Programação 2 _ T07

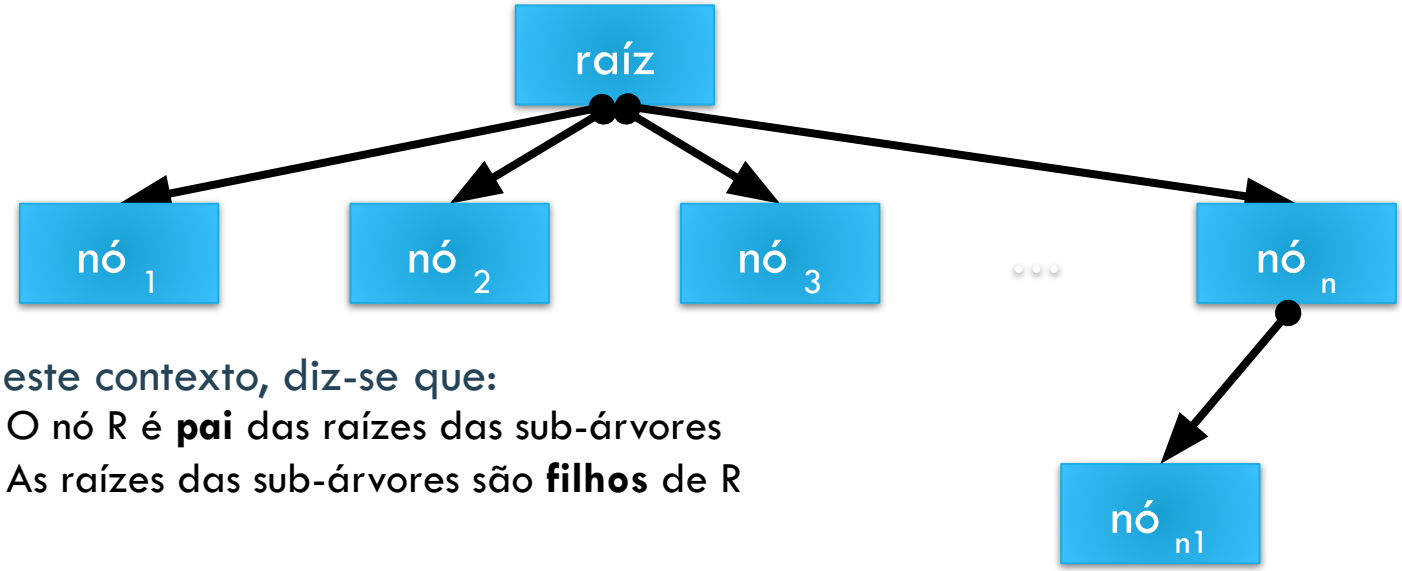
Árvores - Árvores binárias, árvores binárias de pesquisa, árvores AVL e árvores 'red-black'.

Rui Camacho
(slides por Luís Teixeira)
MIEEC 2020/2021

ÁRVORES

Definição recursiva:

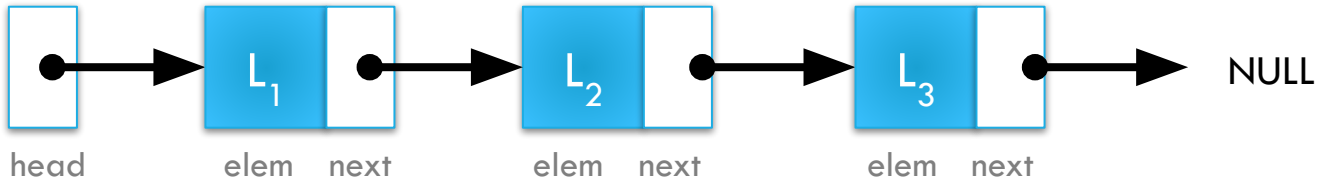
- Conjunto de nós que pode ser...
  - Conjunto **vazio**
  - Um nó R chamado **raiz** e 0 ou mais **(sub)árvores** cujas raízes estão ligadas a R por uma aresta direcionada.

