

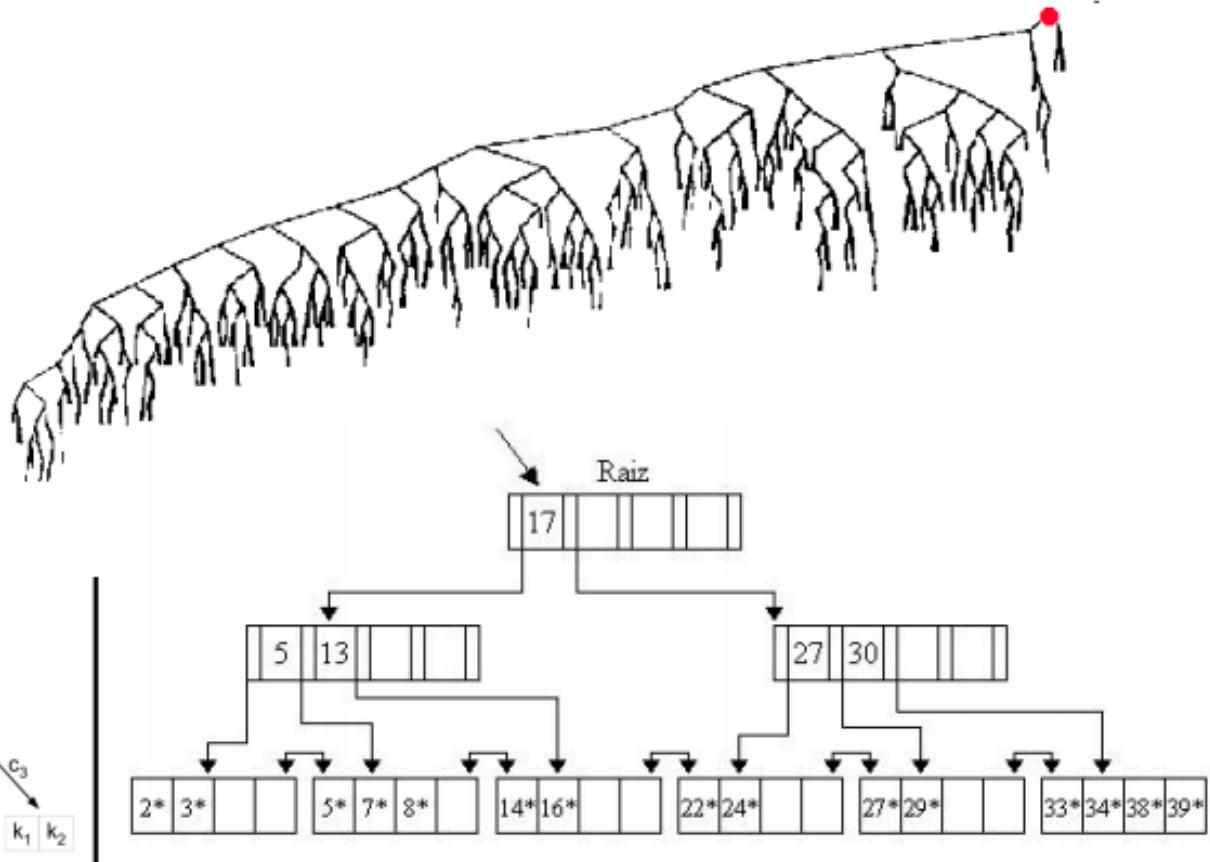
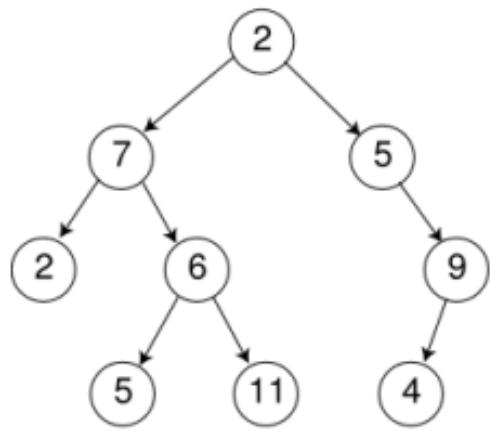
Árvores

Árvore Binária de Busca

(Fonte: Material adaptado dos Slides do prof. Monael.)

Árvores

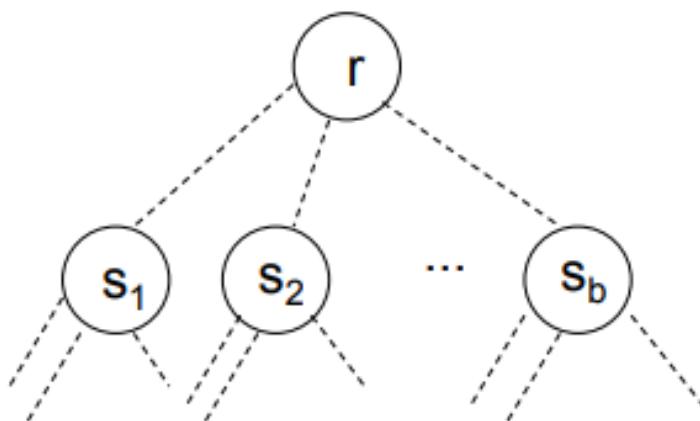
- Exemplos:



Árvores

Uma árvore enraizada T , ou simplesmente árvore, é um conjunto finito de elementos denominados nós tais que:

- $T = \emptyset$ é a árvore dita vazia. Ou
- Existe um nó especial r , chamado raiz de T ;
 - os nós restantes constituem um único conjunto, que pode ser vazio ou não. Esses são divididos em b ($b \geq 1$) conjuntos distintos não vazios que são as subárvores de r , cada subárvore a qual é, por sua vez, uma árvore.



Árvores

- Motivação:
 - Por que usar árvores?
 - Custo
 - É uma ED muito eficiente para armazenar informações
 - É particularmente adequada quando existe a necessidade de considerar todos ou alguma combinação de requisitos, tais como:
 - Acesso Direto e Seqüencial eficientes
 - Facilidade de Inserção e retirada de itens
 - Otimização da utilização da memória

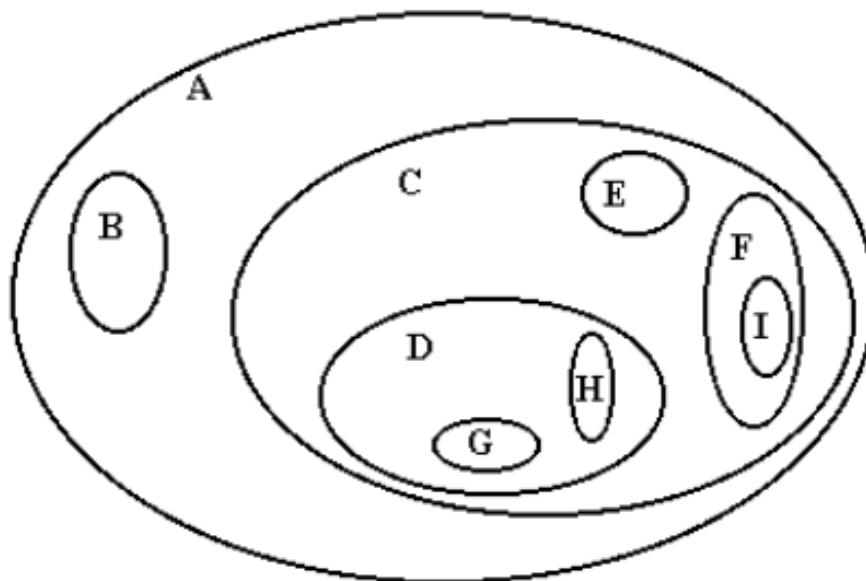
Árvores

- Motivação:
 - Por que usar árvores?
 - Custo

Operação	Listas	Árvores
Inserção	$O(n)$	$O(\log_b n)$
Remoção	$O(n)$	$O(\log_b n)$
Busca	$O(n)$	$O(\log_b n)$

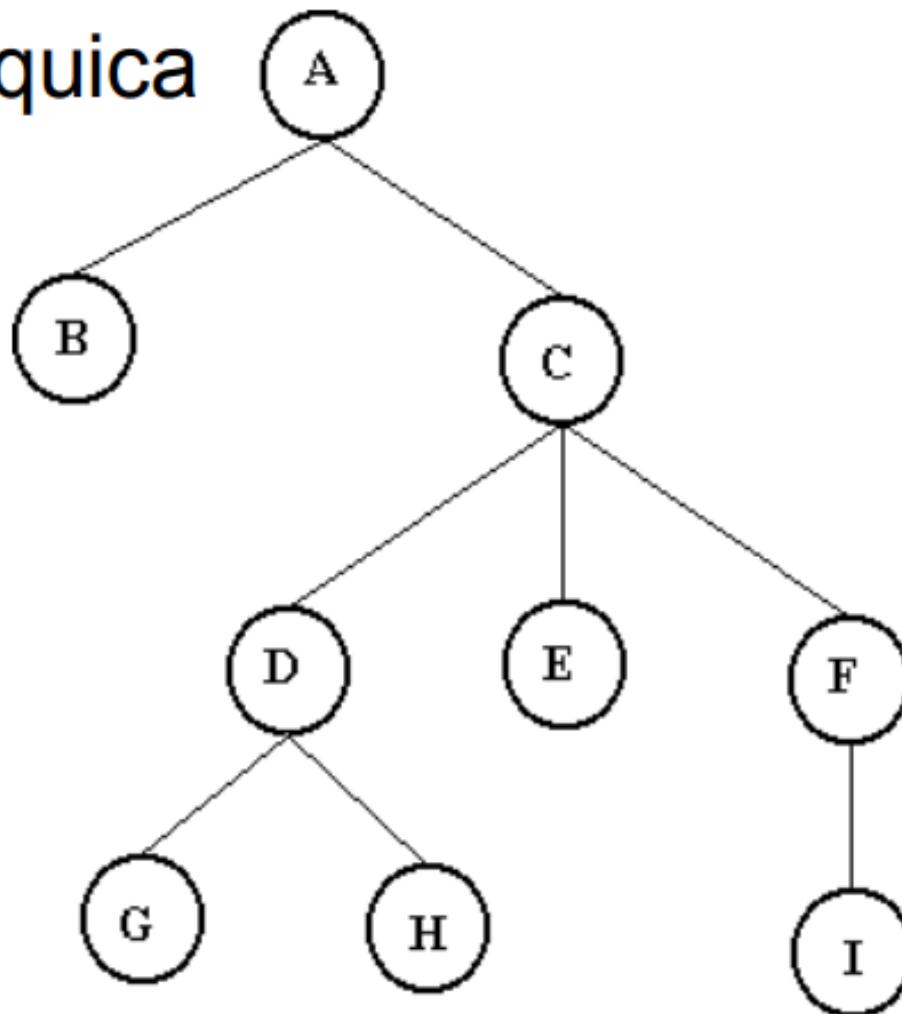
Árvores

- Exemplos de representações:
 - Parênteses Aninhados
 $(A (B) (C (D (G) (H)) (E) (F (I))))$
 - Diagrama de Inclusão



Árvores

- Exemplos de representações:
 - Hierárquica

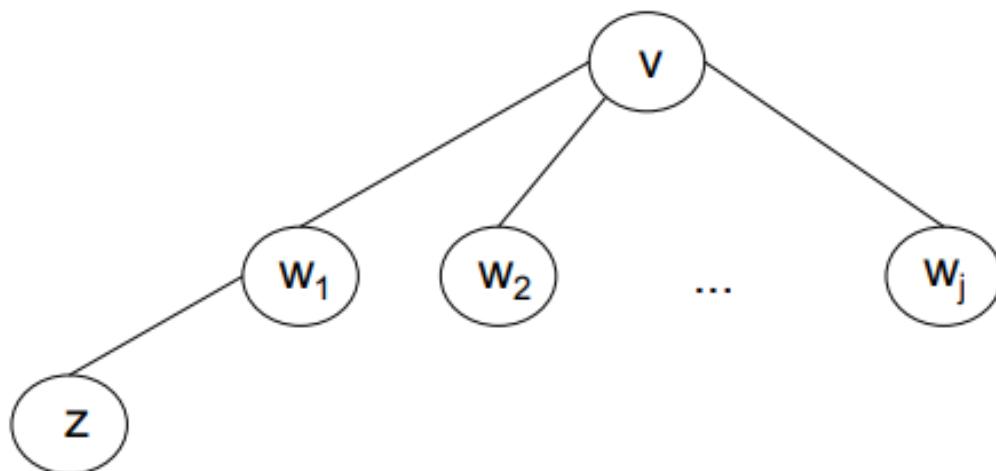


Árvores

- Definições

- Nós filhos, pais, tios, irmãos e avôs

- Seja v o nó raiz da subárvore T .
 - Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
 - v é chamado pai de w_1, w_2, \dots, w_j .
 - Os nós w_1, w_2, \dots, w_j são irmãos.
 - Se z é filho de w_1 então w_2 é tio de z e v é avô de z .

