



Faz rotação simples com filho k2 à esquerda



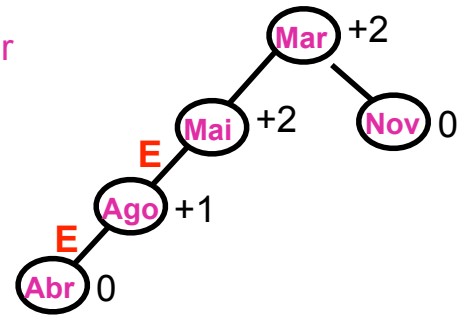
Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
~~Mai,~~ ~~Mar,~~ ~~Nov,~~ ~~Ago,~~ Abr, Jan, Dez, Jun, Fev

Elemento a inserir | Depois da inserção | Depois de equilibrada

Abr



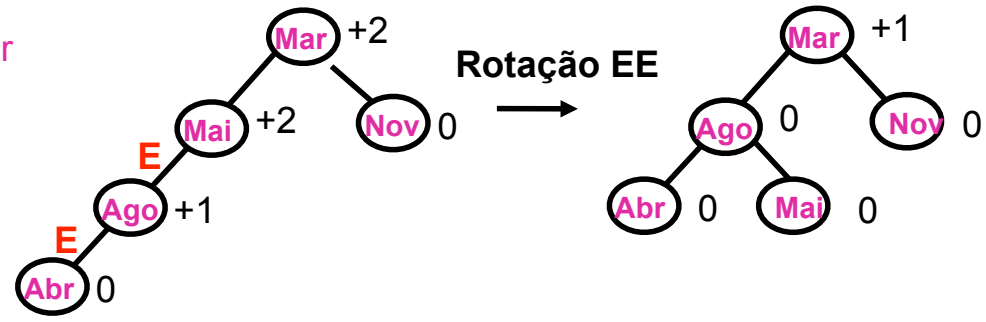
Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
~~Mai,~~ ~~Mar,~~ ~~Nov,~~ ~~Ago,~~ Abr, Jan, Dez, Jun, Fev

Elemento a inserir | Depois da inserção | Depois de equilibrada

Abr

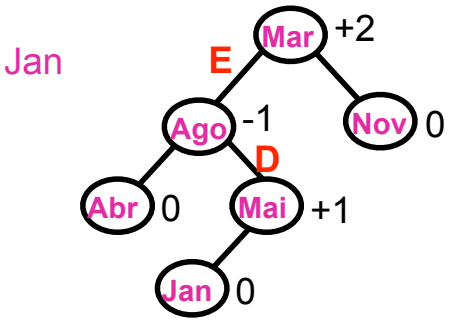


Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

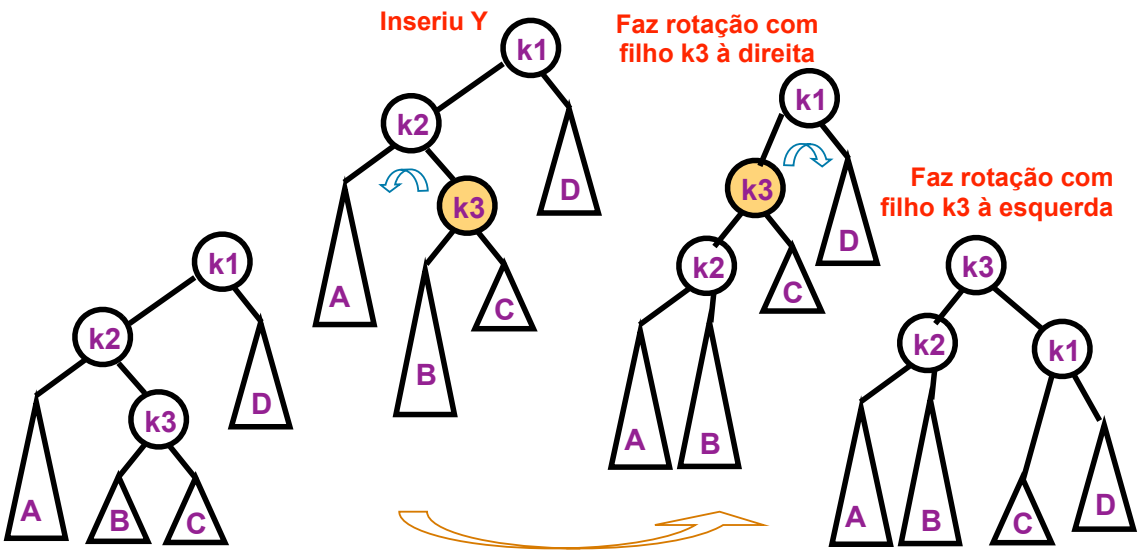
Meses a inserir na árvore:
~~Mai~~, ~~Mar~~, ~~Nov~~, ~~Ago~~, ~~Abr~~, Jan, Dez, Jun, Fev

Elemento a inserir | Depois da inserção | Depois de equilibrada



Árvores AVL
conceitos

Situação 3:
ED - O novo elemento Y é inserido na subárvore Direita da subárvore Esquerda de k1.



