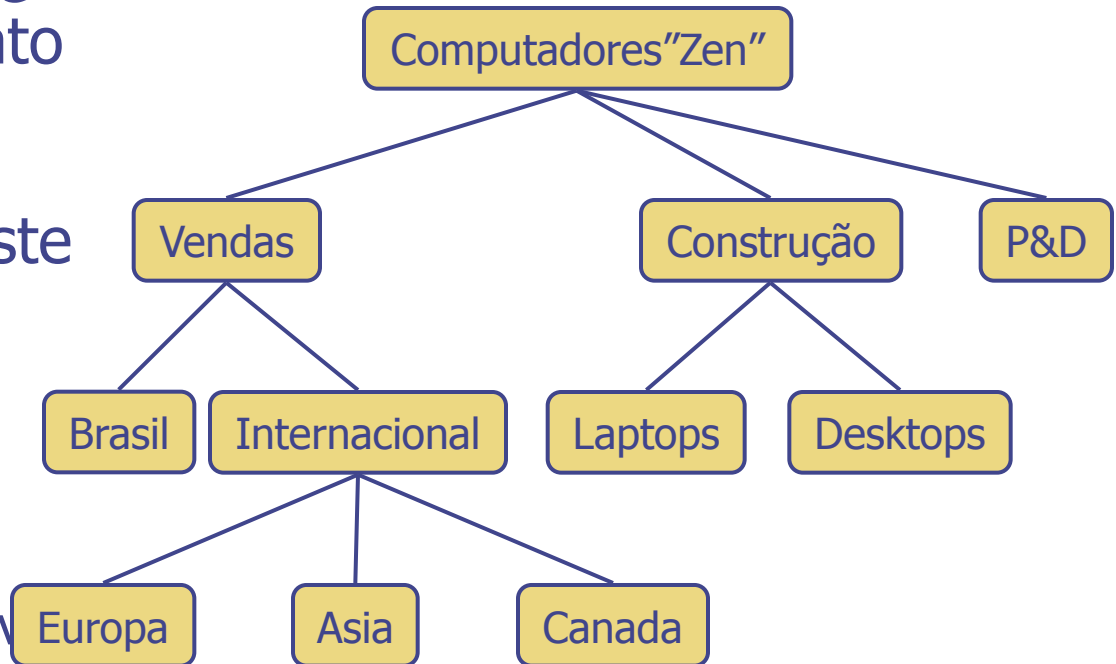


O que é uma árvore

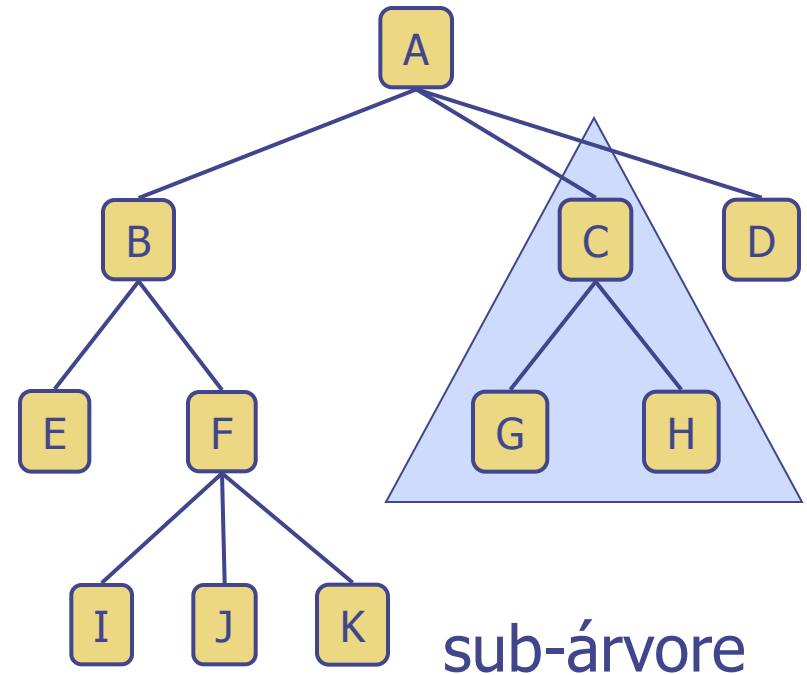
- ◆ Em Computação, é um modelo abstrato de uma estrutura hierárquica
- ◆ Uma árvore consiste de nós com uma relação pai-filho
- ◆ Aplicações:
 - Estrutura organizacional
 - Sistemas de arquivos
 - Ambiente de programação



Terminologia de árvore

- ◆ Raiz (root): Nó sem pai (A)
- ◆ Nó interno: Nó com, pelo menos, um filho (A, B, C, F)
- ◆ Nó externo: Nó sem filhos (E, I, J, K, G, H, D)
- ◆ Ancestral de um nó: pai, avô, bisavô, etc.
- ◆ Profundidade de um nó: Número de ancestrais
- ◆ Altura de um árvore: Profundidade máxima (3)
- ◆ Descendente de um nó: filho, neto, bisneto, etc.