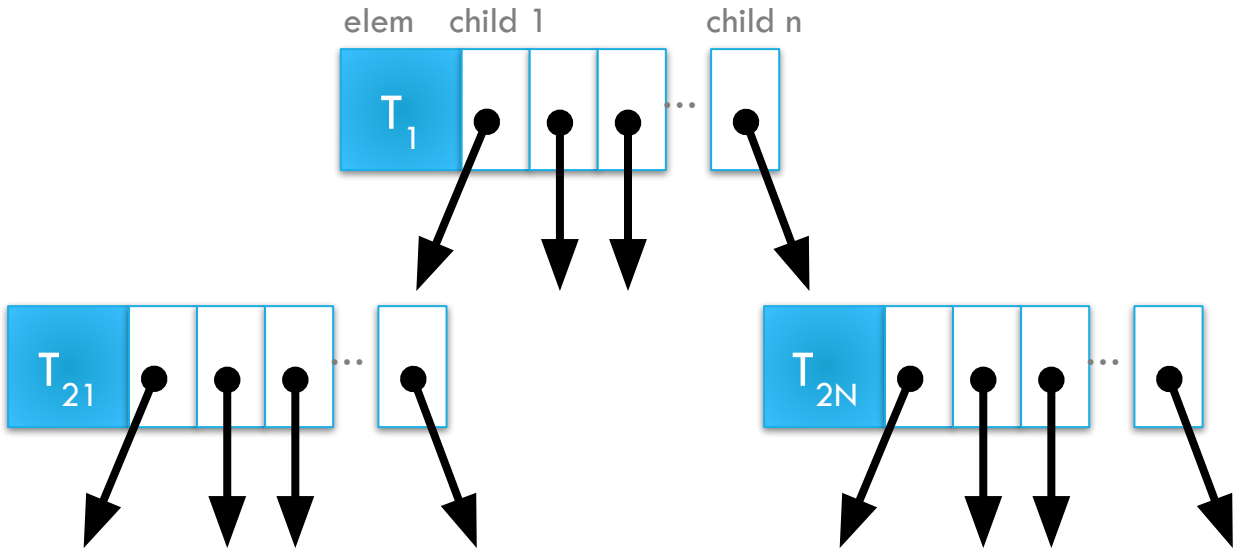


Neste contexto, diz-se que:

- O nó R é **pai** das raízes das sub-árvores
- As raízes das sub-árvores são **filhos** de R

LISTAS VS. ÁRVORES





ÁRVORES

À exceção da raiz, todos os nós de uma árvore têm 1 (e apenas 1) pai.

- A raiz não tem pai.

Há um caminho único da raiz a cada nó.

- O **tamanho do caminho** é o número de arestas a percorrer

Folha: nó sem filhos.

Profundidade de um nó:

- Comprimento do caminho da raiz até ao nó
  - Profundidade da raiz é 0
  - Profundidade de um nó é 1 + profundidade do pai

ÁRVORES

Altura de um nó

- Comprimento do caminho do nó até à folha a maior profundidade
  - Altura de uma folha é 0
  - Altura de um nó é 1 + a altura do seu filho de maior altura
- Altura da árvore: altura da raiz