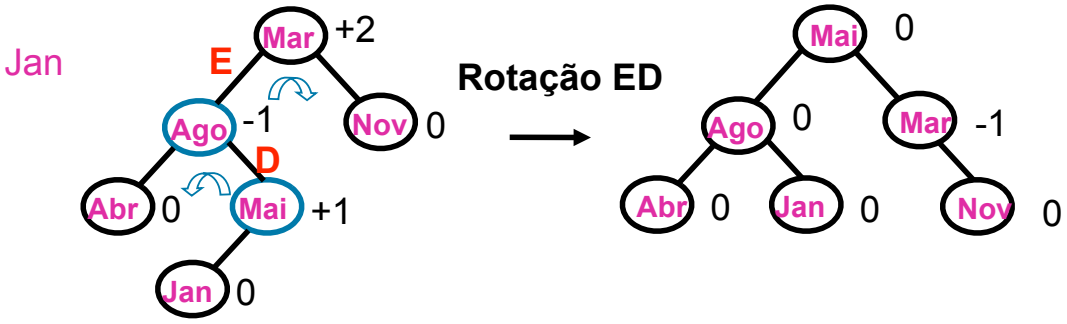
Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Dia 28 de Fevereiro
ficámos neste ponto

Meses a inserir na árvore:
~~Mai, Mar, Nov, Ago, Abr~~ Jan, Dez, Jun, Fe

Elemento a inserir Depois da inserção Depois de equilibrada

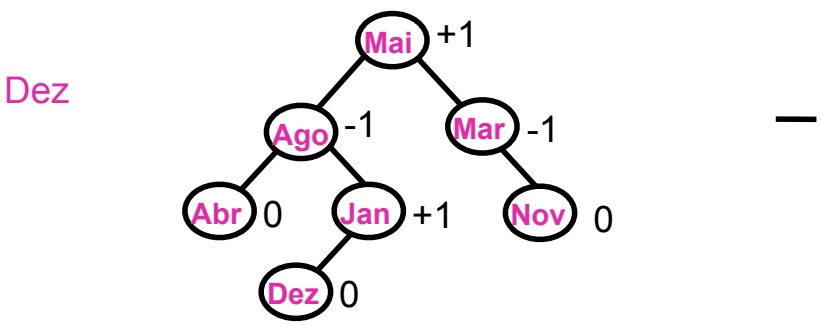


Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
~~Mai, Mar, Nov, Ago, Abr, Jan,~~ Dez, Jun, Fev

Elemento a inserir Depois da inserção Depois de equilibrada



—

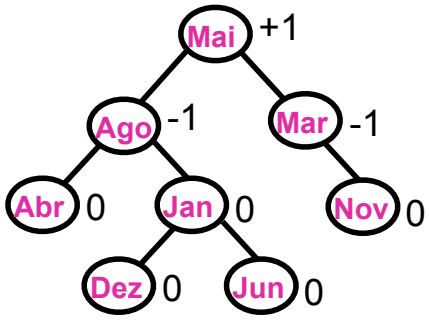
Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
Mai, Mar, Nov, Ago, Abr, Jan, Dez, Jun, Fev

Elemento a inserir | Depois da inserção | Depois de equilibrada

Jun



—

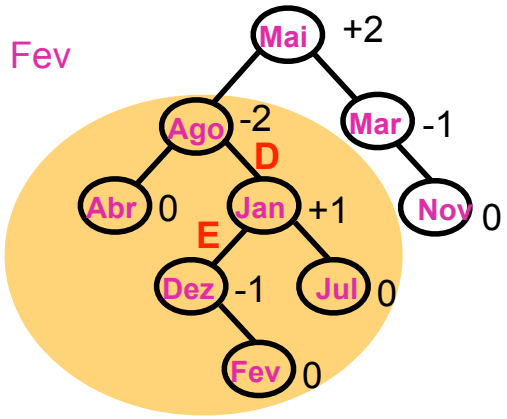
Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
Mai, Mar, Nov, Ago, Abr, Jan, Dez, Jun, Fev