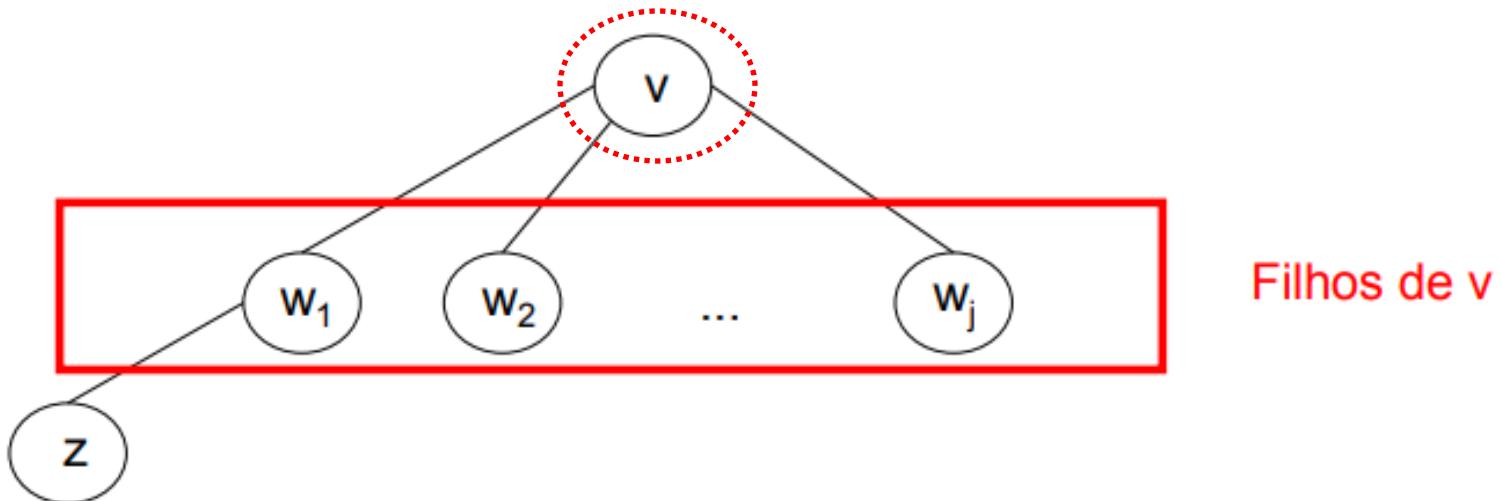


Árvores

- Definições

- Nós filhos, pais, tios, irmãos e avôs

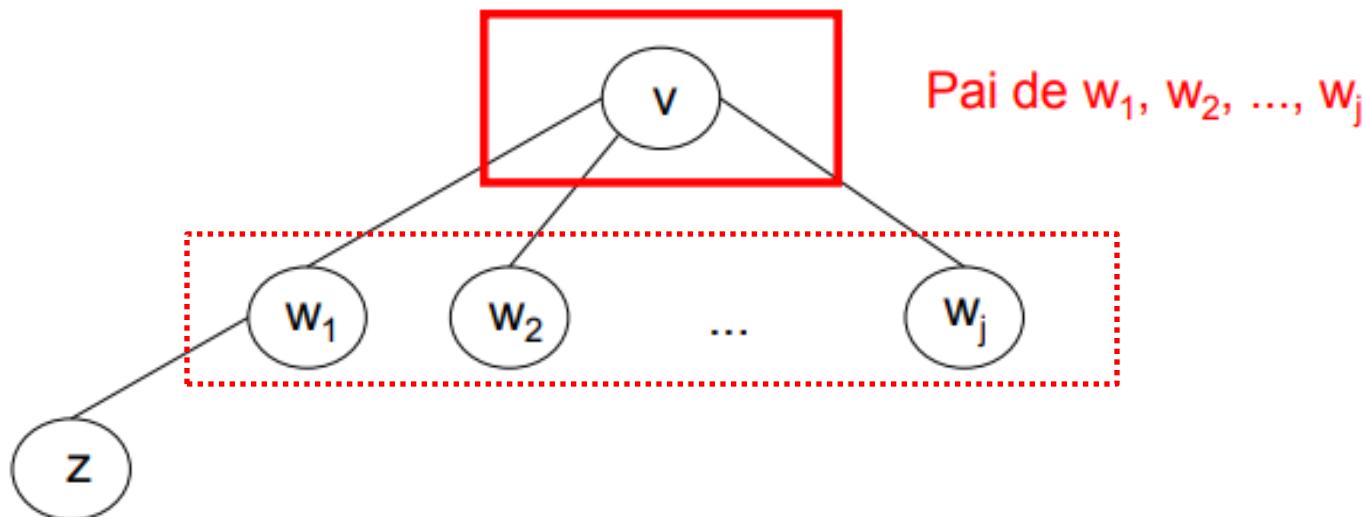
- Seja v o nó raiz da subárvore T .
 - Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
 - v é chamado pai de w_1, w_2, \dots, w_j .
 - Os nós w_1, w_2, \dots, w_j são irmãos.
 - Se z é filho de w_1 então w_2 é tio de z e v é avô de z .



Árvores

- Definições
 - Nós filhos, pais, tios, irmãos e avôs

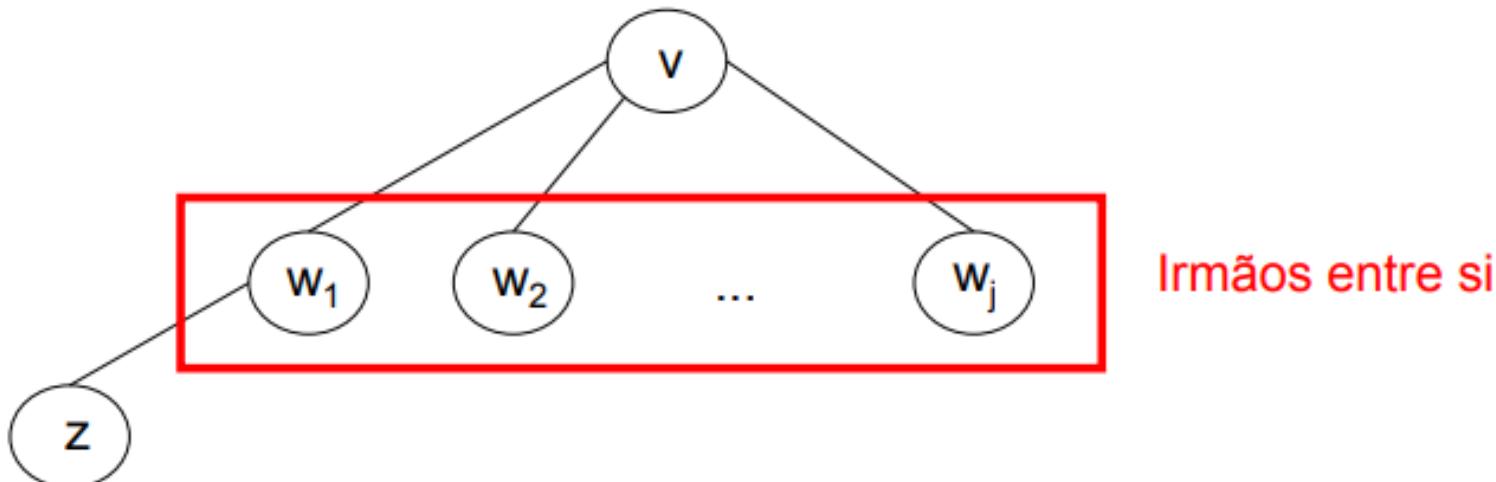
- Seja v o nó raiz da subárvore T .
- Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
- v é chamado pai de w_1, w_2, \dots, w_j .
- Os nós w_1, w_2, \dots, w_j são irmãos.
- Se z é filho de w_1 então w_2 é tio de z e v é avô de z .



Árvores

- Definições
 - Nós filhos, pais, tios, irmãos e avôs

- Seja v o nó raiz da subárvore T .
- Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
- v é chamado pai de w_1, w_2, \dots, w_j .
- Os nós w_1, w_2, \dots, w_j são irmãos.
- Se z é filho de w_1 então w_2 é tio de z e v é avô de z .

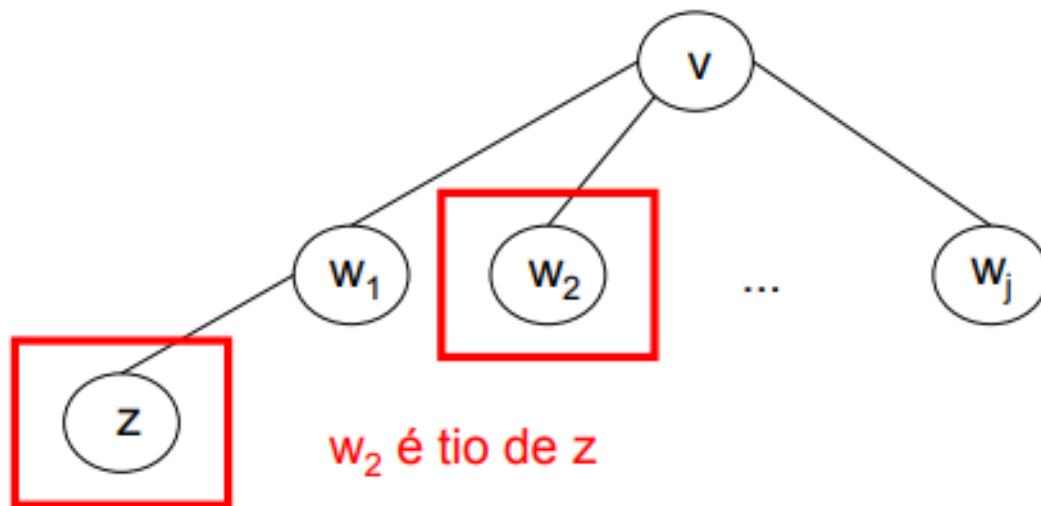


Árvores

- Definições

- Nós filhos, pais, tios, irmãos e avôs

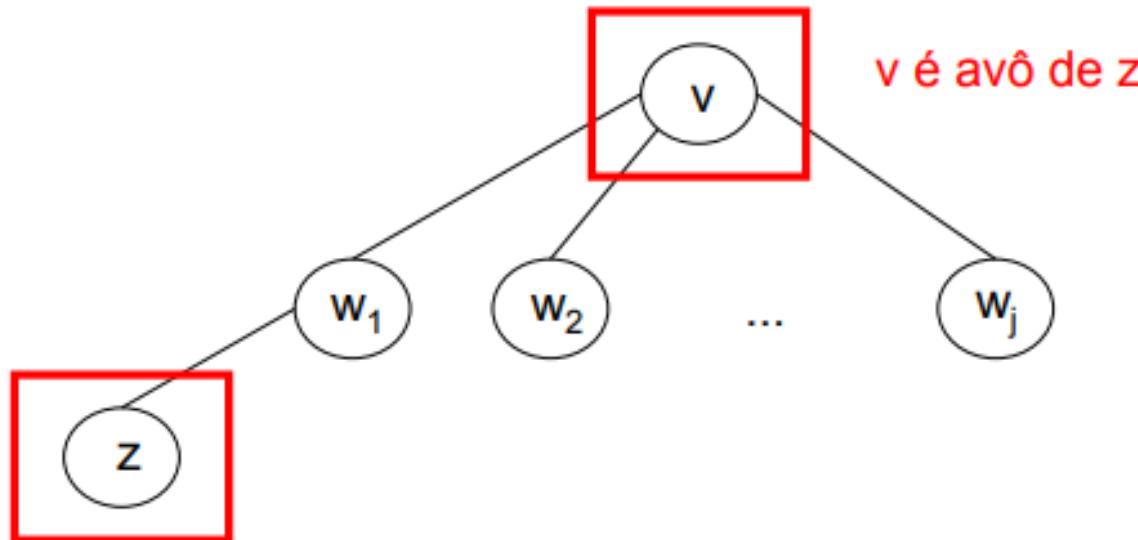
- Seja v o nó raiz da subárvore T .
 - Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
 - v é chamado pai de w_1, w_2, \dots, w_j .
 - Os nós w_1, w_2, \dots, w_j são irmãos.
 - Se z é filho de w_1 então w_2 é tio de z e v é avô de z .



Árvores

- Definições
 - Nós filhos, pais, tios, irmãos e avôs

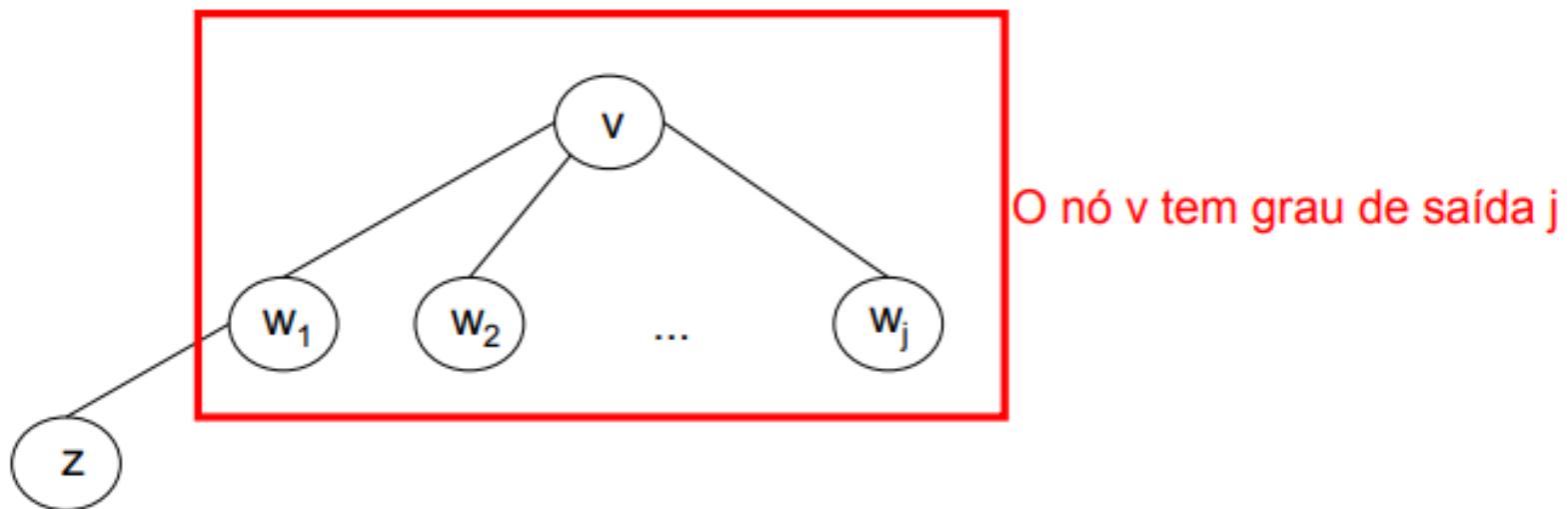
- Seja v o nó raiz da subárvore T .
- Os nós raízes w_1, w_2, \dots, w_j das subárvores de v são chamados filhos de v .
- v é chamado pai de w_1, w_2, \dots, w_j .
- Os nós w_1, w_2, \dots, w_j são irmãos.
- Se z é filho de w_1 então w_2 é tio de z e v é avô de z .



Árvores

- Definições
 - Grau de saída, descendente e ancestral

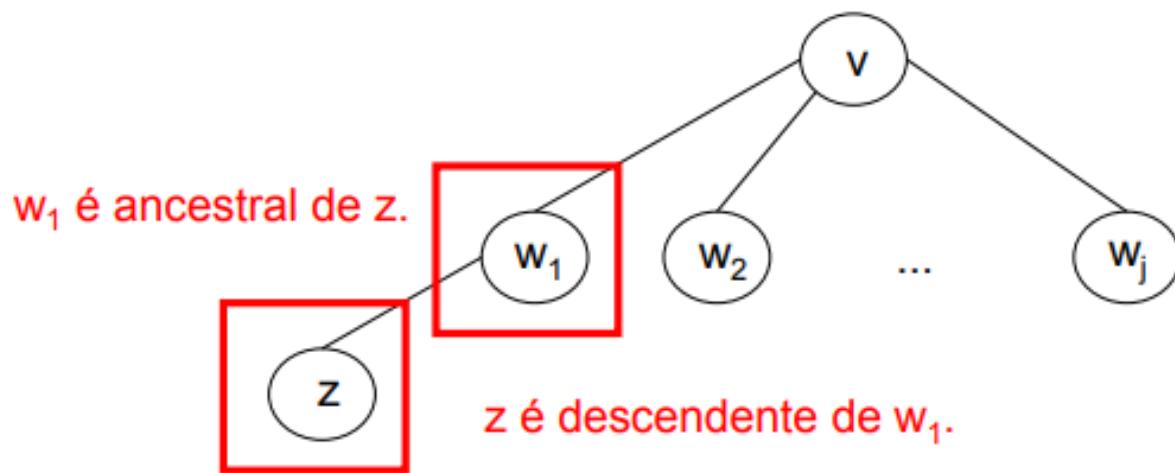
- O número de filhos de um nó é chamado grau de saída desse nó.
- Se x pertence à subárvore de v , então, x é descendente de v e v é ancestral de x .