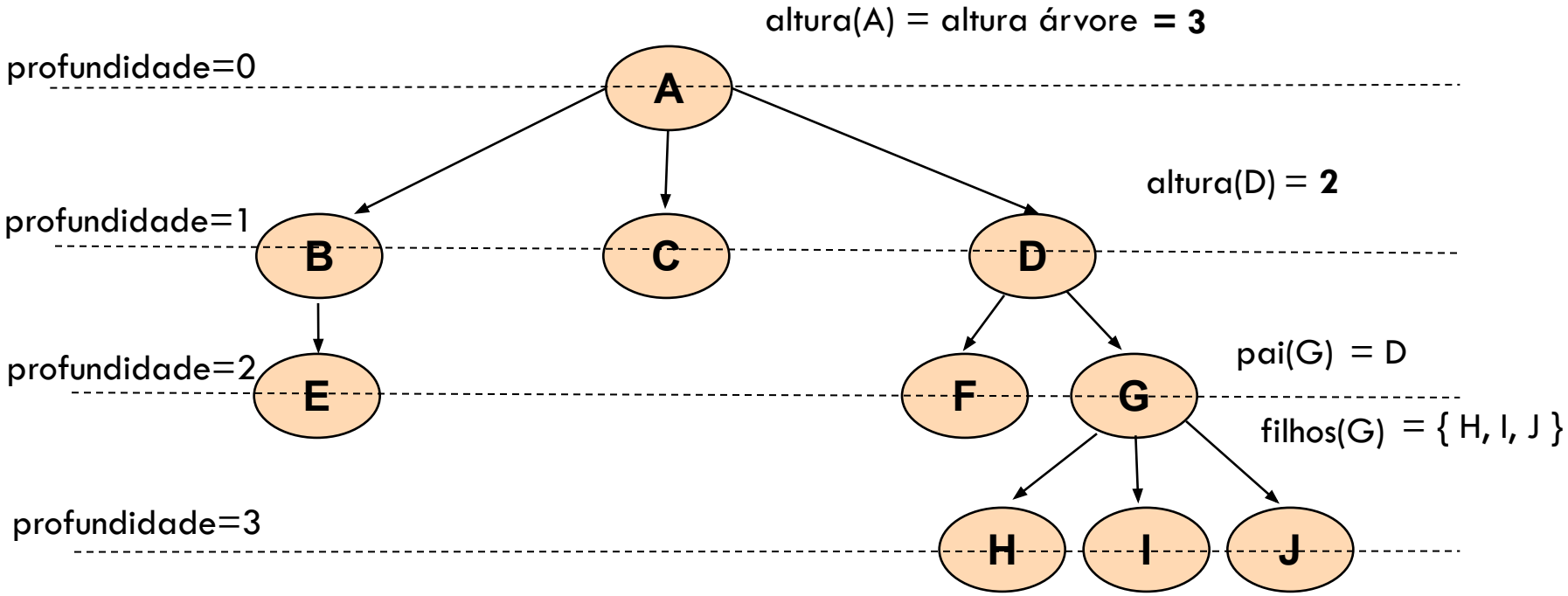
Se existe caminho do nó u para o nó v, então:

- u é antepassado de v
- v é descendente de u

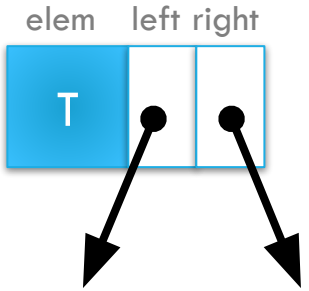
Tamanho de um nó: número de descendentes

ÁRVORES

Exemplo:



ÁRVORES BINÁRIAS



Definição:

- Árvore em que cada nó tem no máximo 2 filhos.

Propriedades:

- Uma árvore binária não vazia com **altura *h*** tem no **mínimo *h+1***, e no **máximo $2^{h+1}-1$** nós
- A **altura** de uma árvore com ***n* elementos** (n>0) é no mínimo $\log_2 n$, e no máximo ***n-1***
- A **profundidade média** de uma árvore de ***n* nós** é **O($\sqrt{n}$)**

ÁRVORES BINÁRIAS

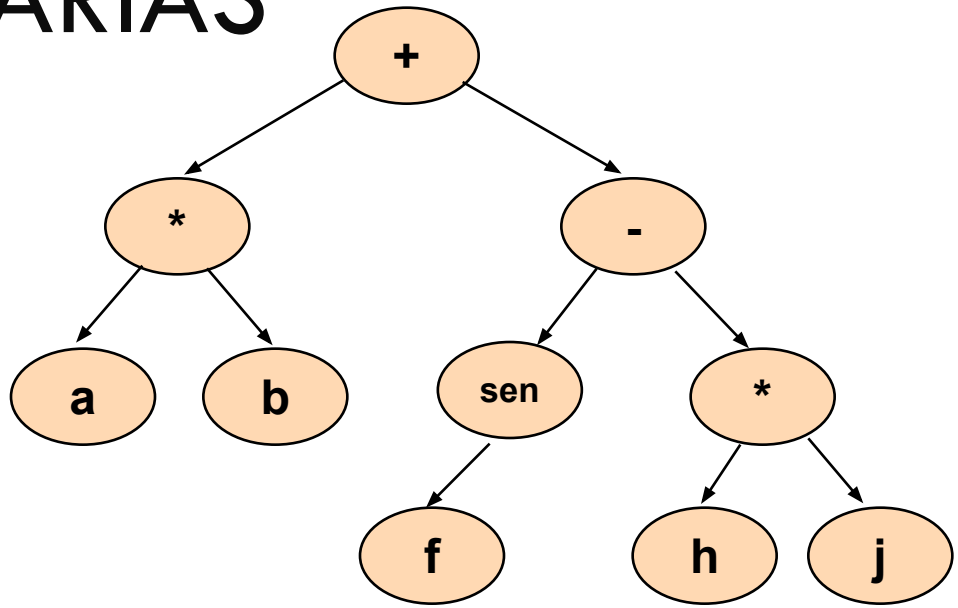
Percorrer árvores

Os elementos de uma árvore (binária) podem ser enumerados por quatro ordens diferentes. As três primeiras definem-se recursivamente:

- ***Pré-ordem*:** Primeiro a raiz, depois a sub-árvore esquerda, e finalmente a sub-árvore direita
- ***Em-ordem*:** Primeiro a sub-árvore esquerda, depois a raiz, e finalmente a sub-árvore direita
- ***Pós-ordem*:** Primeiro a sub-árvore esquerda, depois a sub-árvore direita, e finalmente a raiz
- ***Por nível*:** Os nós são processados por nível (profundidade) crescente, e dentro de cada nível, da esquerda para a direita

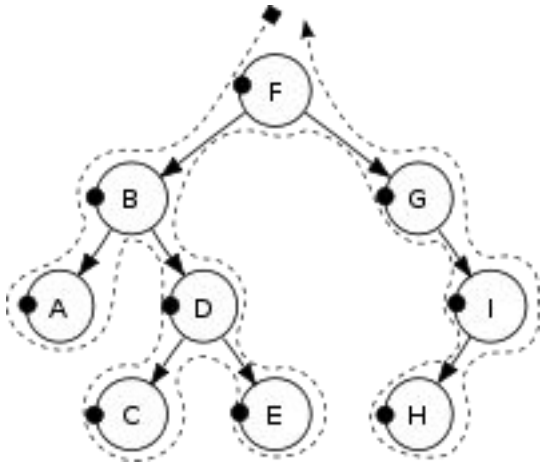
ÁRVORES BINÁRIAS

Percorrer árvores - exemplo

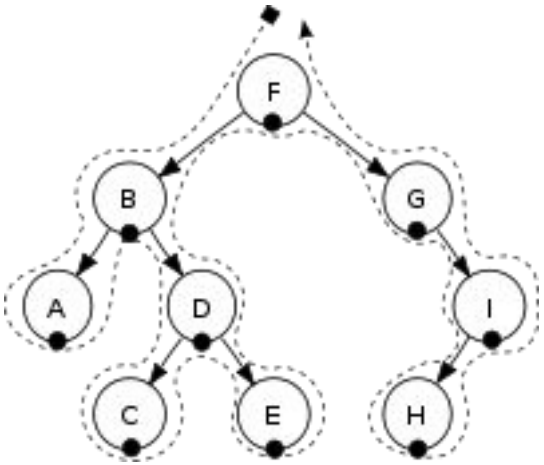


Pré-ordem	**+ * a b – sen f * h j**
Em-ordem	**a * b + f sen – h * j**
Pós-ordem	**a b * f sen h j * - +**
Por nível	**+ * - a b sen * f h j**