- ◆ Sub-árvore: árvore formada por um nó e seus descendentes



TAD árvore

◆ Métodos genéricos:

- integer `size()`
- integer `height()`
- boolean `isEmpty()`
- Iterator `elements()`
- Iterator `nos()`

◆ Métodos de acesso:

- No `root()`
- No `parent(No)`
- Iterator `children(No)`

◆ Métodos de consulta:

- boolean `isInternal(No)`
- boolean `isExternal(No)`
- boolean `isRoot(No)`
- integer `depth(No)`

◆ Métodos de atualização:

- Object `replace(No, o)`

◆ Métodos adicionais podem ser definidos pela estrutura que estende/implementa o TAD árvore

Profundidade

- ◆ A profundidade de um nó v pode ser definida recursivamente como:
 - Se v for raiz, então a profundidade é 0
 - Senão, a profundidade é 1 mais a profundidade do pai de v

Algoritmo *depth*(v)

se (*isRoot*(v))

retorne 0

senão

retorne 1+*depth*(*parent*(v))

Altura

- ◆ A altura de um nó v pode ser definida recursivamente como:
 - Se v for externo, então a altura é 0
 - Senão, a altura é 1 mais a maior altura de um filho de v

Algoritmo $altura(v) - O(n)$

se ($isExternal(v)$)

retorne 0

senão

$h=0$

para cada w em $children(v)$

$h=\max(h, altura(w))$

retorne $1+h$

Algoritmo $altura2(v) O(n^2)$

$h=0$

para cada w em $nos()$

se ($isExternal(w)$)

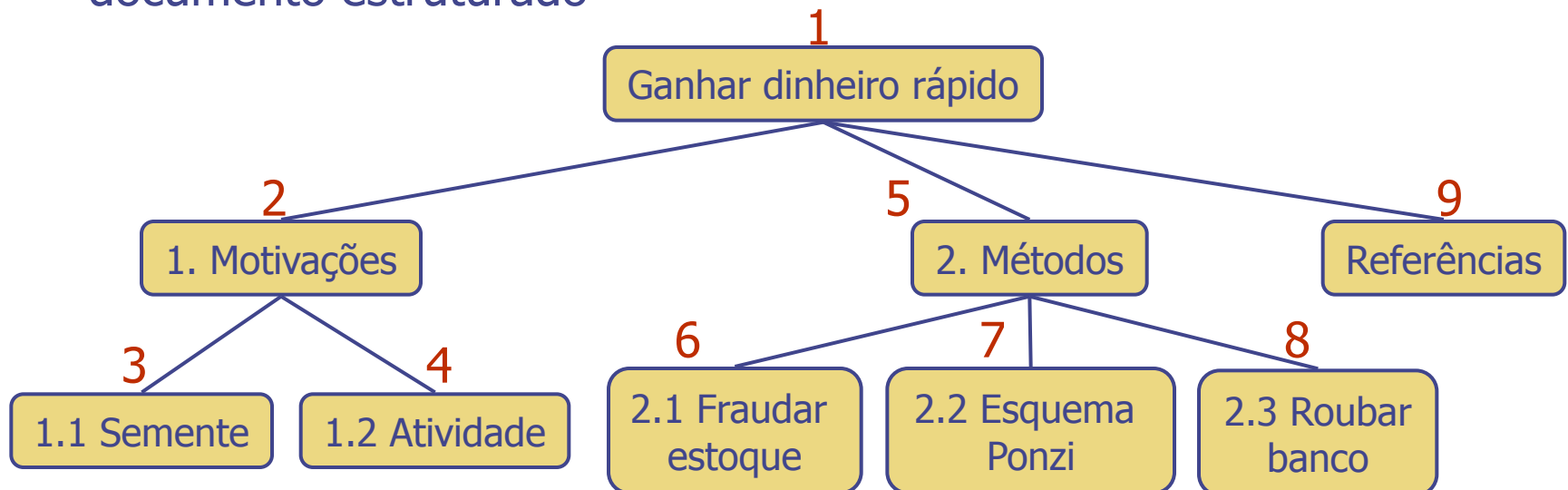
$h=\max(h, depth(w))$

retorne h

Travessia pré ordem

- ◆ Uma travessia visita os nós de uma árvore de uma forma sistemática
- ◆ Em uma travessia pré-ordem, um nó é visitado antes de seus descendentes
- ◆ Aplicação: imprimir um documento estruturado