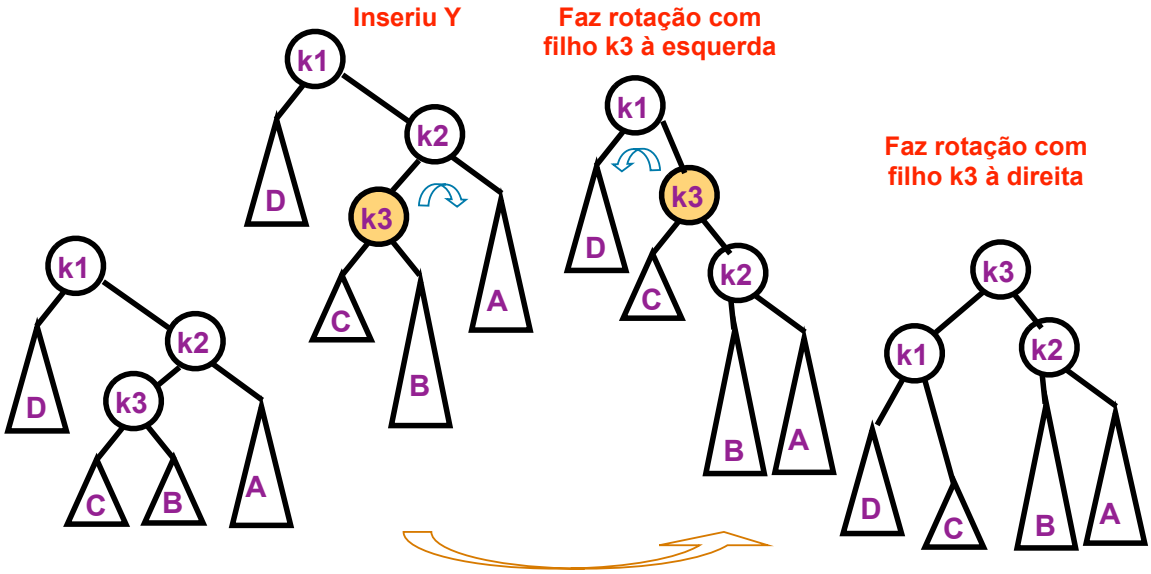
Elemento a inserir | Depois da inserção | Depois de equilibrada

Fev



Árvores AVL
conceitos

Situação 3:
DE - O novo elemento Y é inserido na subárvore Esquerda da subárvore Direita de k1.

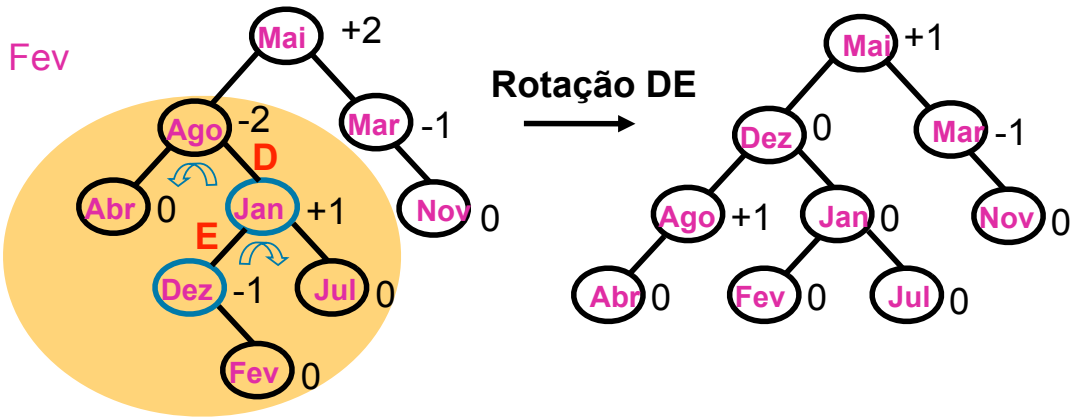


Árvores AVL
conceitos

Exemplo de inserção AVL (cont.)

Meses a inserir na árvore:
~~Mai,~~ ~~Mar,~~ ~~Nov,~~ ~~Ago,~~ ~~Abr,~~ ~~Jan,~~ ~~Dez,~~ ~~Jun,~~ Fev

Elemento a inserir Depois da inserção Depois de equilibrada



Árvores AVL
conceitos

Inserção AVL

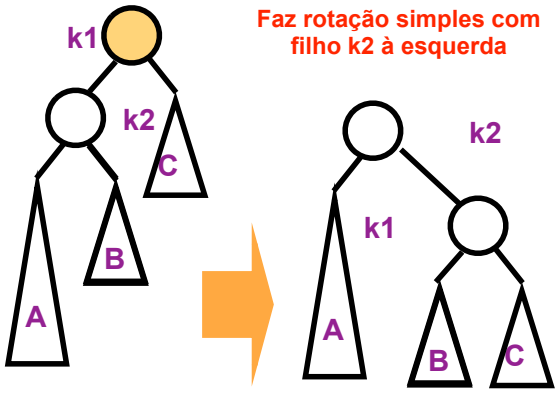
Cada nó da árvore tem um campo para guardar o seu FE

Método de inserção

- Inserir o novo elemento;
- actualizar o factor de equilíbrio dos nós afectados;
- determinar se é necessário reequilibrar a árvore;
- determinar tipo de rotação necessária;
- reequilibrar;

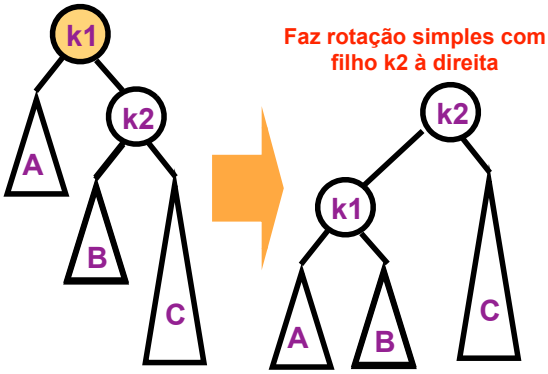
Árvores AVL
implementação

```
package DataStructures;

final class Rotations
{
   static BinaryNode withLeftChild( BinaryNode k1 )
   {
      BinaryNode k2 = k1.left;
      k1.left = k2.right;
      k2.right = k1;
      return k2;
   }
```