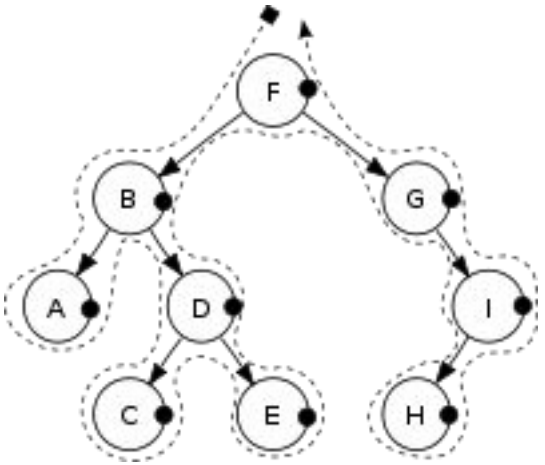
ÁRVORES BINÁRIAS



FBADCEGIH



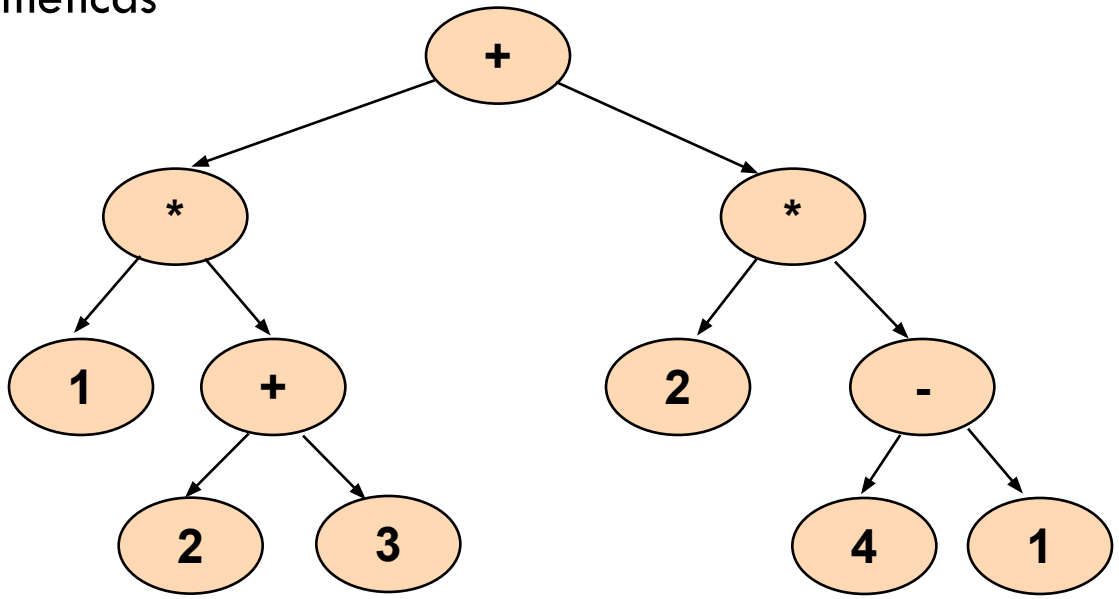
ABCDEFGIH



ACEDBHIGF

ÁRVORES BINÁRIAS - APLICAÇÕES

Expressões aritméticas



Expressão = **1 * (2 + 3) + (2 * (4 – 1))**

ÁRVORES BINÁRIAS - APLICAÇÕES

Construção da árvore de expressões

O algoritmo é similar ao algoritmo de *conversão infixa* para RPN (algoritmo **shunting-yard**) mas usa **duas pilhas**, uma para guardar os **operadores** e outra para guardar **sub-árvores** correspondentes a sub-expressões.

O algoritmo procede da seguinte forma:

- Números são transformados em árvores de 1 elemento e colocados na pilha de operandos.
- Operadores são tratados como no programa de conversão infixa, usando uma pilha apenas para operadores e parênteses.
- Quando um operador é retirado da pilha, duas sub-árvores (operandos) são retirados da pilha de operandos e combinados numa nova sub-árvore, que por sua vez é colocada na pilha.
- Quando a leitura da expressão chega ao fim, todos os operadores existentes na pilha são processados.

ÁRVORES BINÁRIAS EM C

Tal como a definição de árvore, costumam também ser recursivos:

- os algoritmos que as manipulam
- o tipo de dados que as suportam. Logo...
  - uma árvore é um nó (raiz)
  - qualquer nó de uma árvore é uma (sub)árvore

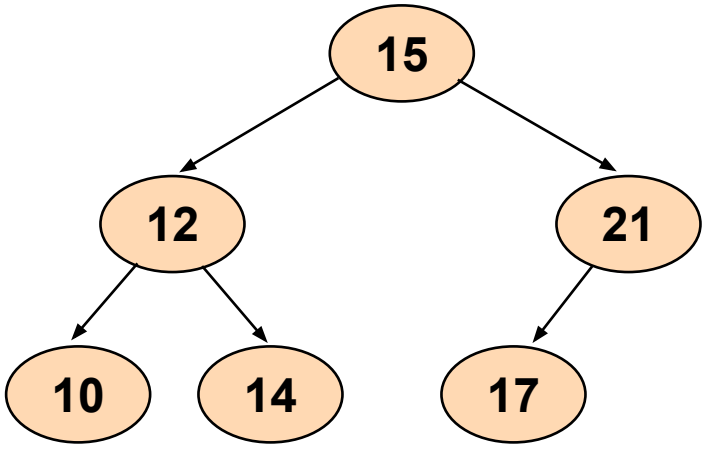
```
typedef struct node
{
     data_type     value;      /* data_type a definir */
     struct node *left_child;
     struct node *right_child;
      /*  struct node *parent;           raramente utilizado */
} tree_node;


/* se, adicionalmente, quisermos definir um tipo mais intuitivo... */
typedef tree_node*  tree;
```