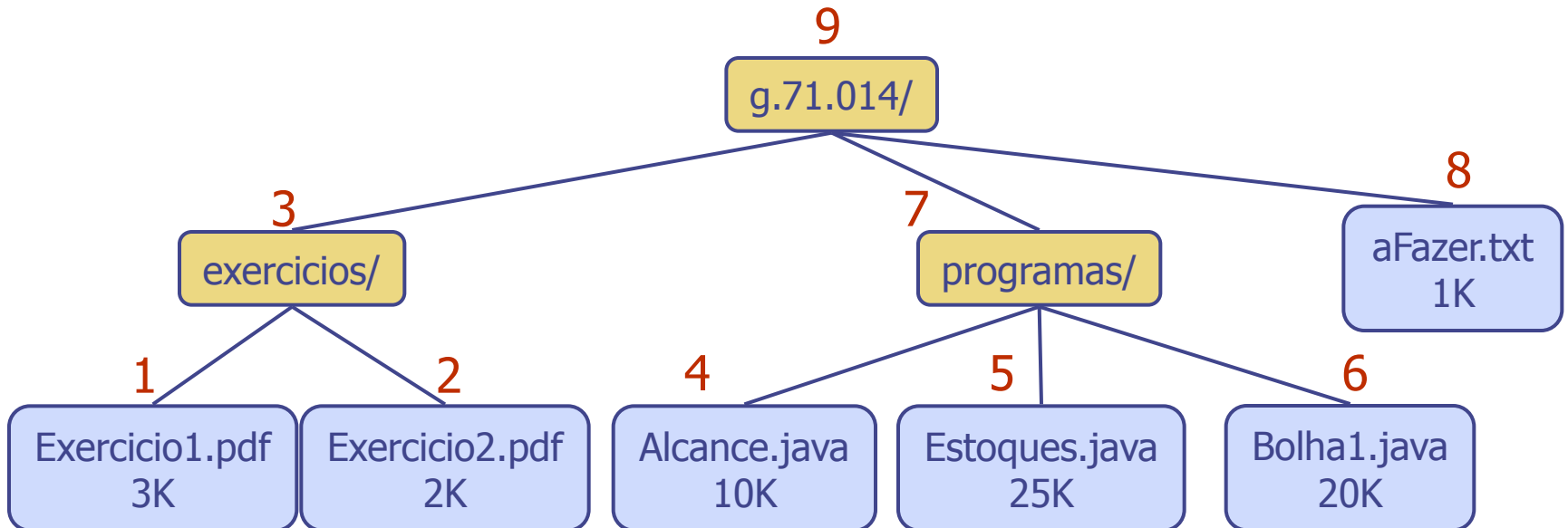
Algoritmo *preOrder(v)*
visite(v)
para cada filho *w* de *v*
preorder(w)



Travessia pós ordem

- ◆ Em uma travessia pós-ordem, um nó é visitado depois de seus descendentes
- ◆ Aplicação: Computar o espaço usado por diretórios, subdiretórios e arquivos

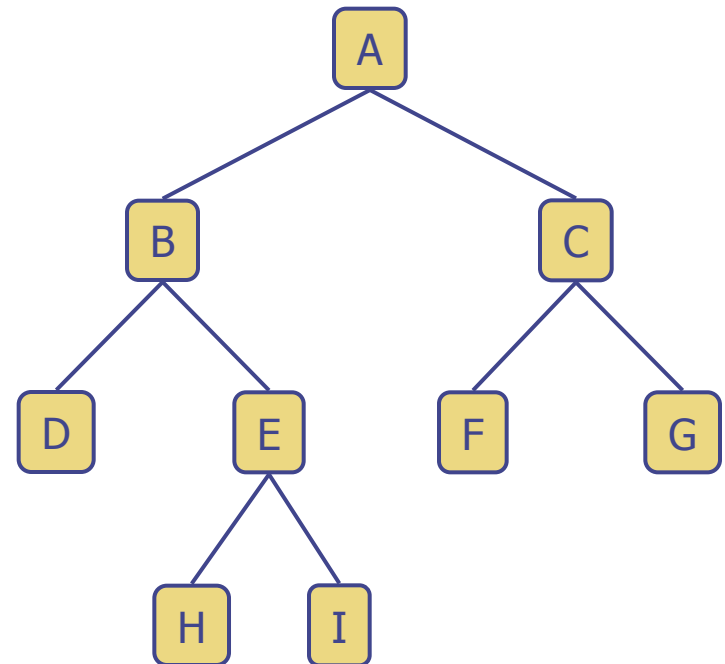
Algoritmo *postOrder(v)*
para cada filho w of v
postOrder(w)
visite(v)



Árvore binária

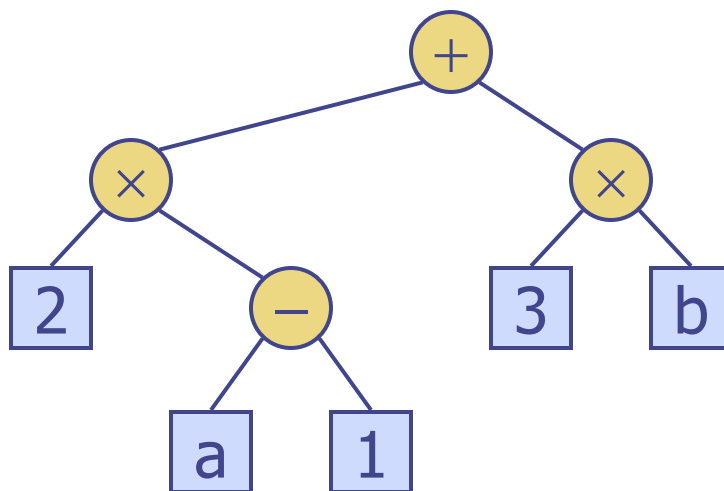
- ◆ Uma árvore binária é uma árvore com as seguintes propriedades:
 - Cada nó interno tem, no máximo, dois filhos
 - ◆ Árvore binária própria é aquela em que cada nó tem exatamente zero ou dois filhos
 - Os filhos de um nó é um par ordenado
- ◆ Chamamos os filhos de um nó de filho da esquerda e filho da direita
- ◆ Podemos, também, definir uma árvore binária recursivamente como:
 - uma árvore consistindo de um único nó, ou
 - Uma árvore cuja raiz tem um par ordenado de filho, cada um dos quais é uma árvore binária