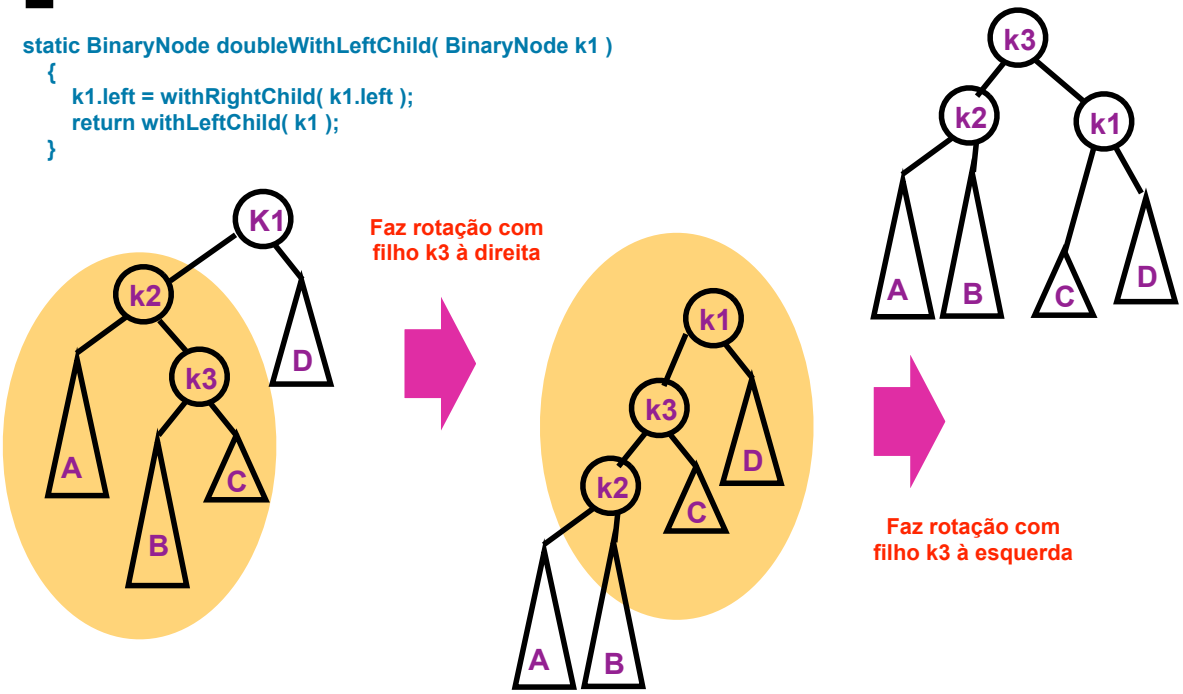
Árvores AVL
implementação

```
static BinaryNode withRightChild( BinaryNode k1 )
  {
    BinaryNode k2 = k1.right;
    k1.right = k2.left;
    k2.left = k1;
    return k2;
  }
```



Árvores AVL
implementação

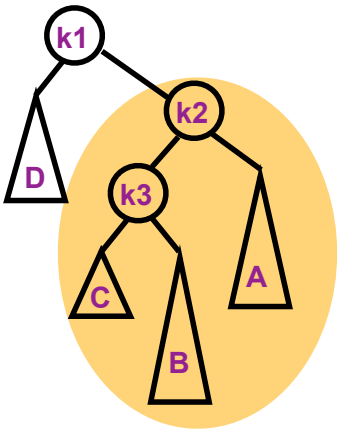
```
static BinaryNode doubleWithLeftChild( BinaryNode k1 )
  {
    k1.left = withRightChild( k1.left );
    return withLeftChild( k1 );
  }
```

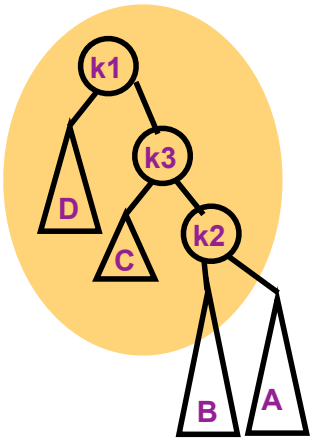


Árvores AVL
implementação

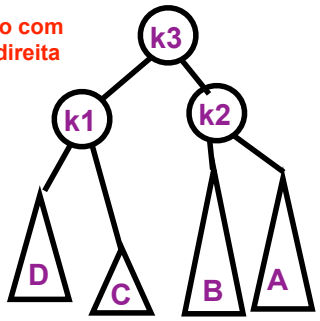
```
static BinaryNode doubleWithRightChild( BinaryNode k1 )
  {
    k1.right = withLeftChild( k1.right );
    return withRightChild( k1 );
  }
}
```