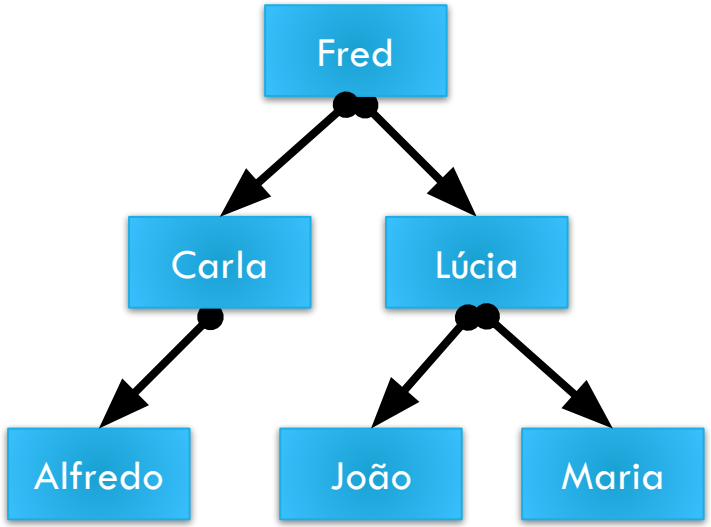
ÁRVORES BINÁRIAS DE PESQUISA

Definição:

Árvore binária, sem elementos repetidos, que verifica a seguinte propriedade:

- Para **cada nó**, todos os valores da sub-árvore esquerda são menores, e todos os valores da sub-árvore direita são maiores, que o valor desse nó





ÁRVORES BINÁRIAS DE PESQUISA

Estrutura linear:

- Pesquisa de um elemento pode ser feita em $O(\log n)$
- ...mas não a inserção ou remoção de um elemento!

Estrutura em árvore binária:

- Pode manter tempo de acesso logarítmico nas operações de inserção e remoção de um elemento.

ÁRVORES BINÁRIAS DE PESQUISA

Pesquisa

- Usa a propriedade de ordem na árvore para escolher caminho, eliminando uma sub-árvore a cada comparação

Inserção

- Como pesquisa; novo nó inserido onde pesquisa falha

Máximo e mínimo

- Procura, escolhendo sempre a subárvore direita (máximo), ou sempre a sub-árvore esquerda (mínimo)

Remoção

- Nó folha: apagar nó
- Nó com 1 filho: filho substitui pai
- Nó com 2 filhos: elemento é substituído pelo menor da sub-árvore direita (ou maior da esquerda); o nó deste tem no máximo 1 filho e é apagado

ÁRVORES BINÁRIAS DE PESQUISA - APLICAÇÃO

Contagem de ocorrência de palavras

- Pretende-se escrever um programa que leia um ficheiro de texto e apresente uma listagem ordenada das palavras nele existentes e o respetivo número de ocorrências.
- Algoritmo:
  - Guardar as palavras e contadores associados numa árvore binária de pesquisa.
  - Usar ordem alfabética para comparar os nós.

ÁRVORES AVL

Uma **árvore AVL** (Adelson-Velskii and Landis) é uma árvore binária de pesquisa balanceada.

Uma árvore binária é dita **balanceada** se:

- para qualquer nó da árvore, a diferença entre as alturas das suas duas sub-árvores (direita e esquerda) não é maior do que 1.