Árvores

- Definições
 - Grau de saída, descendente e ancestral

- O número de filhos de um nó é chamado grau de saída desse nó.
- Se x pertence à subárvore de v , então, x é descendente de v e v é ancestral de x .

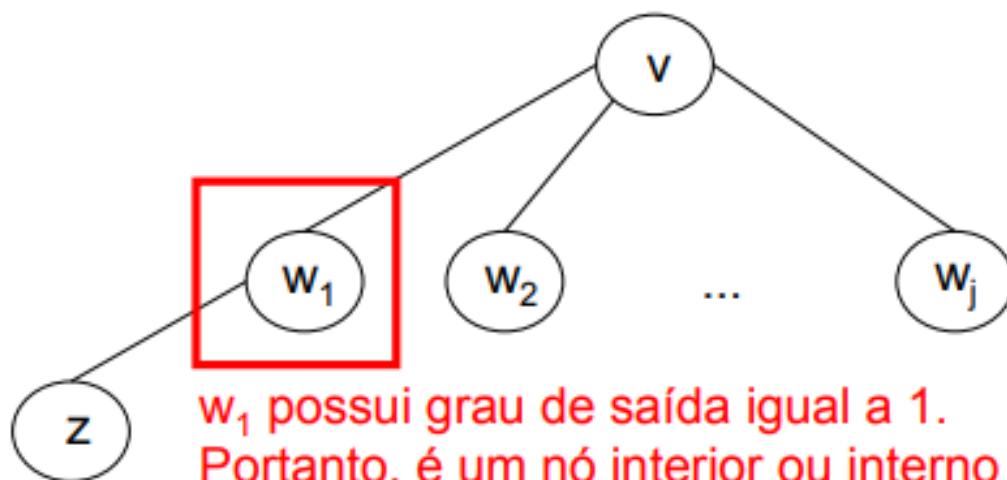


Árvores

- Definições

- Nó folha e Nó interior

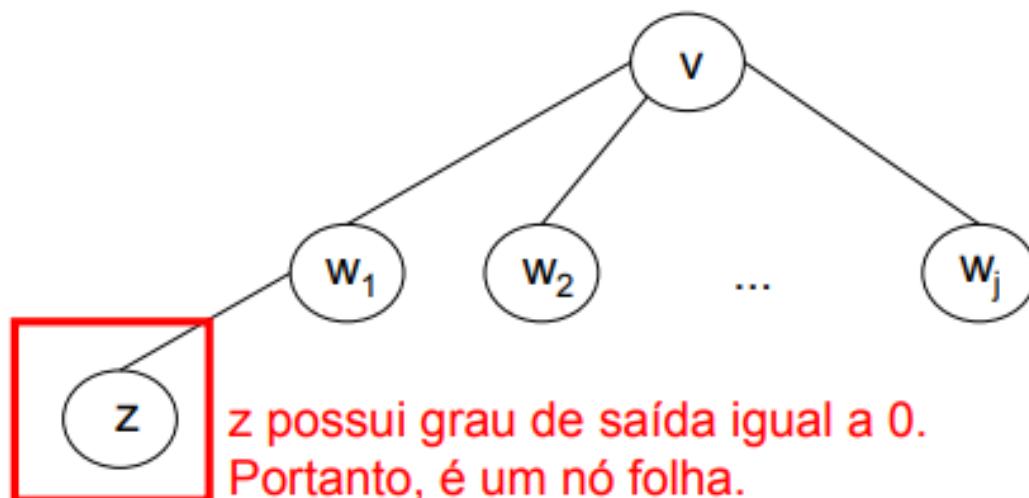
- Um nó que não possui descendentes é chamado de nó folha, ou seja, um nó folha é aquele com grau de saída nulo.
 - Um nó que não é folha (isto é, possui grau de saída diferente de zero) é chamado nó interior ou nó interno.



Árvores

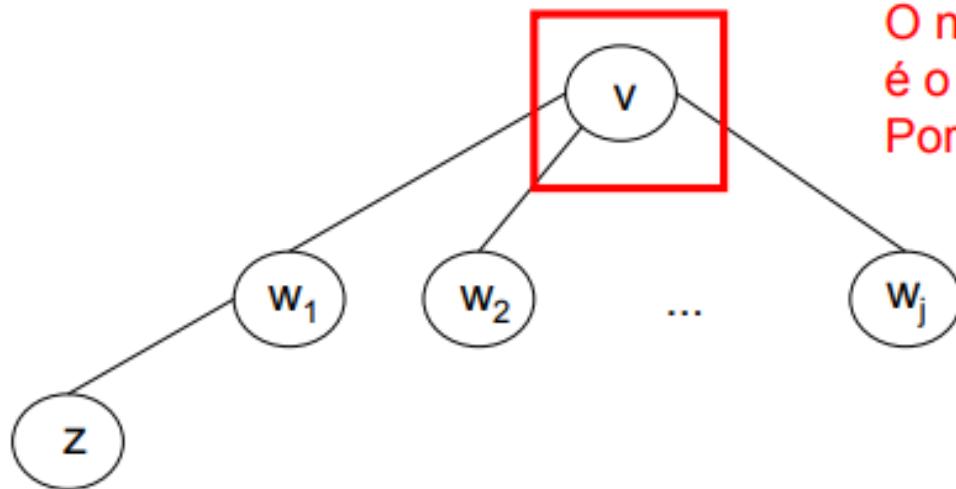
- Definições
 - Nó folha e Nó interior

- Um nó que não possui descendentes é chamado de nó folha, ou seja, um nó folha é aquele com grau de saída nulo.
- Um nó que não é folha (isto é, possui grau de saída diferente de zero) é chamado nó interior ou nó interno.



Árvores

- Definições
 - Grau de uma árvore
 - O grau de uma árvore é o máximo entre os graus de seus nós.



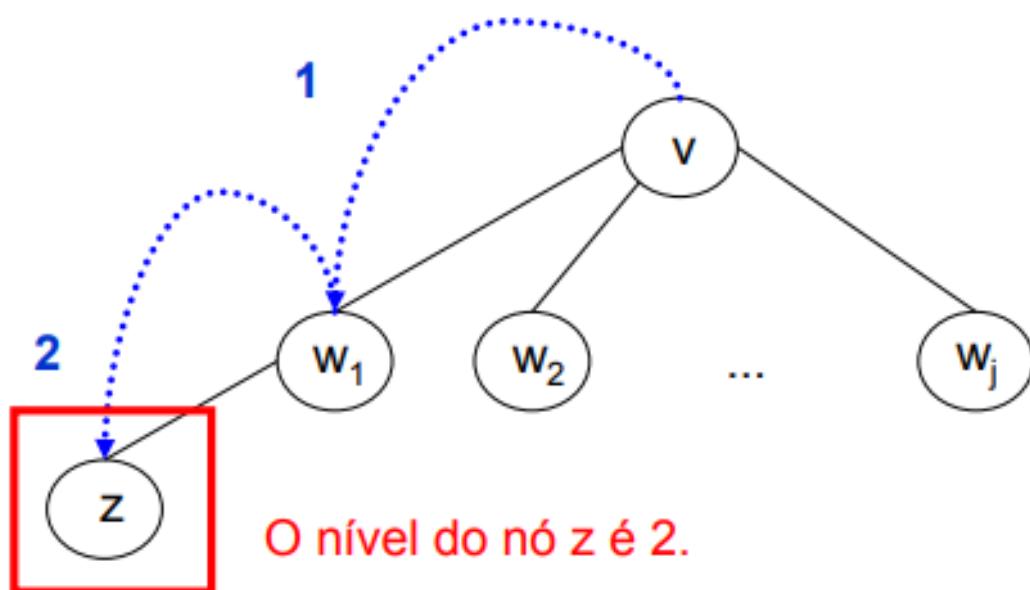
O nó v de grau de saída j, j>2, é o nó de maior grau na árvore. Portanto, j é o grau da árvore.

Árvores

- Definições

- Nível de um Nó e Altura de um Nó

- O nível ou profundidade, de um nó é o número de nós do caminho da raiz até o nó. O nível da raiz, é portanto, 0.
 - A altura de um nó v é o número de nós no maior caminho de v até um de seus descendentes. As folhas têm altura 0.

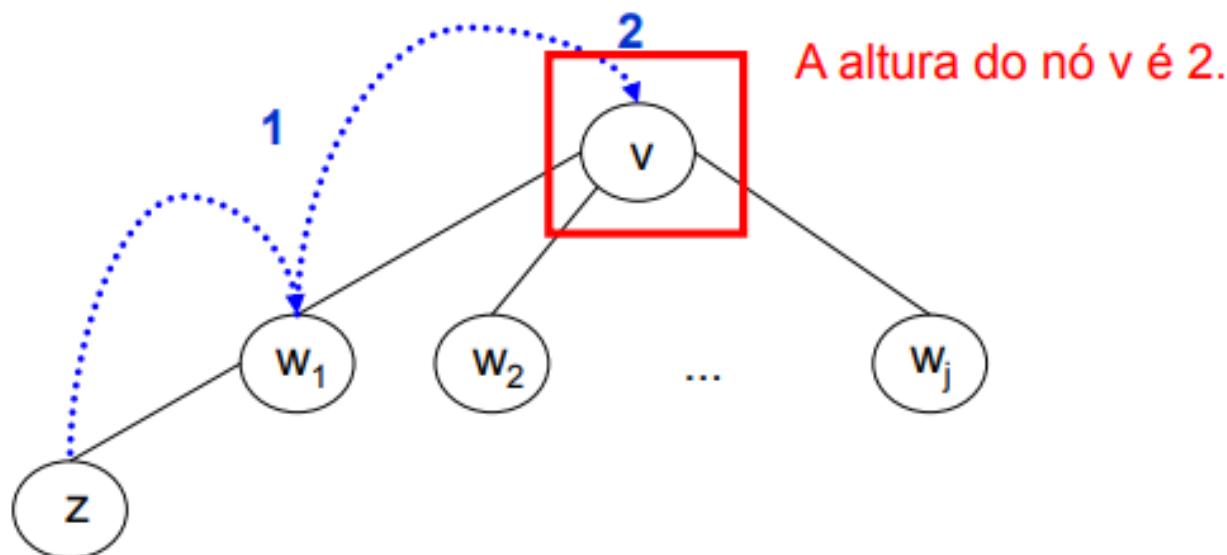


Árvores

- Definições

- Nível de um Nó e Altura de um Nó

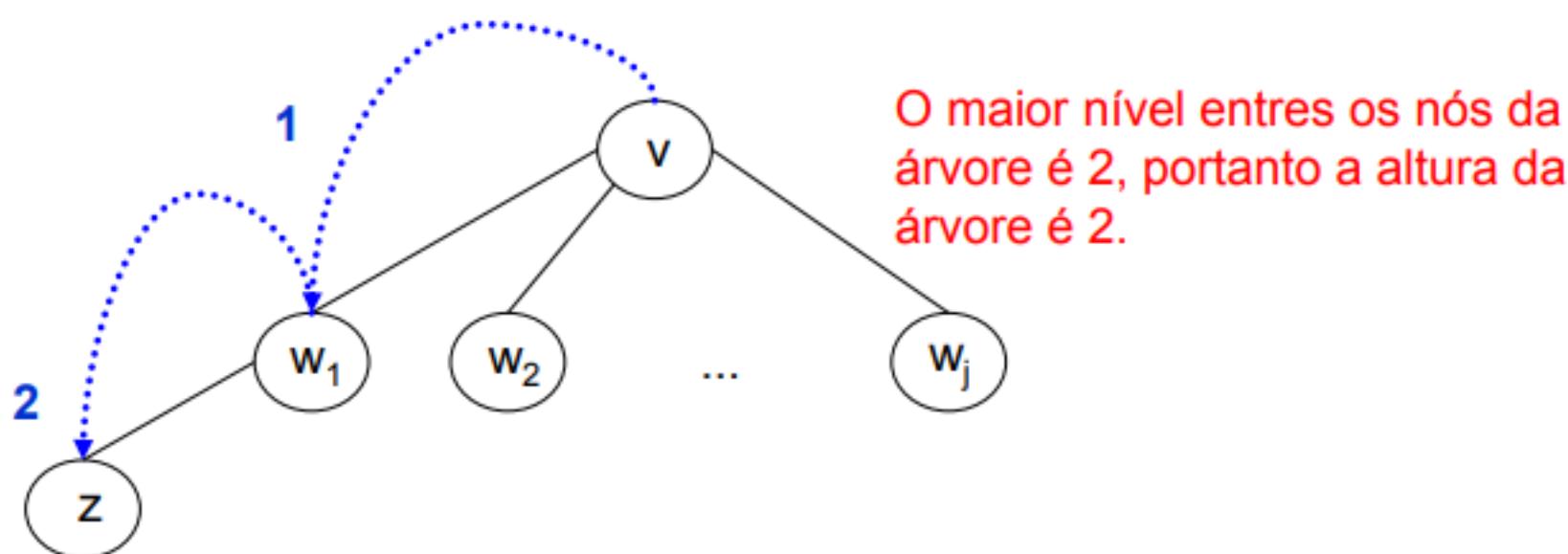
- O nível ou profundidade, de um nó é o número de nós do caminho da raiz até o nó. O nível da raiz, é portanto, 0.
 - A altura de um nó v é o número de nós no maior caminho de v até um de seus descendentes. As folhas têm altura 0.



Árvores

- Definições
 - Nível da Raiz e Altura de uma Árvore

- O nível da raiz é 0.
- A altura de uma árvore é igual ao máximo nível de seus nós. Representa-se a altura de árvore T por $h(T)$.

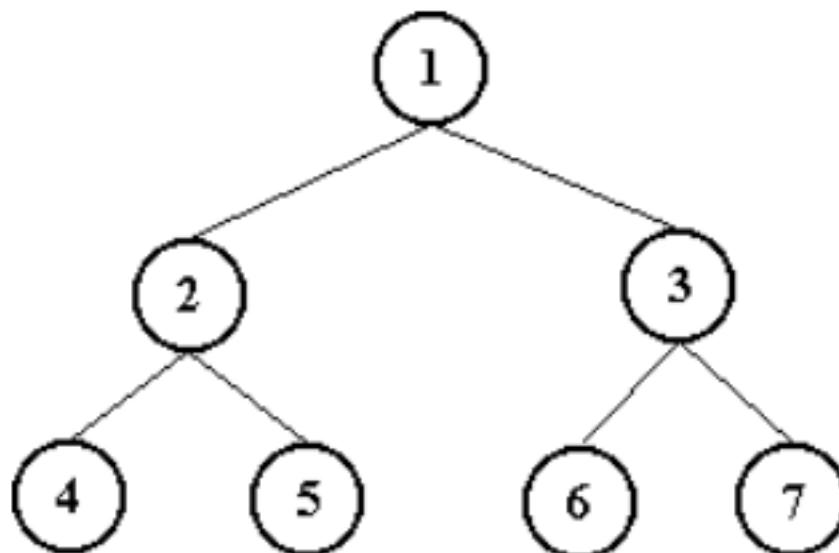


Árvores

- Definições
 - Árvore Ordenada
 - Uma árvore ordenada é aquela na qual os filhos de cada nó estão ordenados. Assume-se ordenação da esquerda para a direita

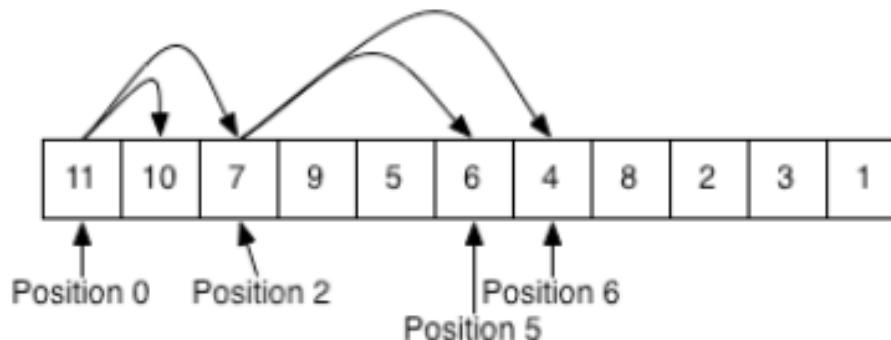
Árvores

- Definições
 - Árvore Cheia (ou completa)
 - Uma árvore de grau d é uma árvore cheia se possui o número máximo de nós, isto é, todos os nós tem número máximo de filhos exceto as folhas, e todas as folhas estão na mesma altura.