- ◆ Aplicações:
 - Expressões aritméticas
 - Processo de decisão
 - busca



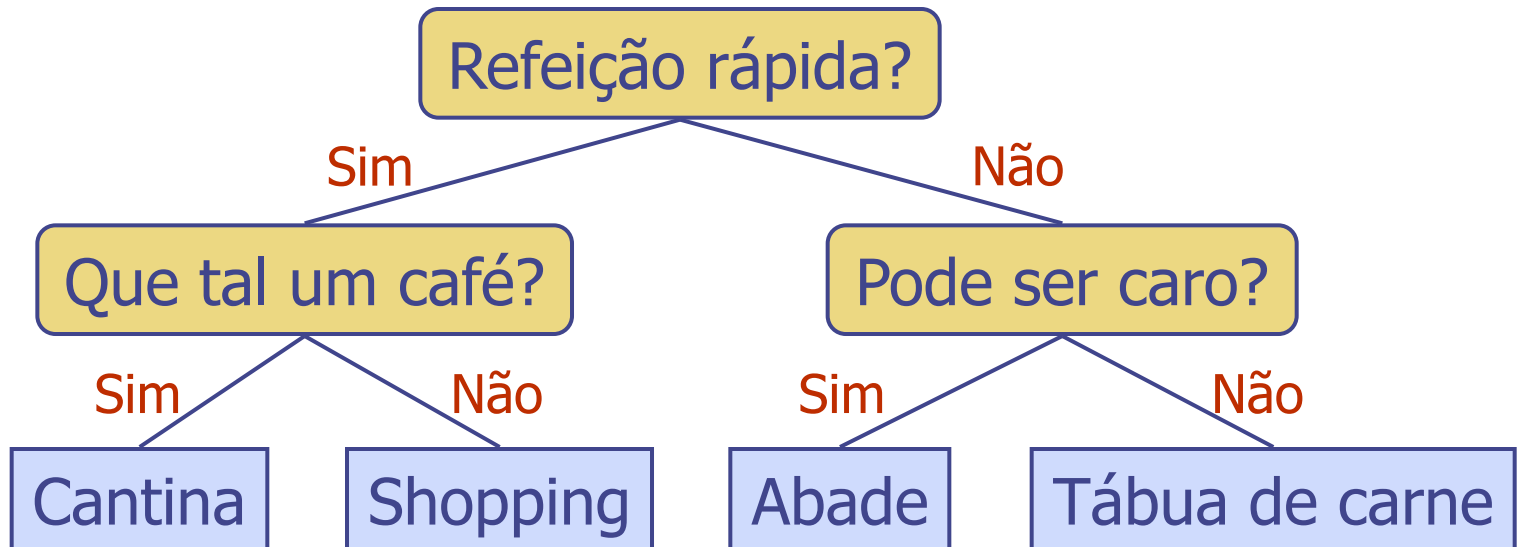
Árvore de expressões aritméticas

- ◆ Árvore binária associada com uma expressão aritmética
 - Nós internos: operadores
 - Nós externos: operandos
- ◆ Exemplo: árvore da expressão aritmética para a expressão $(2 \times (a - 1) + (3 \times b))$



Árvore de decisão

- ◆ Árvore binária associada com um processo de decisão
 - Nós internos: questões com respostas sim/não
 - Nós externos: decisões
- ◆ Exemplo: Onde jantar



Propriedades de AB (BT)

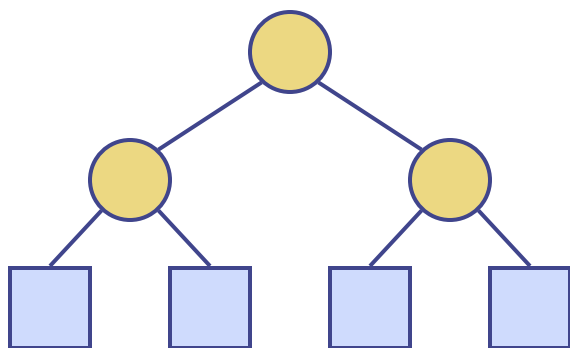
◆ Notação

n número de nós

e número de nós
externos

i número de nós
internos

h altura (*height*)



◆ Propriedades:

■ $e = i + 1$

■ $n = 2e - 1$

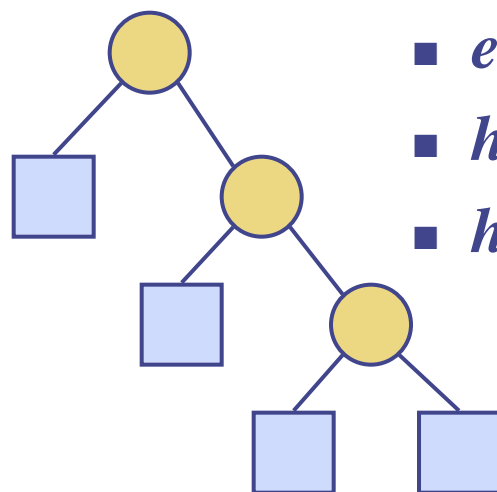
■ $h \leq i$

■ $h \leq (n - 1)/2$

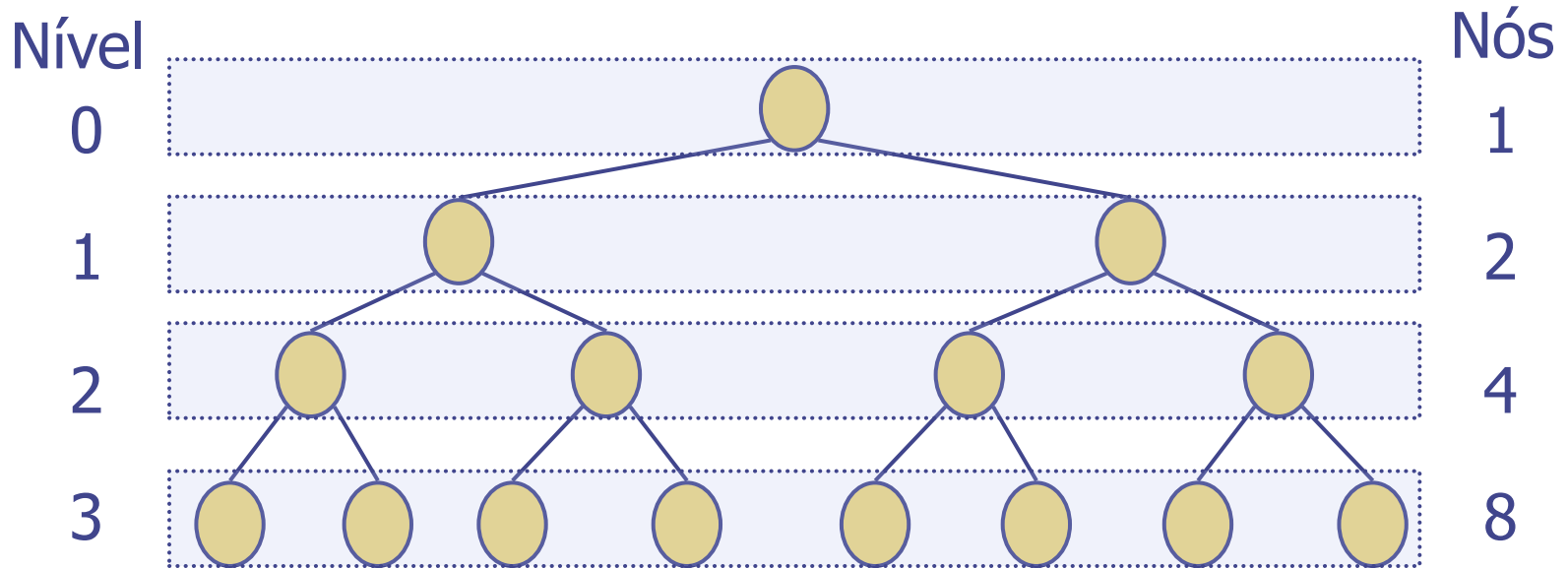
■ $e \leq 2^h$

■ $h \geq \log_2 e$

■ $h \geq \log_2 (n + 1) - 1$



Propriedades de AB (BT)



Número máximo de nós em um nível h é 2^h

Número total de nós é, no máximo é $2^{h+1} - 1$

TAD Árvore Binária

- ◆ O TAD ArvoreBinaria possui os métodos de árvore.
- ◆ Métodos adicionais:
 - Position `leftChild(p)`
 - Position `rightChild(p)`
 - boolean `hasLeft(p)`
 - boolean `hasRight(p)`
- ◆ Métodos de atualização podem ser definidos por estruturas de dados que implementam o TAD ArvoreBinaria

Travessia em ordem

- ◆ Na travessia inorder, um nó é visitado depois do filho da esquerda e antes do filho da direita
- ◆ Aplicação: Desenhar uma árvore binária
 - $x(v)$ = colocação de v
 - $y(v)$ = profundidade de v

Algoritmo *inOrder*(v)

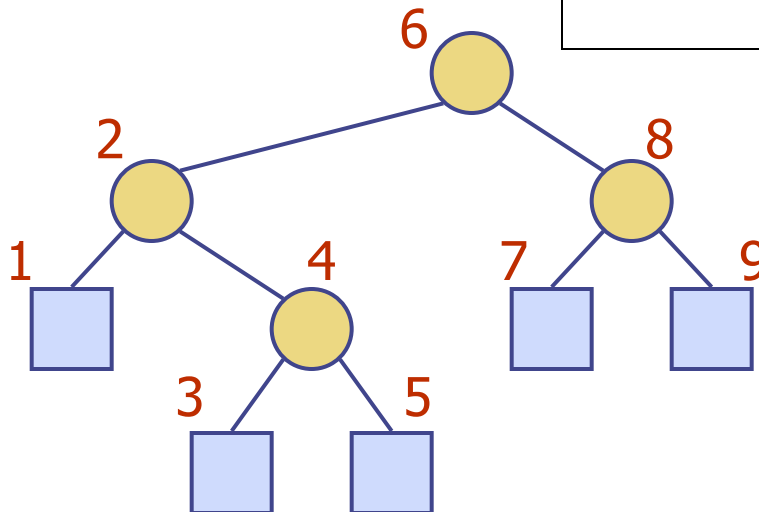
se (*isInternal* (v))

inOrder (*leftChild* (v))

visite(v)

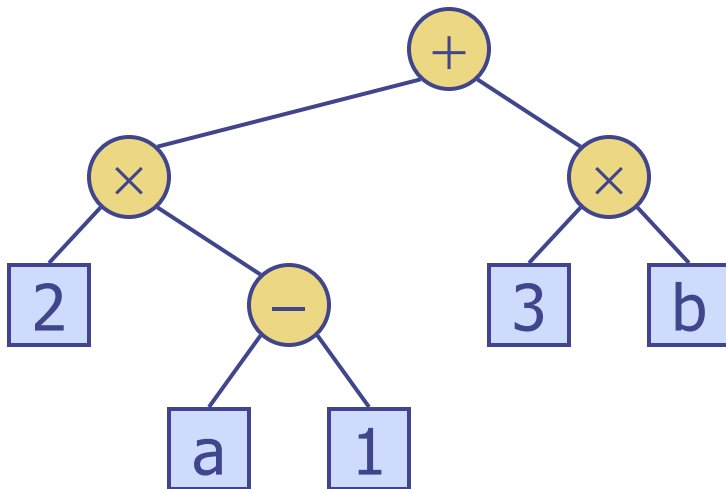
se (*isInternal* (v))

inOrder (*rightChild* (v))



Impressão de expressões aritm.