ÁRVORES AVL

Fator de Balanceamento

=

Altura da sub-árvore Direita

−

Altura da sub-árvore Esquerda

Assim, numa árvore AVL, o fator de balanceamento em cada nó pode ser:

- **-1** (*left heavy*)
- **0**
- **1** (*right heavy*)

ÁRVORES AVL

Tempos de execução O(*log n*) para o pior caso e o caso médio na:

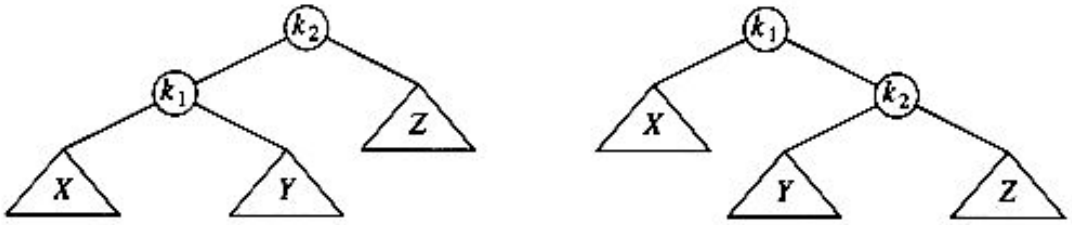
- Pesquisa
- Inserção
- Remoção

Inserções e remoções:

- podem implicar uma **rotação** (simples ou dupla) para manter a árvore balanceada.

ÁRVORES AVL - ROTAÇÕES

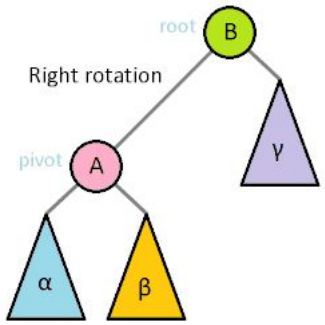
Consideremos as seguintes árvores binárias de pesquisa:

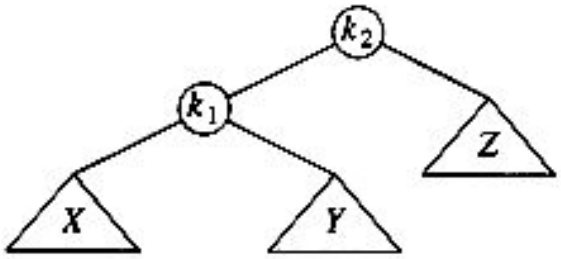


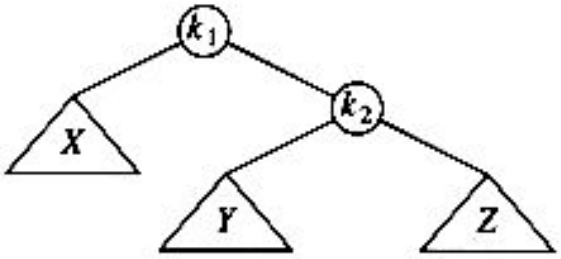
A conversão de uma na outra, em qualquer dos sentidos, é chamada de **rotação** (simples).

- Pode ser realizada em qualquer nó e não apenas na raiz, já que cada nó é raiz de uma (sub)árvore.

ÁRVORES AVL - ROTAÇÕES







Rotação simples à Direita

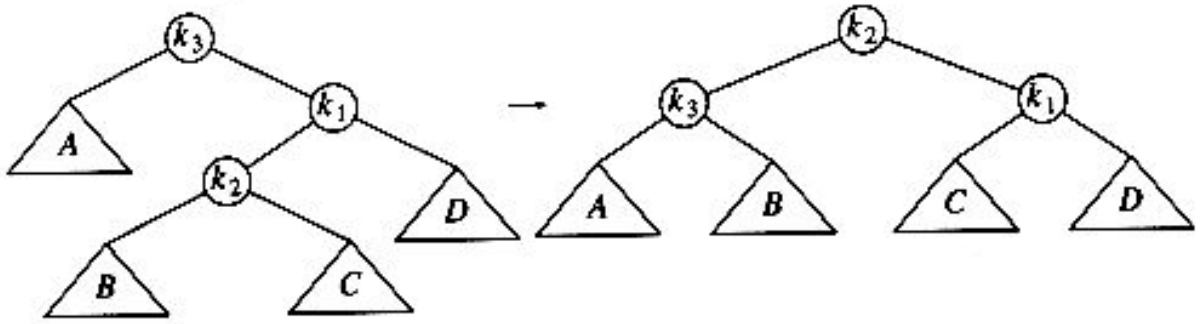
- Sendo k_1 o filho à esquerda de k_2
  - Tornar k_2 o filho à direita de k_1
  - Tornar o filho à direita de k_1 o filho à esquerda de k_2.