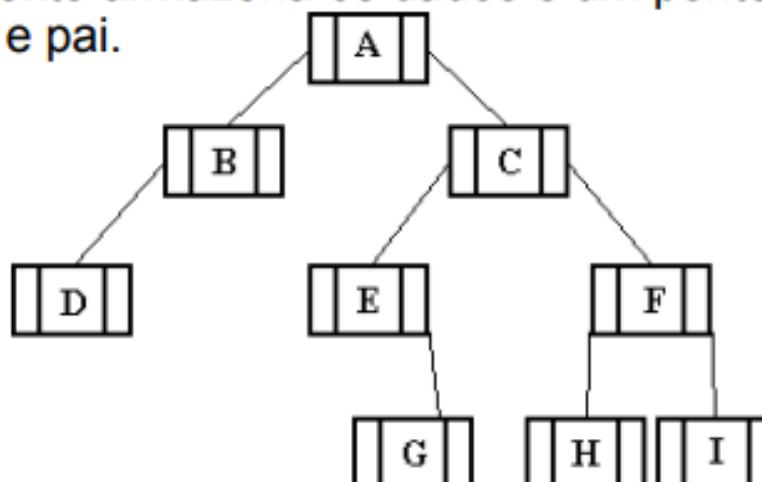


Árvores

- Quanto a implementação, as árvores podem ser:
 - Estáticas**: Os elementos são armazenados em um vetor.



- Dinâmicas**: Os elementos são alocados dinamicamente conforme necessidade. Cada elemento armazena os dados e um ponteiro para o filho da direita, esquerda e pai.

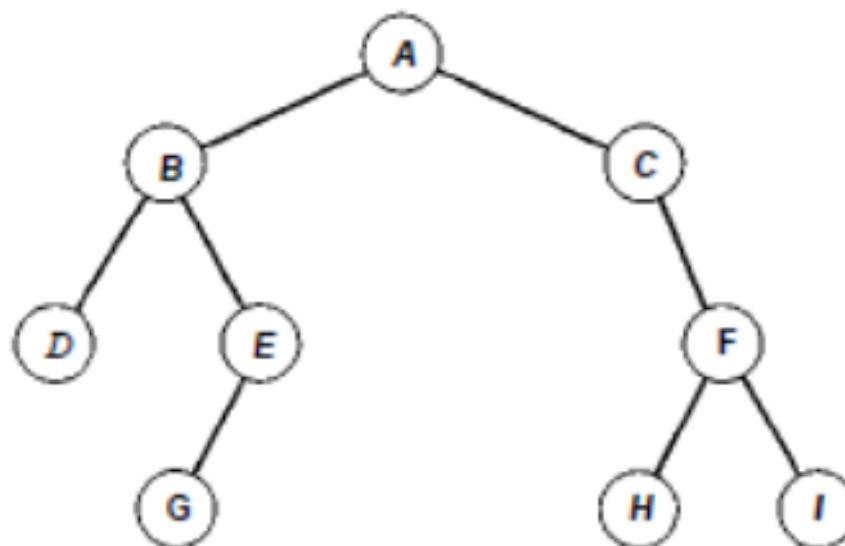


Árvores

- Principais Estrutura de Dados do tipo Árvore:
 - Árvore Binária
 - Árvore AVL
 - Árvore B
 - Árvore B+
 - Árvore Rubro-Negra
 - Heaps *

Árvore Binária

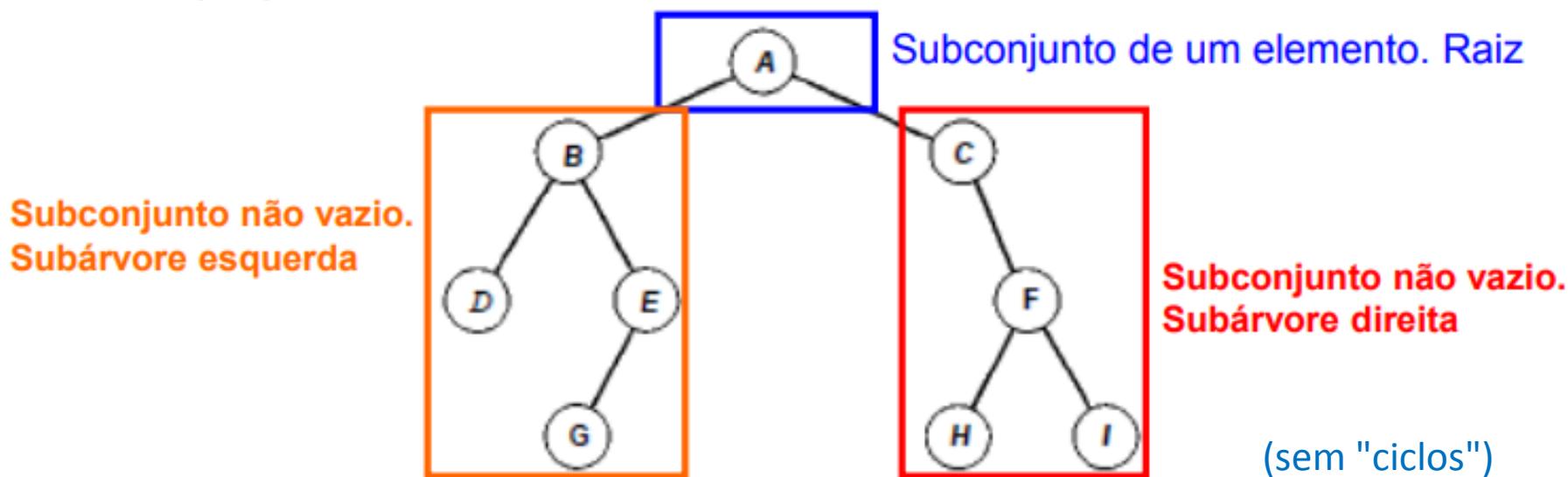
- É um conjunto finito de elementos que está vazio ou partitionado em 3 subconjuntos disjuntos.
 - Um subconjunto de apenas um elemento chamado raiz.
 - Outro subconjunto vazio ou não chamado subárvore direita, que por si só é uma árvore binária.
 - Finalmente, um subconjunto vazio ou não chamado subárvore esquerda, que por si só é uma árvore binária.



(sem "ciclos")

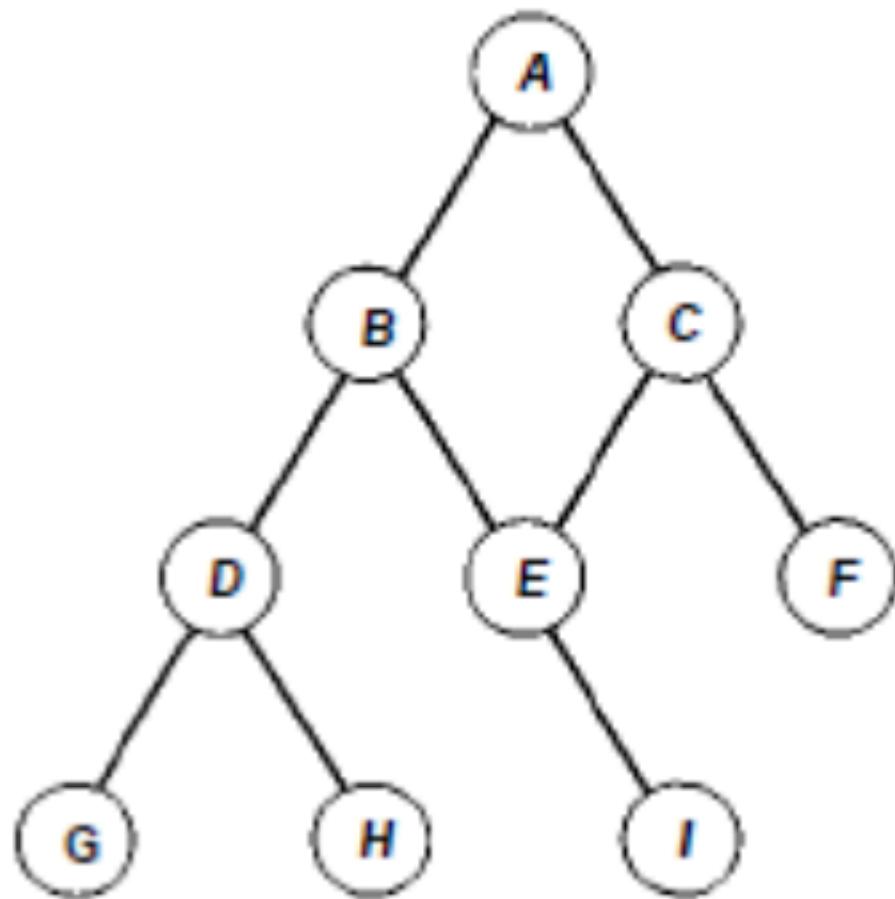
Árvore Binária

- É um conjunto finito de elementos que está vazio ou partitionado em 3 subconjuntos disjuntos.
 - Um subconjunto de apenas um elemento chamado raiz.
 - Outro subconjunto vazio ou não chamado subárvore direita, que por si só é uma árvore binária.
 - Finalmente, um subconjunto vazio ou não chamado subárvore esquerda, que por si só é uma árvore binária.



Árvore Binária

- É árvore binária?



NÃO

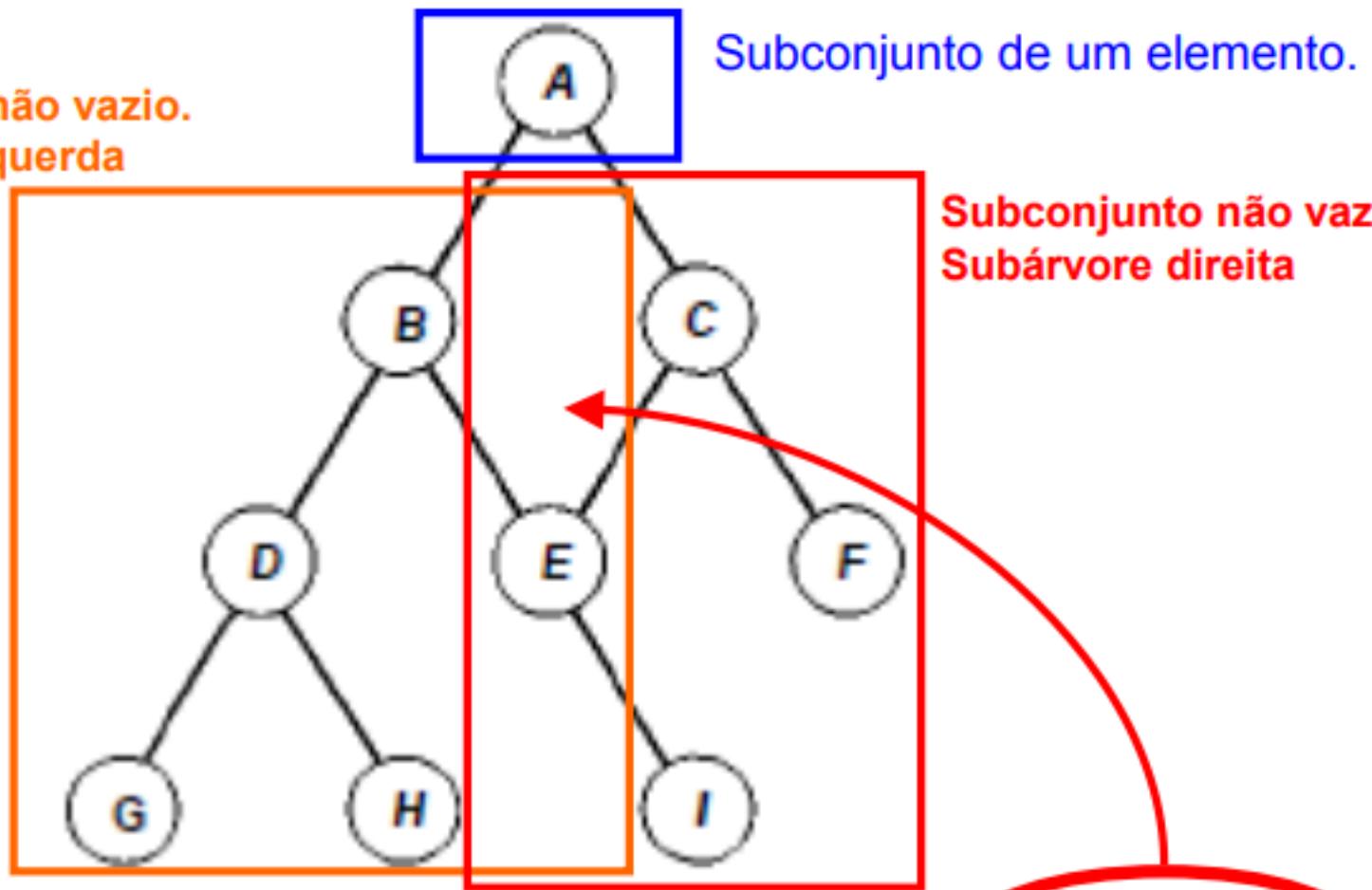
Árvore Binária

- É árvore binária?