- ◆ Especialização de uma travessia inorder
 - Imprime operando/operador quando visita o nó
 - imprime "(" antes de visitar o filho da esquerda
 - imprime ")" depois de visitar o filho da direita



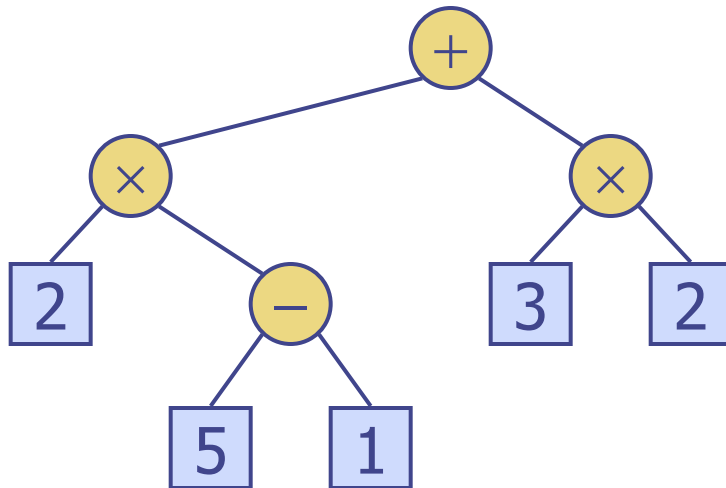
Algoritmo *printExpression(v)*

```
se (isInternal (v))
    print("(")
    inOrder (leftChild (v))
    print(v.element ())
se (isInternal (v) )
    inOrder (rightChild (v))
    print(")")
```

$((2 \times (a - 1)) + (3 \times b))$

Avaliação de expressões aritm.

- ◆ Especialização da travessia pós-ordem
 - Método recursivo retorna o valor de uma subárvore
 - Ao visitar um nó interno, combina os valores das subárvores



Algoritmo *evalExpr(v)*

se (*isExternal* (v))

return *v.element* ()

senão

x ← *evalExpr*(*leftChild* (v))

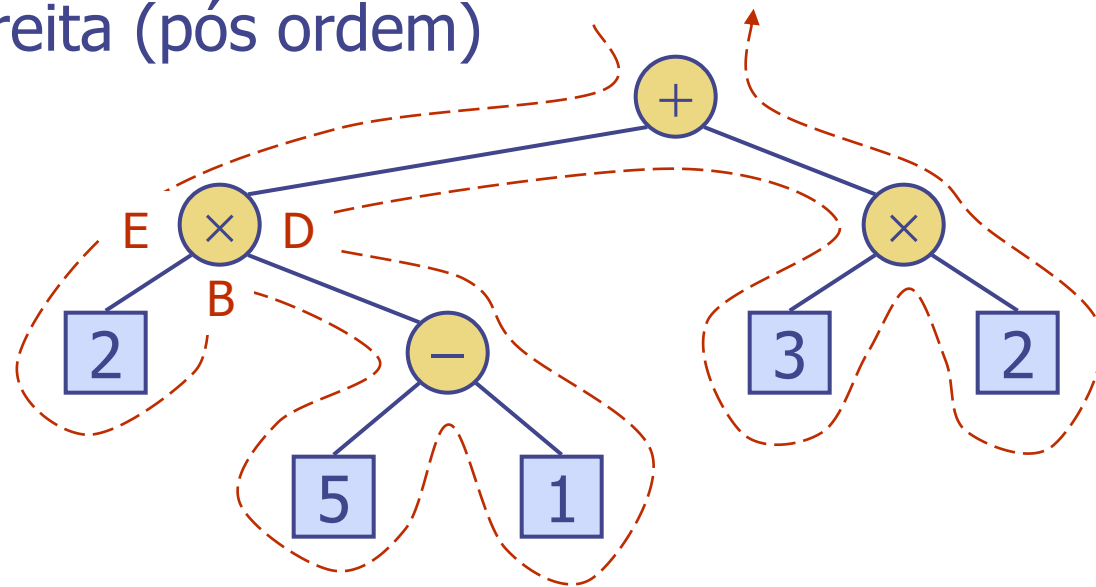
y ← *evalExpr*(*rightChild* (v))