Rotação simples à Esquerda

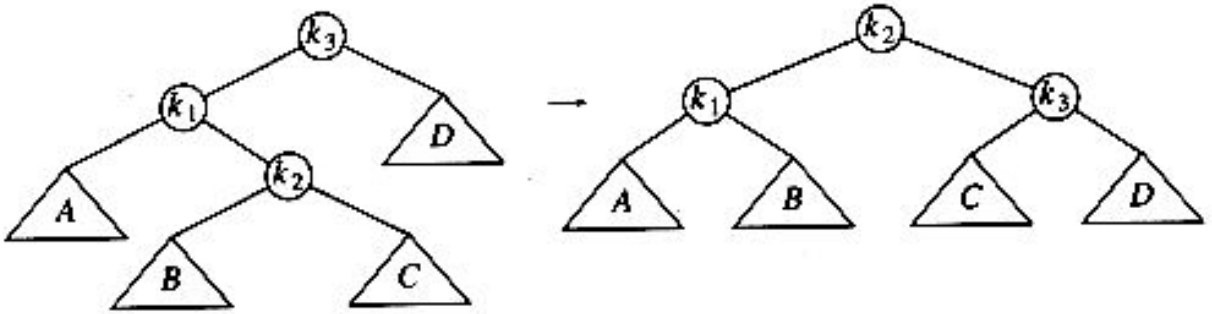
- Sendo k_2 o filho à direita de k_1
  - Tornar k_1 filho à esquerda de k_2
  - Tornar o filho à esquerda de k_2 o filho à direita de k_1.

ÁRVORES AVL - ROTAÇÕES

Rotação dupla à Esquerda (RL)



Rotação dupla à Direita (LR)



ÁRVORES AVL - ROTAÇÕES

Que rotação fazer após inserção/remoção?

- Algoritmo em pseudo-código:

```
IF tree is right heavy
{
    IF tree's right subtree is left heavy

        Perform Double Left (RL) rotation
    ELSE
        Perform Single Left rotation
}
ELSE IF tree is left heavy
{
    IF tree's left subtree is right heavy

        Perform Double Right (LR) rotation
    ELSE
        Perform Single Right rotation
}
```

ÁRVORES 'RED-BLACK'

Uma **árvore 'red-black'** é uma árvore binária de pesquisa auto-balanceada (balanceamento aproximado).

Balanceamento é preservado "pintando" cada nó com uma das duas cores ('red' ou 'black') de modo a satisfazer propriedades que restringem o desequilíbrio no pior caso

O balanceamento não é perfeito mas é o suficiente para garantir pesquisas em O(log n)

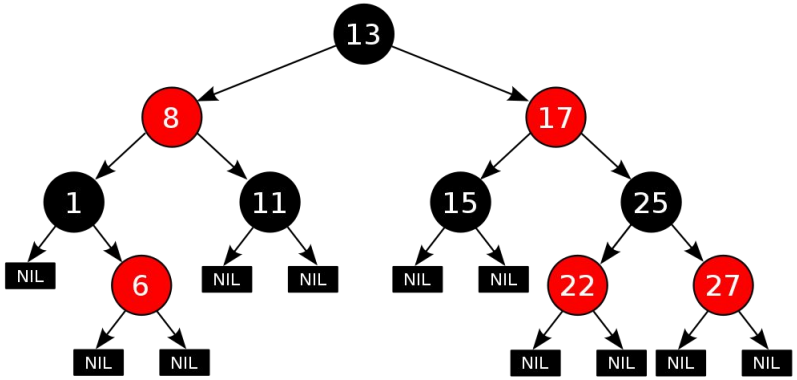
Inserção, remoção e operações de rearranjo são também realizadas em O(log n)

ÁRVORES ‘RED-BLACK’ - PROPRIEDADES

Nós são ‘red’ ou ‘ black’

O nó raiz é ‘black’

Nós-folha (NIL) são ‘black’



Todos o nós ‘red’ têm obrigatoriamente dois filhos ‘black’

Todos os caminhos de um nó para qualquer dos seus nós-folha contêm o mesmo número de nós ‘black’

AVISOS

Applet ilustrativo sobre operações com árvores AVL:

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

Vídeo ilustrativo sobre operações com árvores:

- https://www.youtube.com/watch?v=VbTnLV8pIU