Subconjunto não vazio.
Subárvore esquerda

Subconjunto de um elemento. Raiz

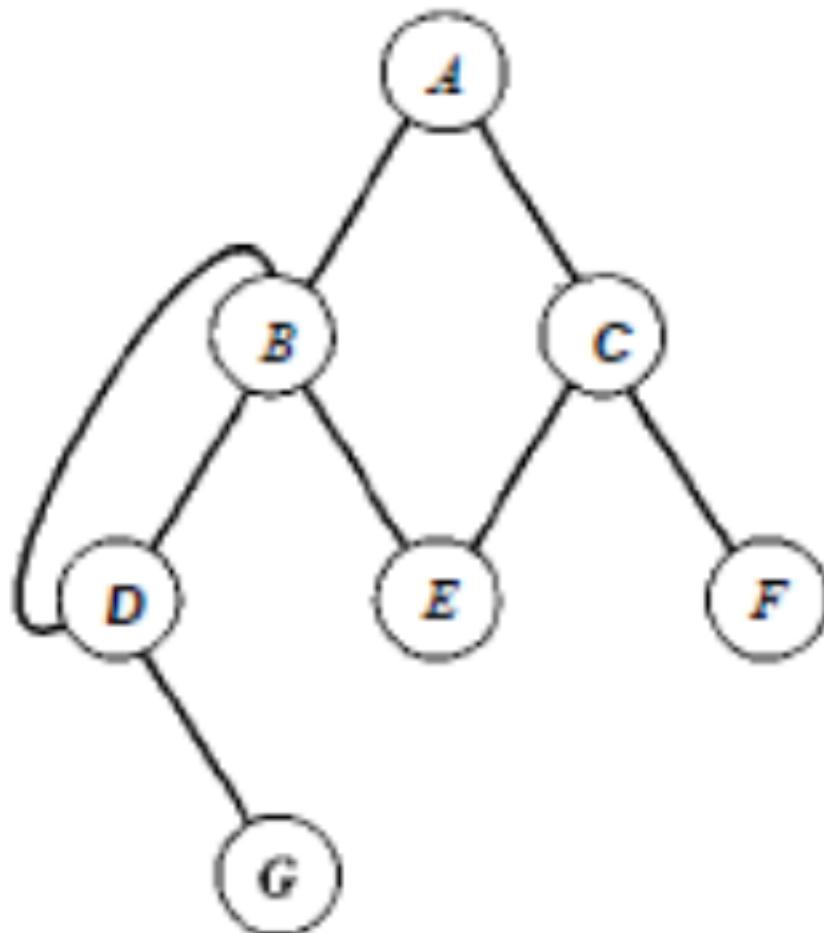
Subconjunto não vazio.
Subárvore direita



particionado em 3 subconjuntos disjuntos

Árvore Binária

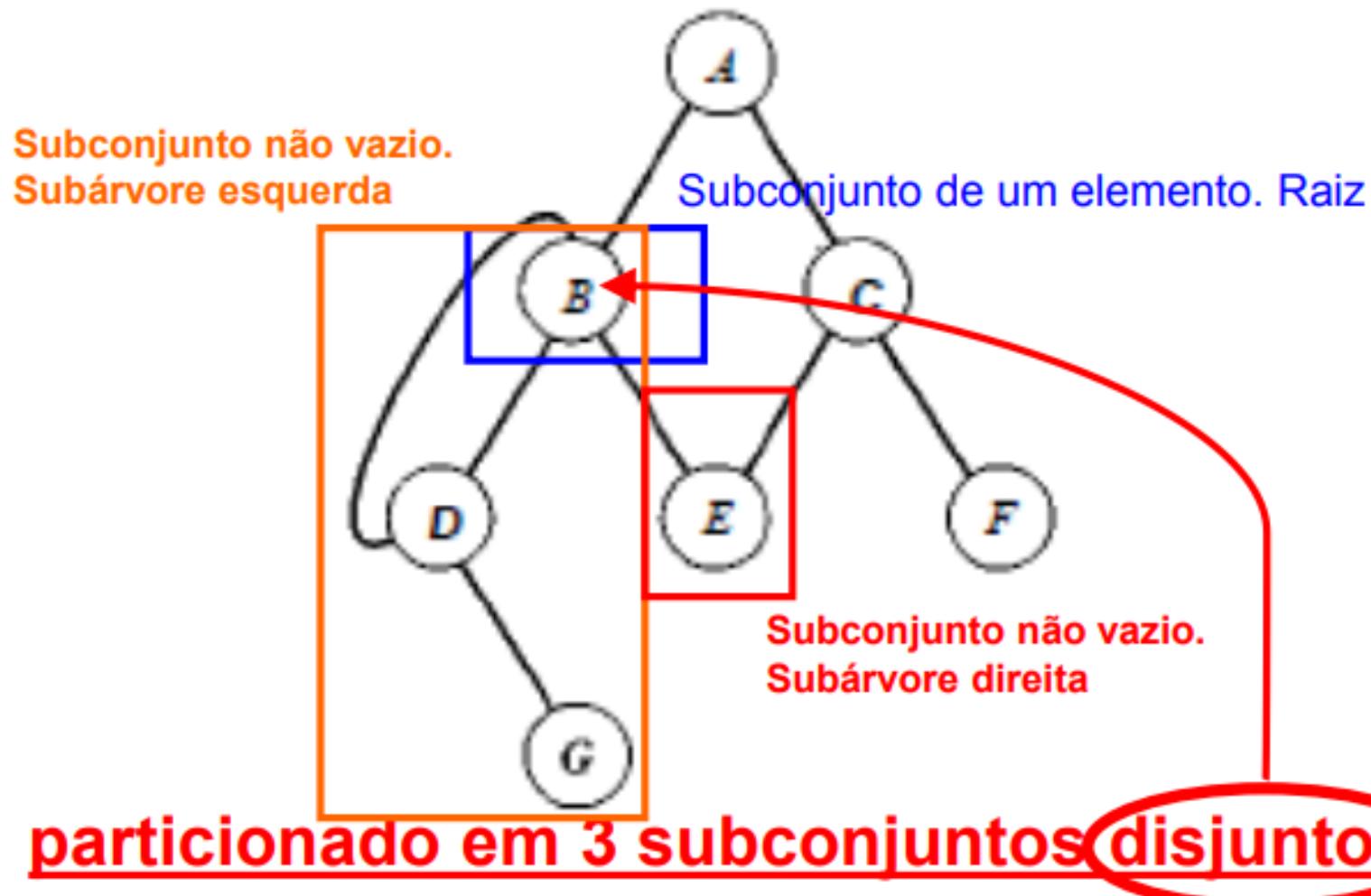
- É árvore binária?



NÃO

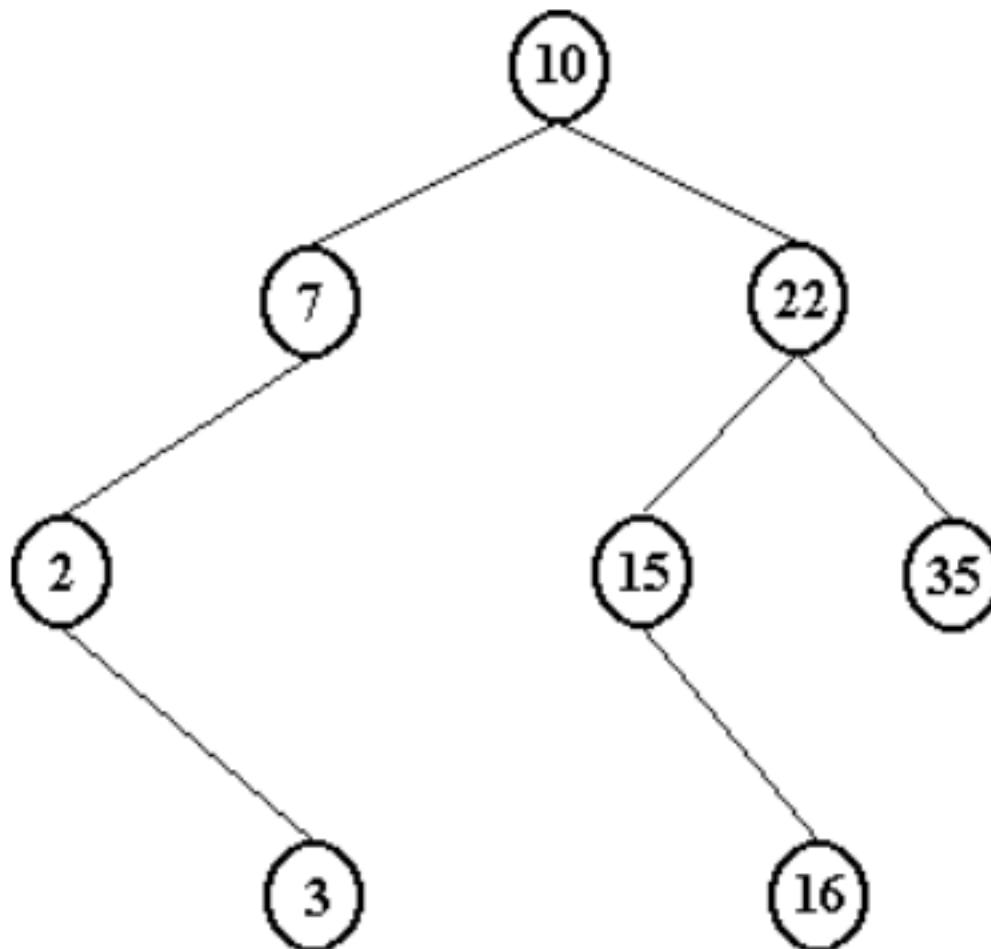
Árvore Binária

- É árvore binária?



Árvore Binária

- É árvore binária?



SIM

Árvore Binária

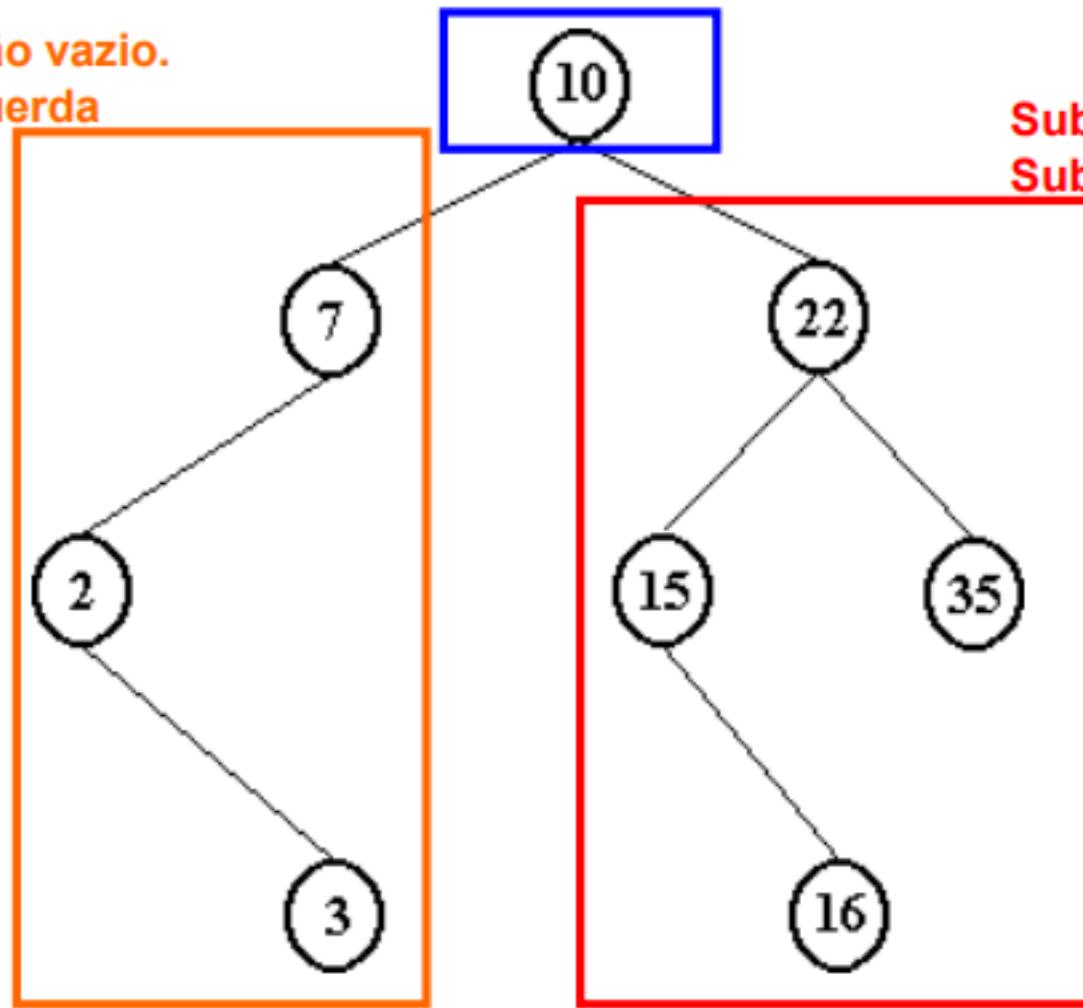
- É árvore binária?

Subconjunto de um elemento. Raiz

Subconjunto não vazio.