◇ ← operador em *v*

return *x* ◇ *y*

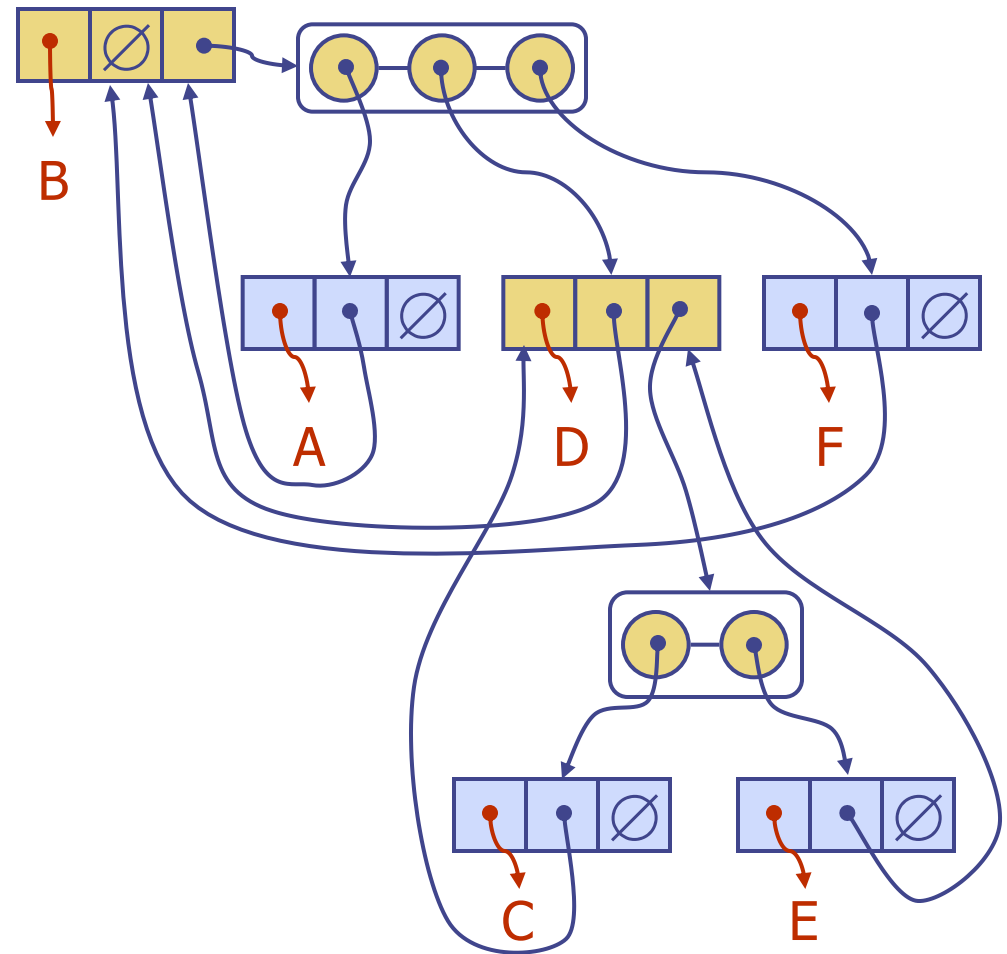
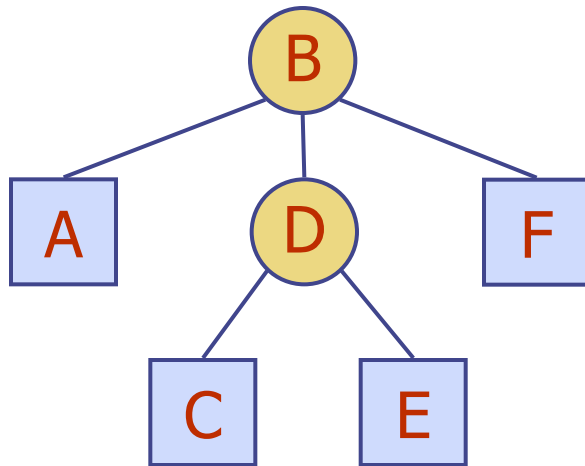
Travessia de Euler (Euler tour)

- ◆ Travessia genérica de árvores binárias
- ◆ Caminha pela árvore e visita cada nó 3 vezes:
 - pela esquerda (pré ordem)
 - Por baixo (em ordem)
 - pela direita (pós ordem)