Faz rotação com filho k3 à esquerda





Faz rotação com filho k3 à direita



Árvores AVL
demonstações na Web

ex. de sequência de teste: 30 50 80 20 15 40 18

http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html

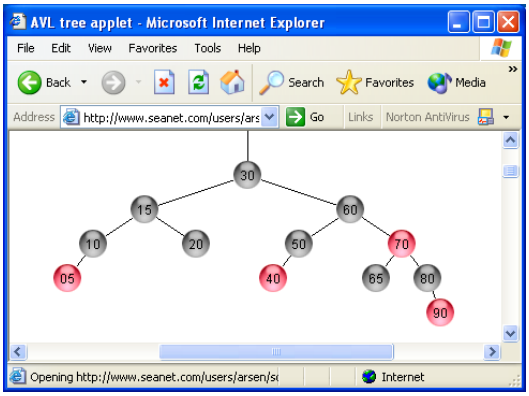
http://www.compapp.dcu.ie/~aileen/balance/ (indisponível em 21Mar2007)

Árvores Vermelhas e Pretas
conceitos

Data Structures & Problem Solving Using JAVA
Mark Weiss

Uma árvore VP é uma árvore binária de pesquisa em que:

1. Cada nó é colorido de preto ou vermelho.
2. A raiz é colorida de preto.
3. Se um nó é vermelho os seus filhos são coloridos de preto.
4. Todos os caminhos de cada nó até às suas folhas têm o mesmo número de nós pretos.

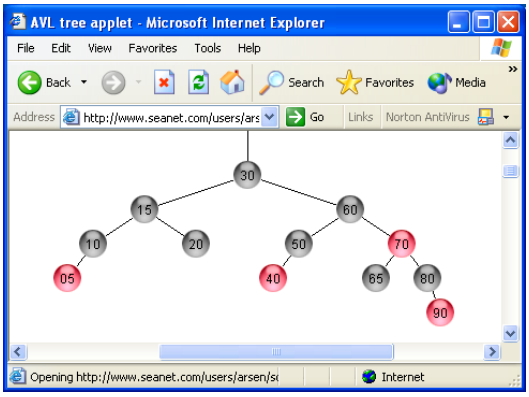


Árvores Vermelhas e Pretas
conceitos

Data Structures & Problem Solving Using JAVA
Mark Weiss

Complexidade::

- Se cada caminho da raiz até ás folhas contém B nós pretos então existirão pelo menos 2^B-1 nós pretos na árvore.
- h<=2B
- n>=2^B-1
- h<=2log(n+1)
- Como a raiz tem um nó preto e não podem haver nós vermelhos consecutivos, então a altura máxima de uma árvore VP é no máximo 2 log(n+1).
- Está garantido que a procura numa árvore VP tem complexidade logaritmica !!

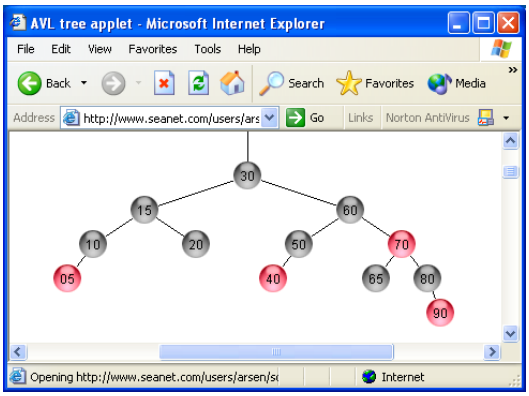


Árvores Vermelhas e Pretas
conceitos

Data Structures & Problem Solving Using JAVA
Mark Weiss

Inserção::

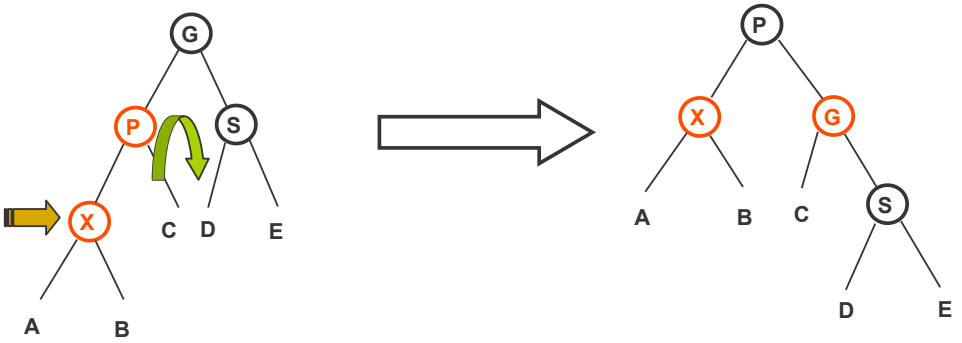
- Os nós são inseridos nas folhas da árvore ➔ um novo nó tem de ter a cor vermelha caso contrário viola 4.
- Se o nó antecessor for vermelho, a inserção do nó leva à violação de 3.
- LOGO: vamos ter de fazer **rotações** e **trocas de côr** nos nós para resolver o conflito 3.