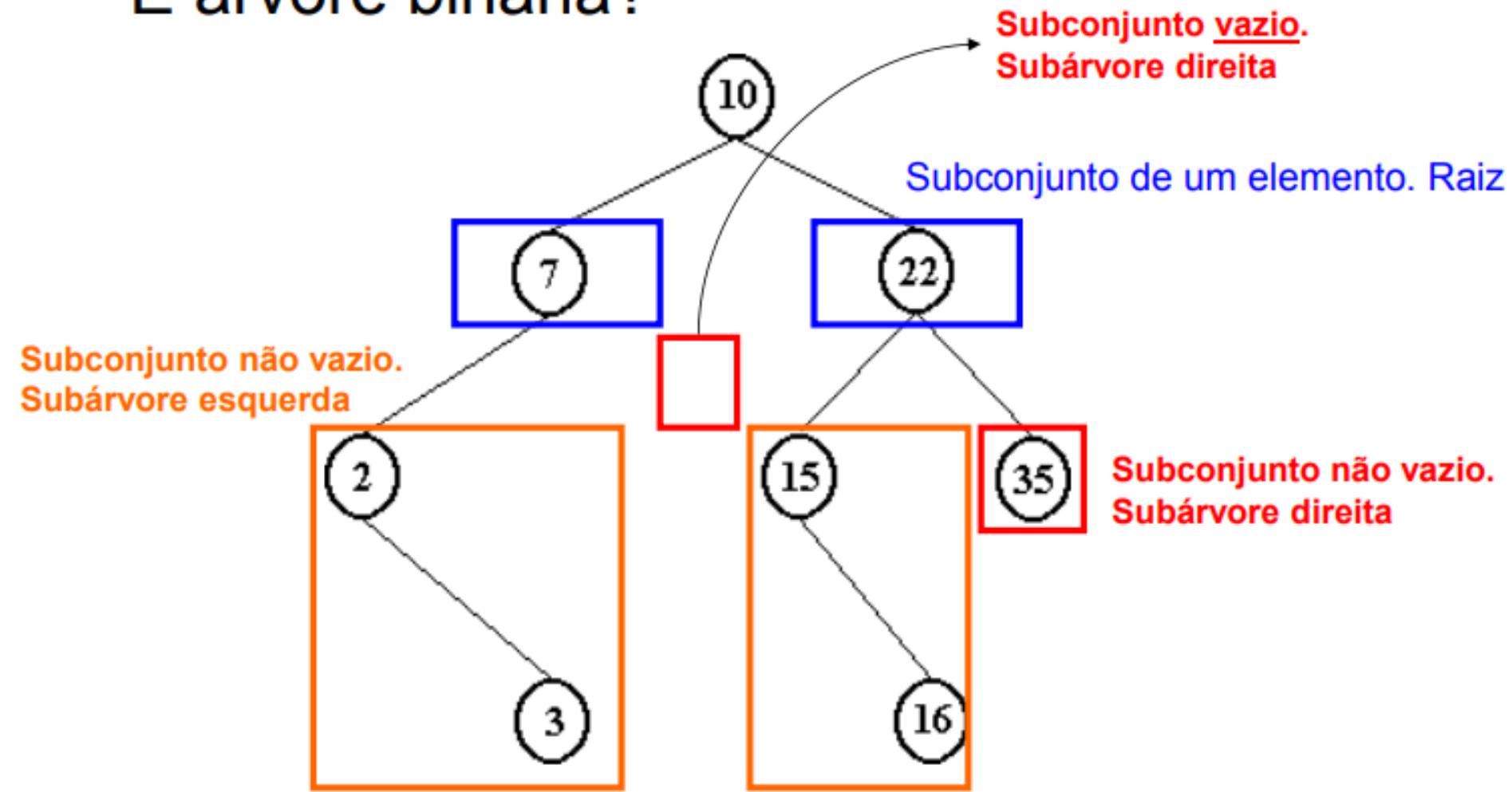
Subárvore esquerda

Subconjunto não vazio.
Subárvore direita



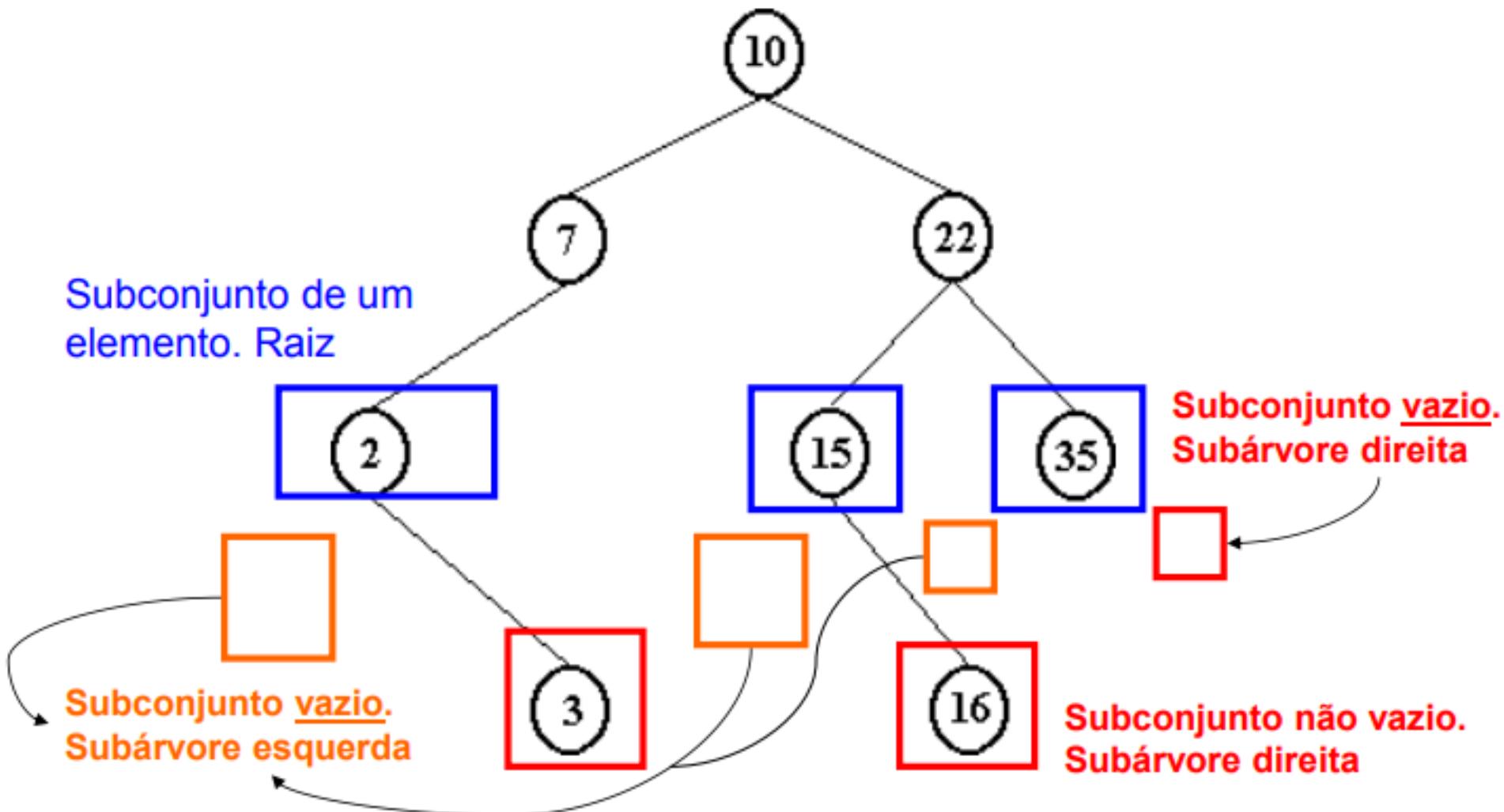
Árvore Binária

- É árvore binária?



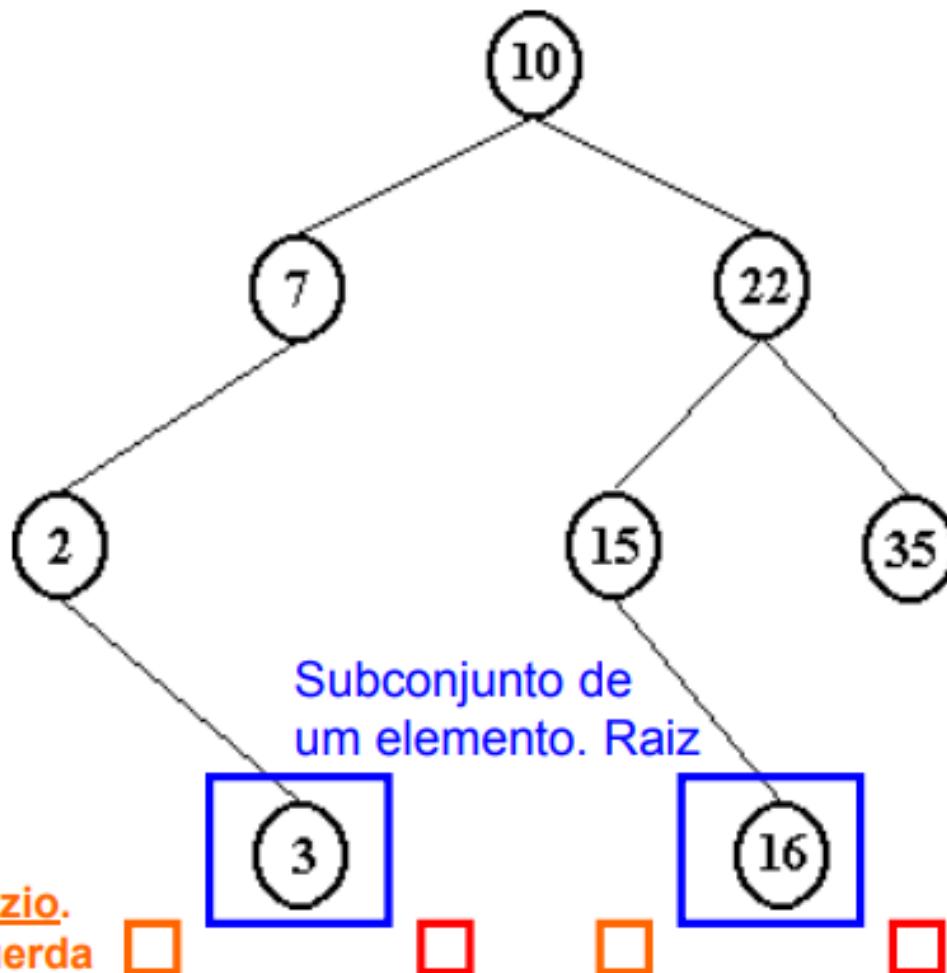
Árvore Binária

- É árvore binária?



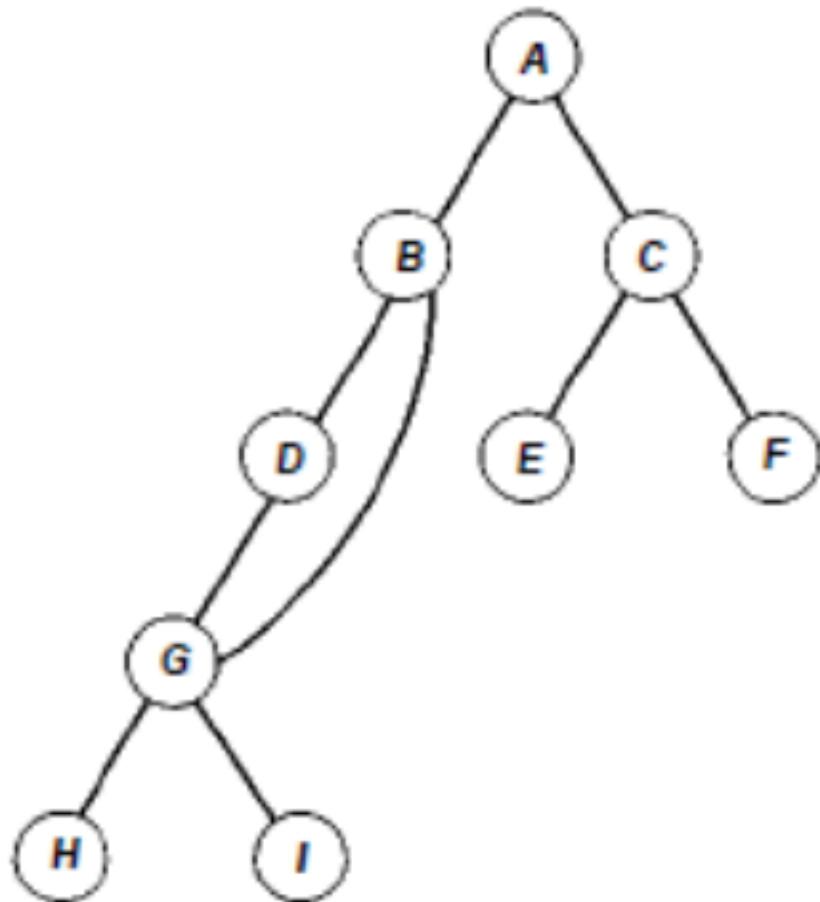
Árvore Binária

- É árvore binária?



Árvore Binária

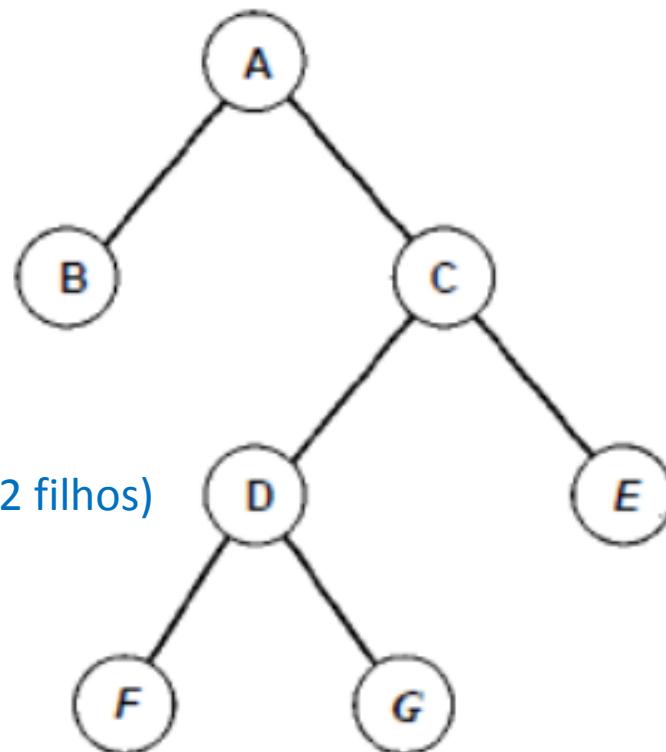
- É árvore binária?



NÃO

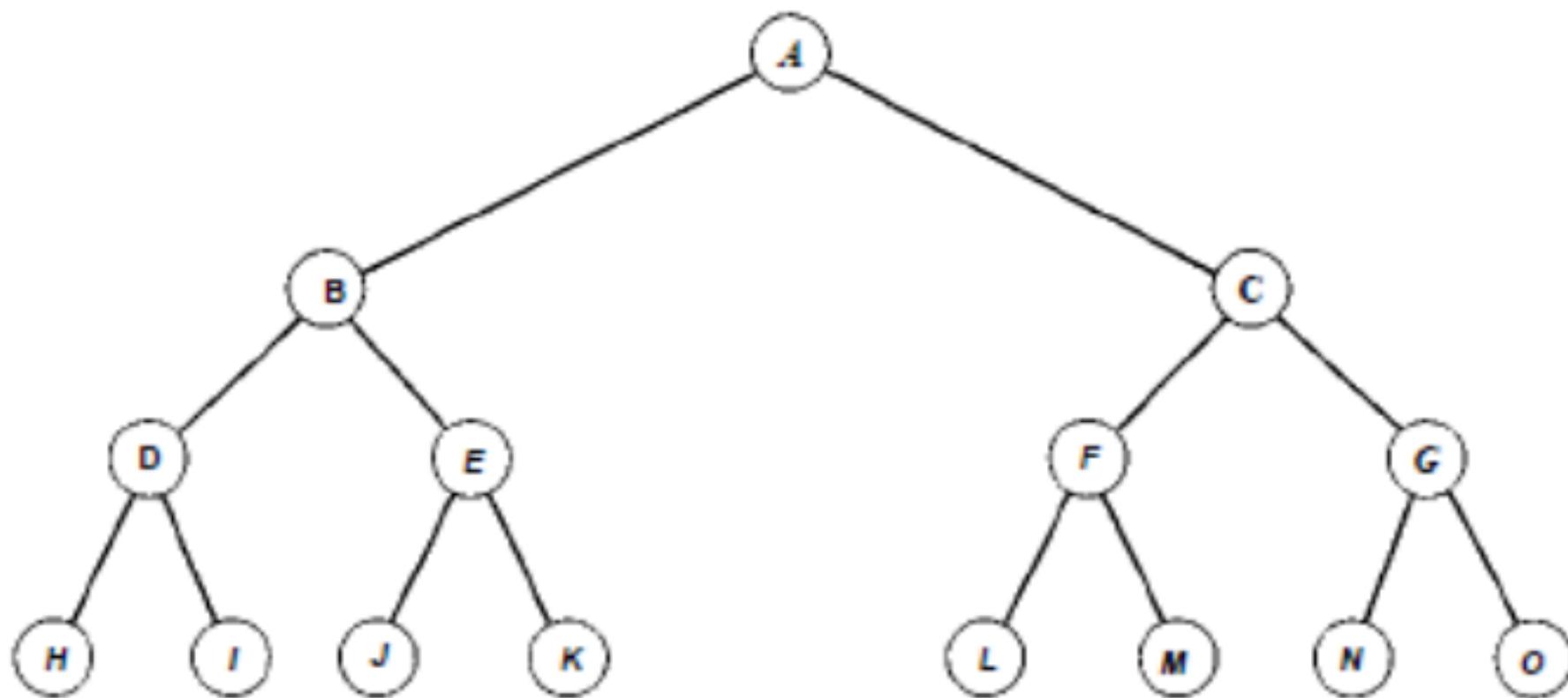
Árvore Binária

- Definição:
 - Árvore Estritamente Binária
 - É toda árvore binária onde todo nó que não é folha em uma árvore binária tiver subárvore esquerda e direita não vazias.