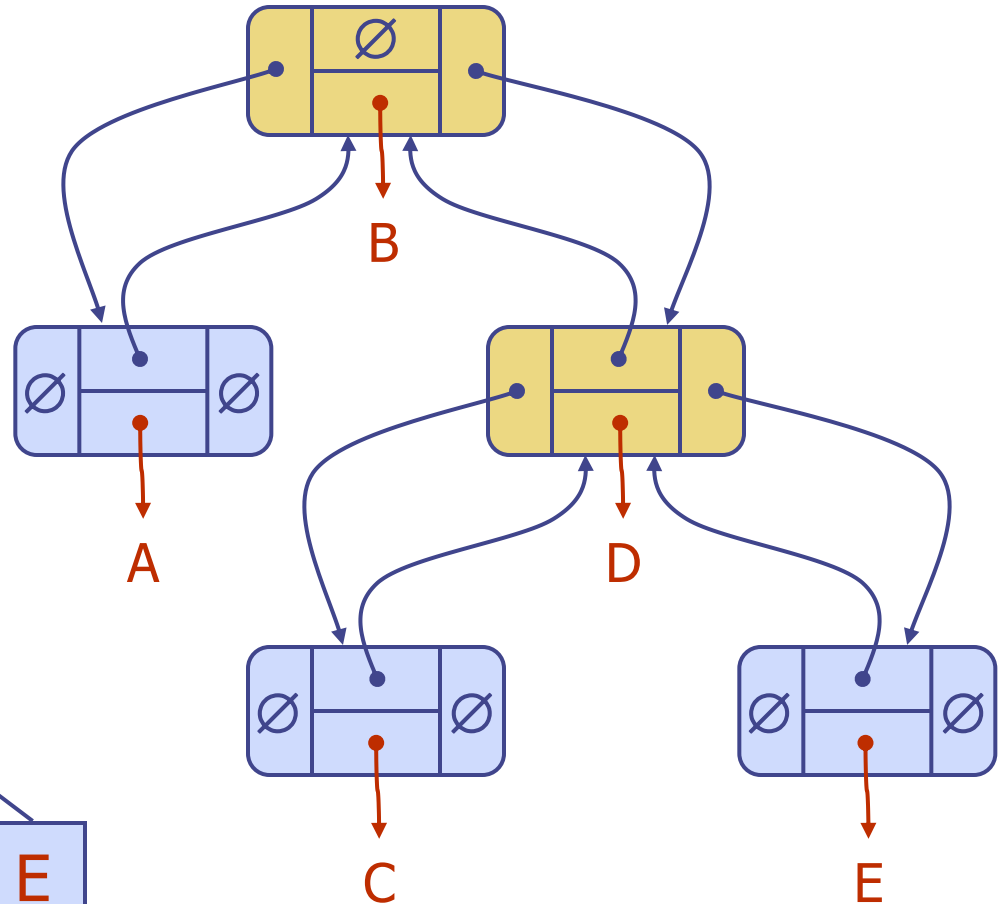
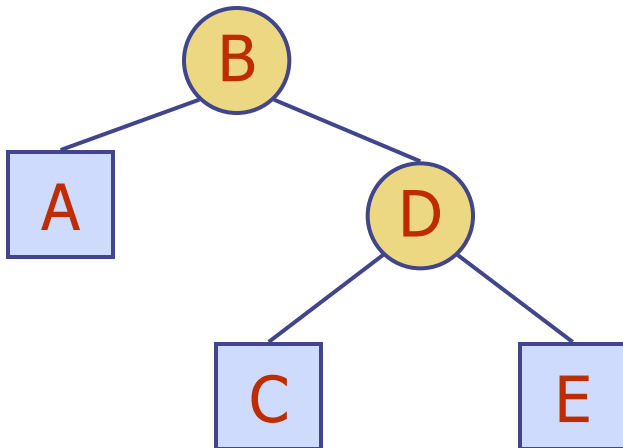
Estrutura de dados para árvores

- ◆ Um nó é um objeto que armazena
 - Elemento
 - Nó pai
 - Nós Filhos (Sequência, Vector, Array, etc)
- ◆ Objetos nós



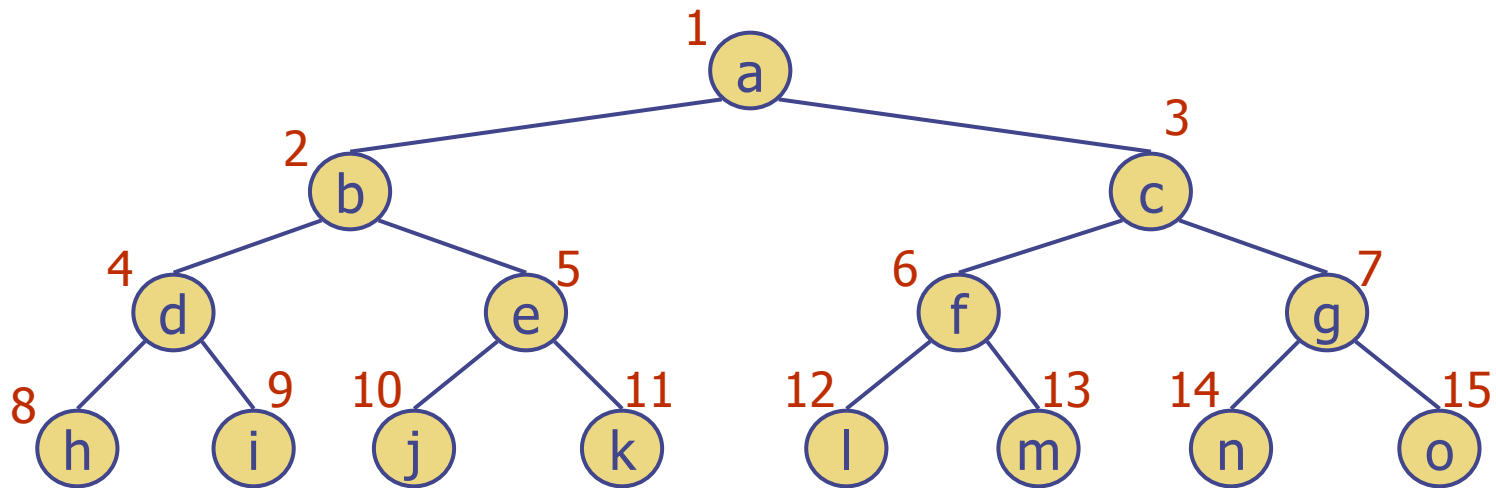
Estrutura de dados para AB

- ◆ Um nó é um objeto que armazena
 - Elemento
 - Nó pai
 - Filho da esquerda
 - Filho da direita
- ◆ Objetos nós



Estrutura de dados para AB

◆ Podemos usar um *array*



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15