Árvores Vermelhas e Pretas
conceitos

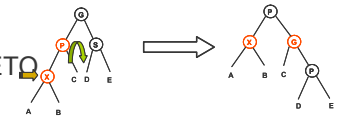
Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO PRETO

*inserção na sub **árvore esquerda** de uma sub **árvore esquerda***



Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO PRETO



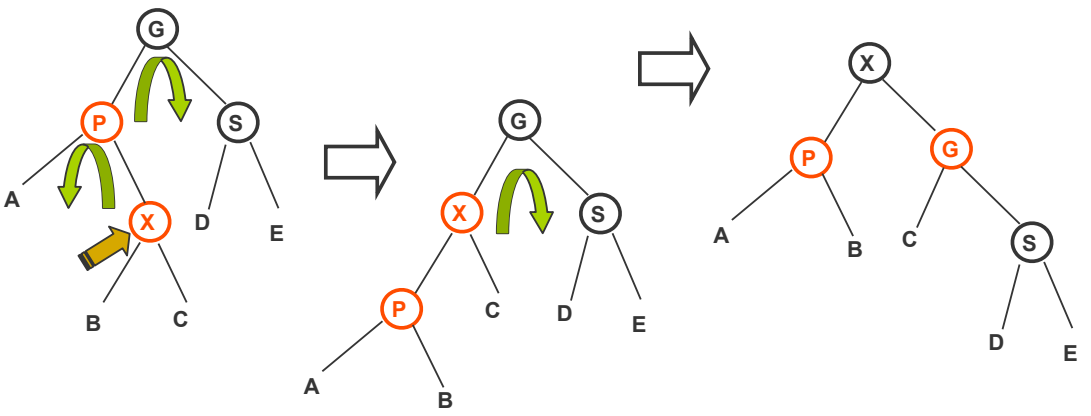
*inserção na sub **árvore direita** de uma sub **árvore direita***

situação simétrica da anterior !!

Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO PRETO

*inserção na sub **árvore direita** de uma sub **árvore esquerda***



Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO PRETO

*inserção na sub **árvore esquerda** de uma sub **árvore direita***

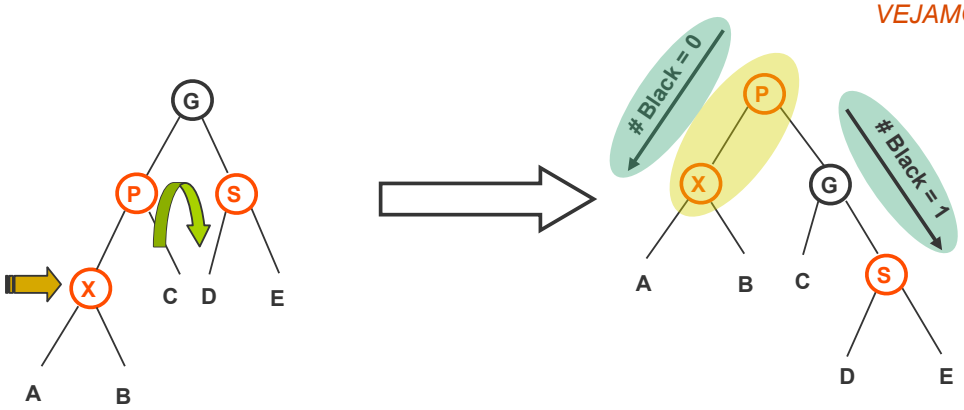
situação simétrica da anterior !!

Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO VERMELHO

nesta situação nem a rotação simples nem a dupla resolvem o problema –
em ambas as soluções ficamos com dois nós vermelhos consecutivos !!

VEJAMOS…

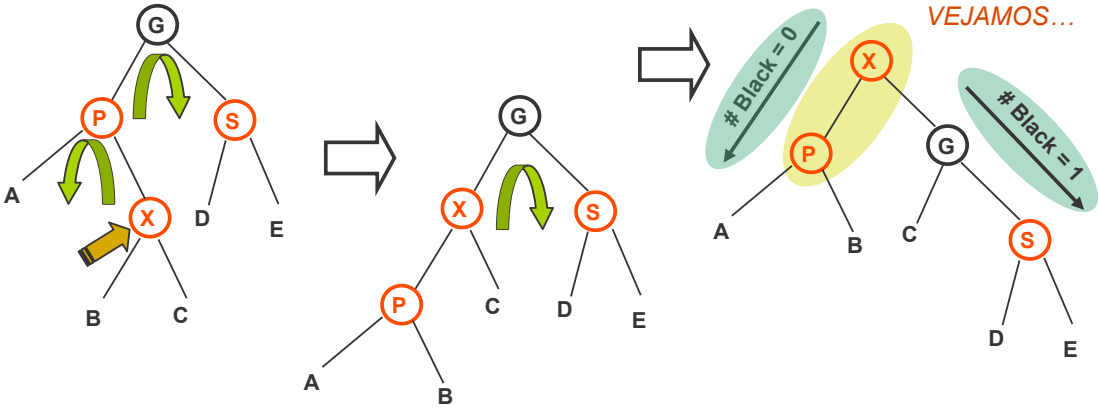


Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO VERMELHO

nesta situação nem a rotação simples nem a dupla resolvem o problema – em ambas as soluções ficamos com dois nós vermelhos consecutivos !!