Árvore Binária

- Definição:
 - Árvore Binária Completa de profundidade d
 - É toda árvore estritamente binária onde todos nós folhas estão no nível d.



Árvore Binária

- Definição:
 - Número Total de Nós em uma Árvore Binária Completa dada a profundidade
 - É a soma de todos os nós em cada nível de 0 até h , onde h é a profundidade.

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h$$

Por indução, o número total de nós é dado por:

$$n = 2^{h+1} - 1$$

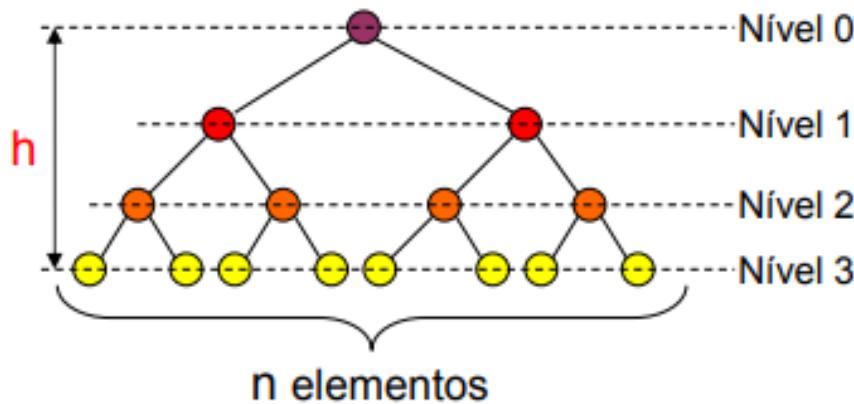
Por se tratar de uma árvore binária completa de profundidade h . Pode-se afirmar que todas as folhas estão no nível h . E tem-se um total de 2^h folhas. Deste modo temos exatamente:

$$ni = (2^{h+1}-1) - 2^h = 2^h - 1 \quad \text{Nós internos nas árvore}$$

Árvore Binária

- Definição:
 - Altura de uma Árvore Binária, dada a quantidade de nós

$$n = 2^{h+1} - 1$$

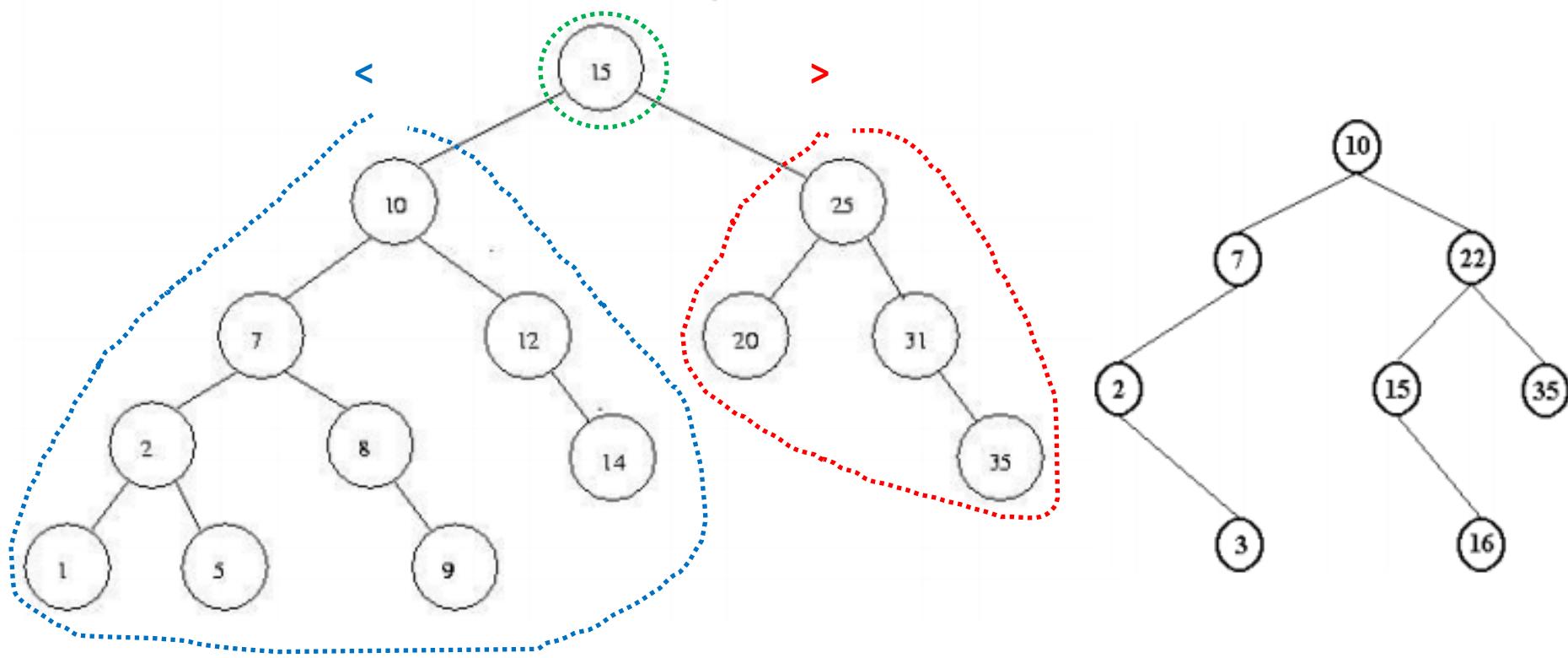


$$\log_2 n = h$$

n	$\log_2 n$
15	3,91
9	3,17
8	3
7	2,81
4	2
2	1

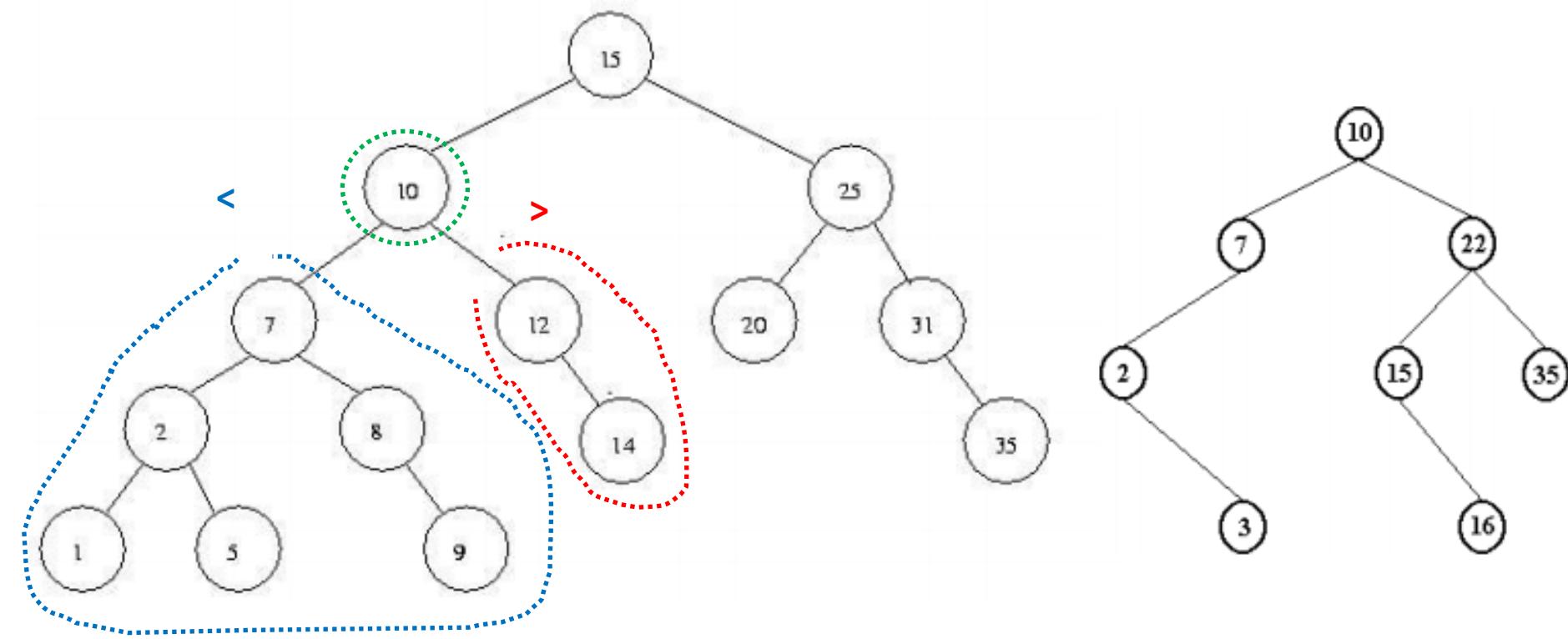
Árvore Binária de Busca

- Definição:
 - Árvore Binária de Busca
 - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.



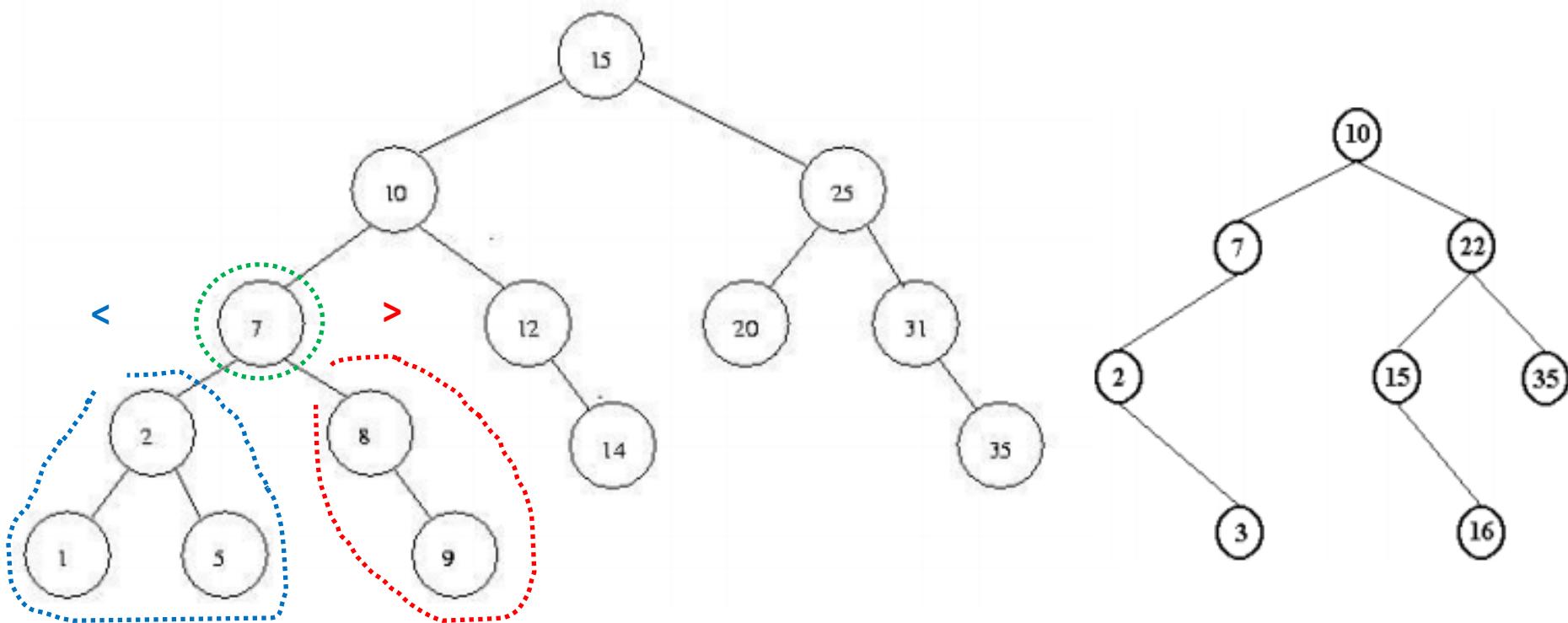
Árvore Binária de Busca

- Definição:
 - Árvore Binária de Busca
 - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.



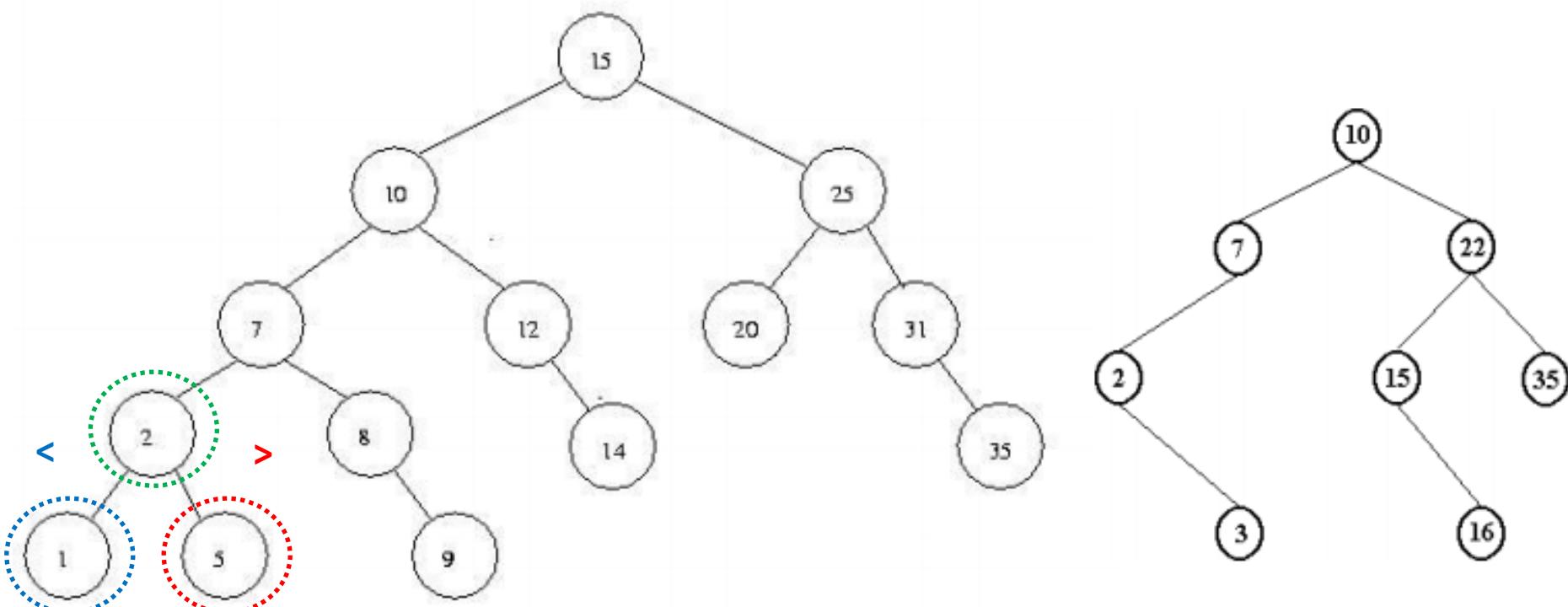
Árvore Binária de Busca

- Definição:
 - Árvore Binária de Busca
 - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.