VEJAMOS…

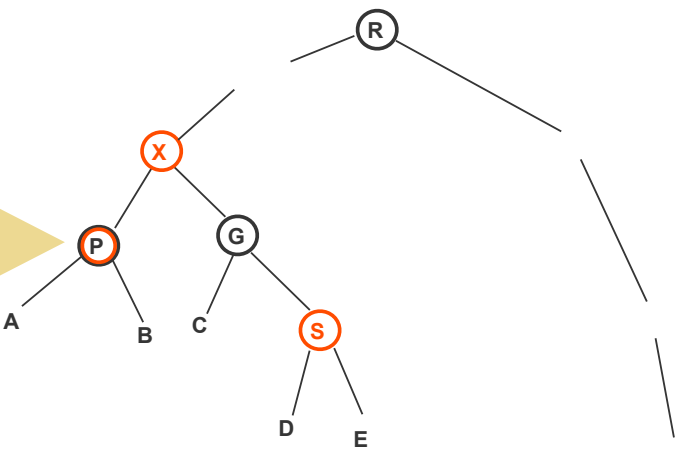


Árvores Vermelhas e Pretas
conceitos

Inserção:: O ASCENDENTE É VERMELHO E O SEU IRMÃO VERMELHO

BOM PODIAMOS FAZER ALGUMA REPINTURA PARA NÃO TER DOIS NÓS VERMELHOS CONSECUTIVOS !!

altera o balanceamento de # de nós pretos em todos os ramos a montante deste nó !!



Árvores Vermelhas e Pretas
conceitos

SOLUÇÃO :: Abordagem TOP-DOWN

Árvores Vermelhas e Pretas
conceitos

SOLUÇÃO :: Abordagem TOP-DOWN

Na travessia da árvore para inserção de um nó garantir que nunca temos os 2 antecedentes directos do nó a inserir vermelhos !

Garantida esta condição:: podemos inserir o novo nó como nova folha da árvore e realizar as operações de rotação como especificadas anteriormente.

SOLUÇÃO:: quando no percurso descendente da árvore encontramos um nó X com os seus descendentes ambos vermelhos pintamos o nó X a vermelho e os seus descendentes directos a preto.

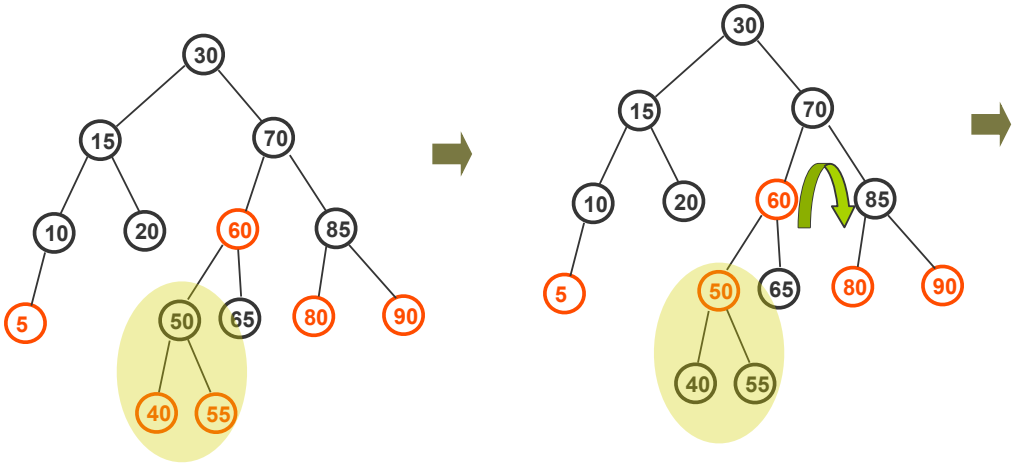
PROBLEMA:: o antecedente de X pode já ser vermelho!

SOLUÇÃO:: aplicamos sobre X e o seu ascendente, ambos a vermelho, as rotações estudadas anteriormente!

Árvores Vermelhas e Pretas
exemplo

SOLUÇÃO :: Abordagem TOP-DOWN :: Exemplo

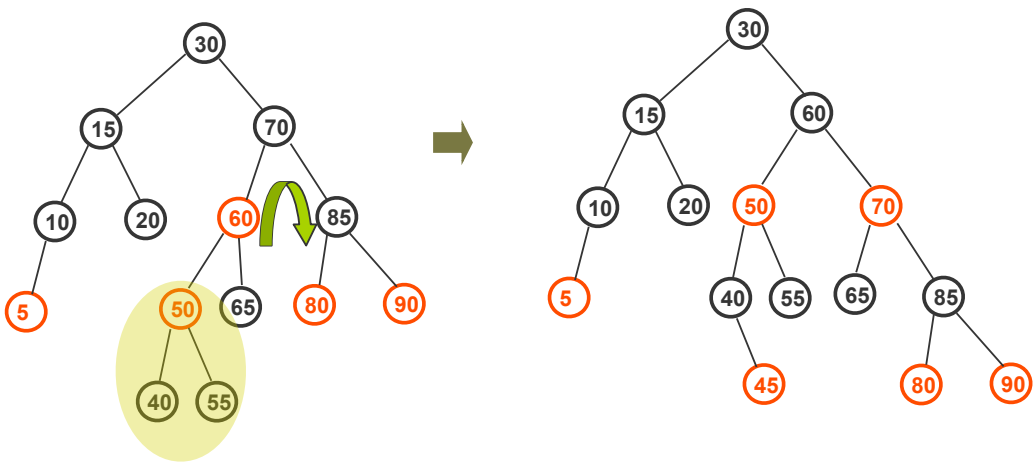
... quero inserir **45**



Árvores Vermelhas e Pretas
exemplo

SOLUÇÃO :: Abordagem TOP-DOWN :: Exemplo

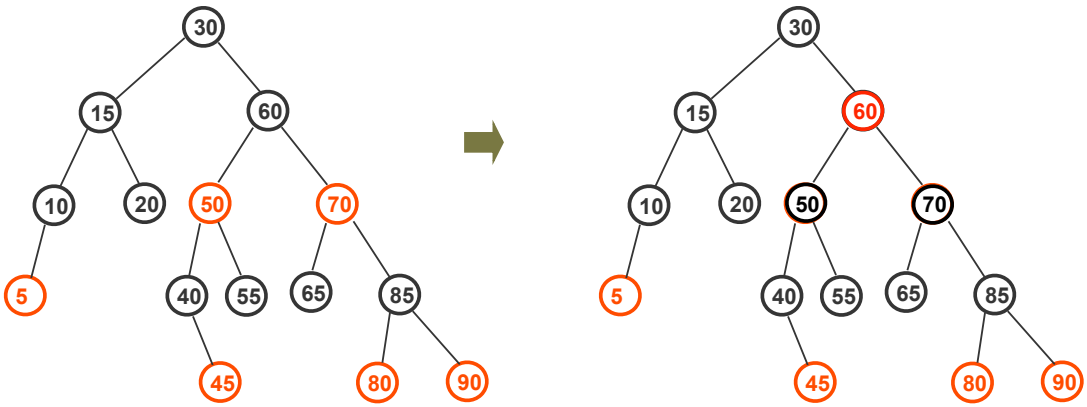
... quero inserir **45**



Árvores Vermelhas e Pretas
exemplo

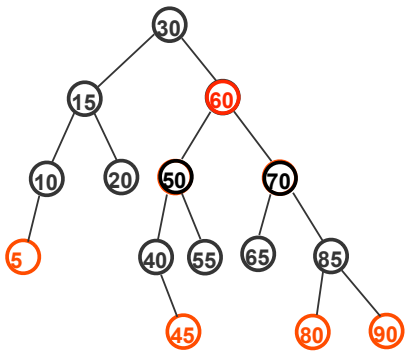
SOLUÇÃO :: Abordagem TOP-DOWN :: Exemplo

..... 45



Árvores Vermelhas e Pretas
exemplo

SOLUÇÃO :: Abordagem TOP-DOWN :: Exemplo



Observações:

- a árvore resultante é bastante equilibrada.
- o número de nós atravessados em média durante uma pesquisa é muito semelhante ao que temos na travessia de uma árvore AVL.
- … isto embora o método de equilíbrio de uma árvore VP seja mais frágil do que o que temos numa AVL.
- as vantagens das árvores VP estão num processo de inserção relativamente simples em que na prática poucas rotações são feitas
- uma desvantagem está em que o processo de eliminação de nós numa árvore VP é pouco eficiente.

Árvores Vermelhas e Pretas
conceitos

Data Structures & Problem Solving Using JAVA
Mark Weiss

Eliminação::

- Top-down
- A remoção é feita sobre nós que são folhas ou só têm um descendente.
- A remoção de nós com dois descendentes passa por movimentações de nós.
- Se o nó a remover for vermelho – não há problema!
- Se o nó a remover for preto (há a violação de 4.)
- SOLUÇÃO:

  assegurar que um nó a remover foi antecipadamente recolorido de vermelho.

Árvores Vermelhas e Pretas
demonstração na Web

ex. de sequência de teste: 30 50 80 20 15 40 18

http://www.ece.uc.edu/~franco/C321/html/RedBlack/redblack.html

Árvores Vermelhas e Pretas

... end ;-)