Árvore Binária de Busca

- Definição:
 - Árvore Binária de Busca
 - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k , e todos os nós da subárvore direita de k são maiores que k . E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.

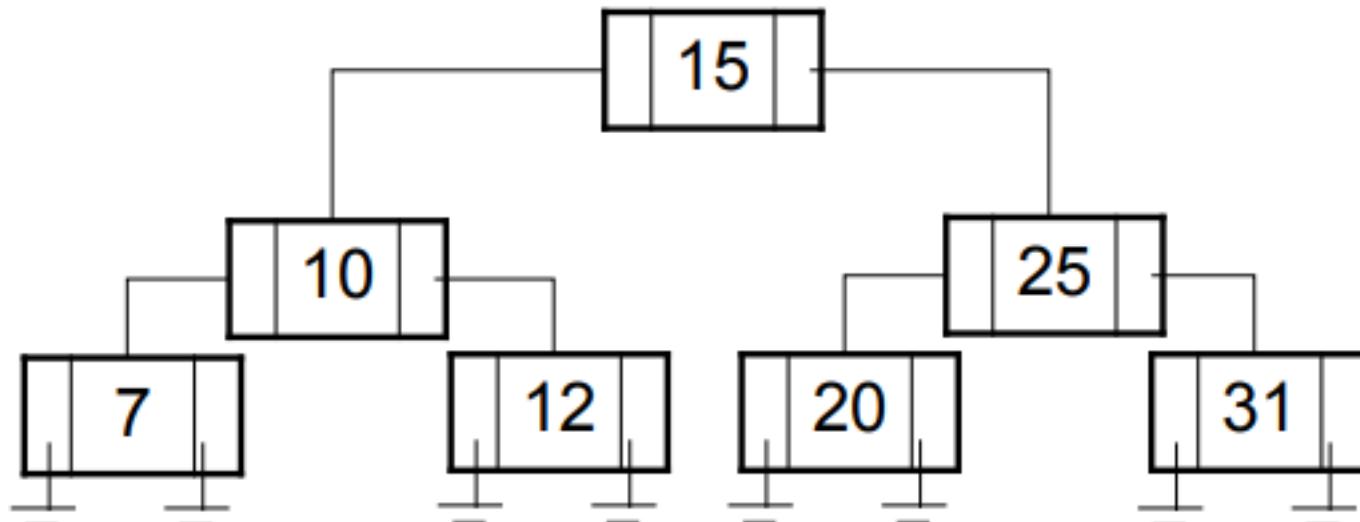


Árvore Binária de Busca

- Exemplo de Árvore Binária de Busca Estática:

0	1	2	3	4	5	6
15	10	25	7	12	20	31

- Exemplo de Árvore Binária de Busca Dinâmica:



Árvore Binária de Busca

- Em Ciência da Computação uma árvore binária de busca dinâmica é uma estrutura de dados que:
 - Consiste de uma seqüencia de registros
 - Cada registro tem quatro campos:
 - uma campo para a chave.
 - um ponteiro para o filho da esquerda (subárvore da esquerda)
 - um ponteiro para o filho da direita (subárvore da direita)
 - um ponteiro para o pai. (nó pai)

Ponteiro para raiz
Quantidade

Árvore

pai		
esquerda	chave	direita

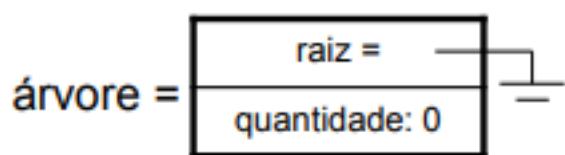
Nó da árvore

Árvore Binária de Busca

- Operações Básicas
 - Inserção de nó na Árvore.
 - Eliminação de nó da Árvore.
 - Percorrer nós da Árvore.

Árvore Binária de Busca

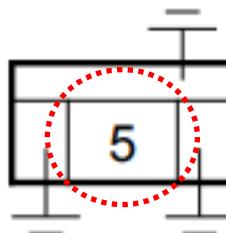
- Funcionamento (Insere)



A principio a árvore está vazia: raiz aponta para NULL.

Árvore Binária de Busca

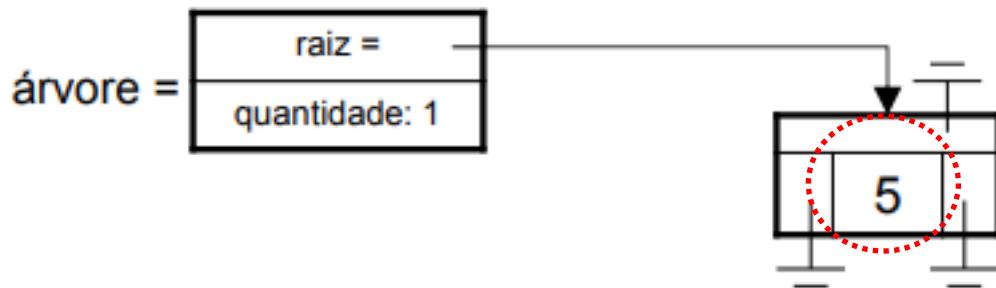
- Funcionamento (Insere)



- • Inserir o elemento: 5
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Caso a árvore esteja vazia. (campo raiz da árvore igual a NULL)
- • Faça árvore no campo raiz apontar para o novo item

Árvore Binária de Busca

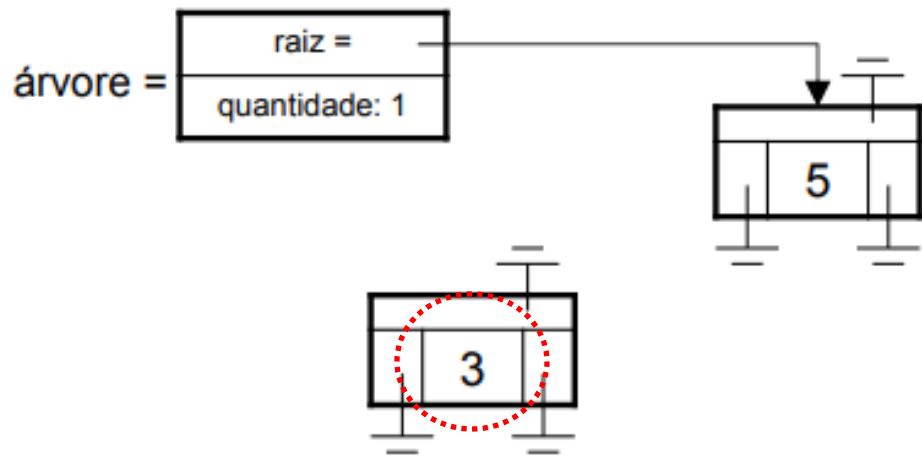
- Funcionamento (Insere)



- • Inserir o elemento: 5
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Caso a árvore esteja vazia. (campo raiz da árvore igual a NULL)
- • Faça árvore no campo raiz apontar para o novo item

Árvore Binária de Busca

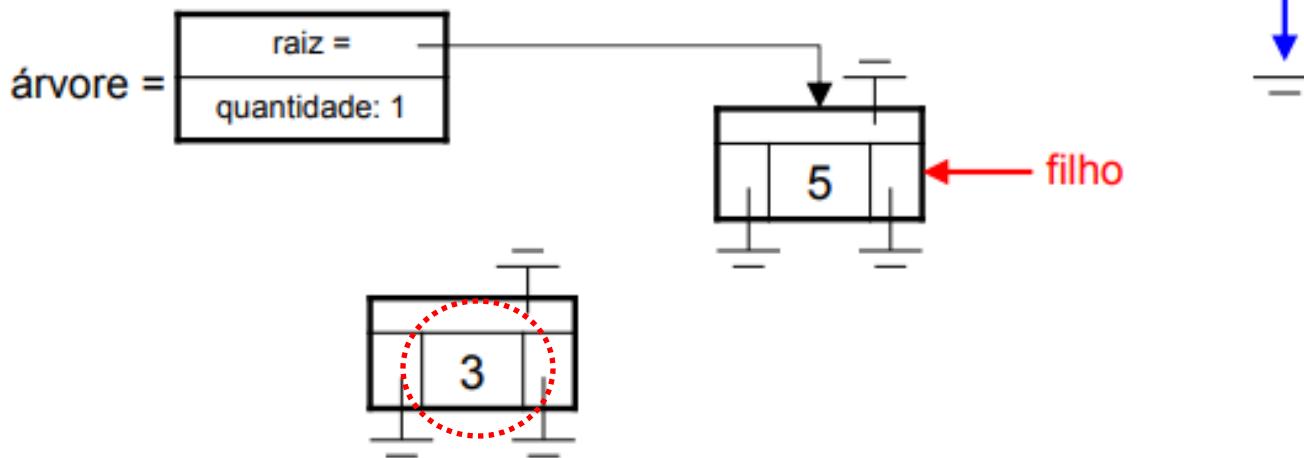
- Funcionamento (Insere)



- Inserir o elemento: 3
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

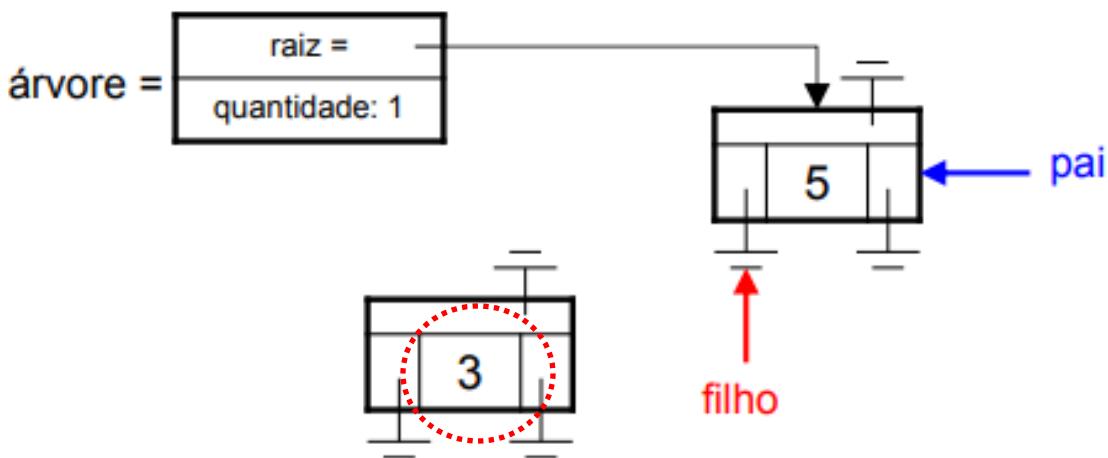
- Funcionamento (Insere)



- • Inserir o elemento: 3
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
 - Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - Faça o ponteiro pai do novo item apontar para o pai.
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

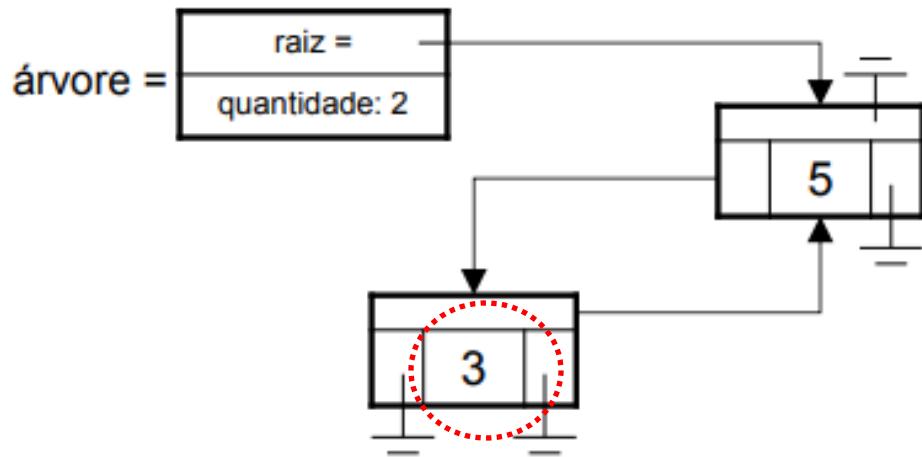
- Funcionamento (Insere)



- Inserir o elemento: 3
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- • Se a chave do pai for maior que a chave do novo item.
 - • Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

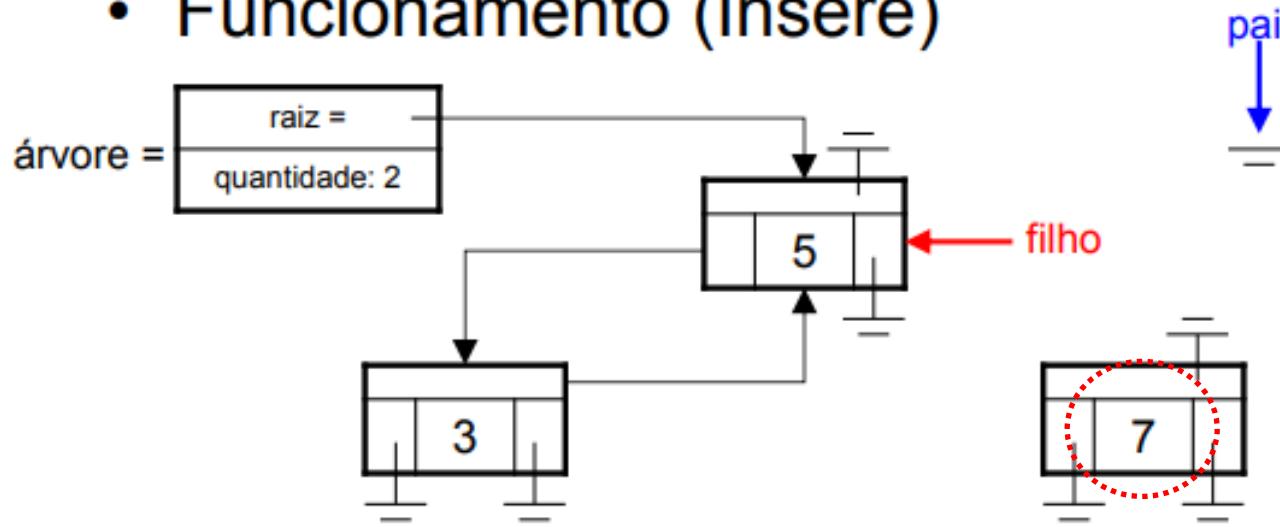
- Funcionamento (Insere)



- Inserir o elemento: 3
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- • Se a chave do pai for maior que a chave do novo item.
 - • Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- • Faça o ponteiro pai do novo item apontar para o pai.
- • Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

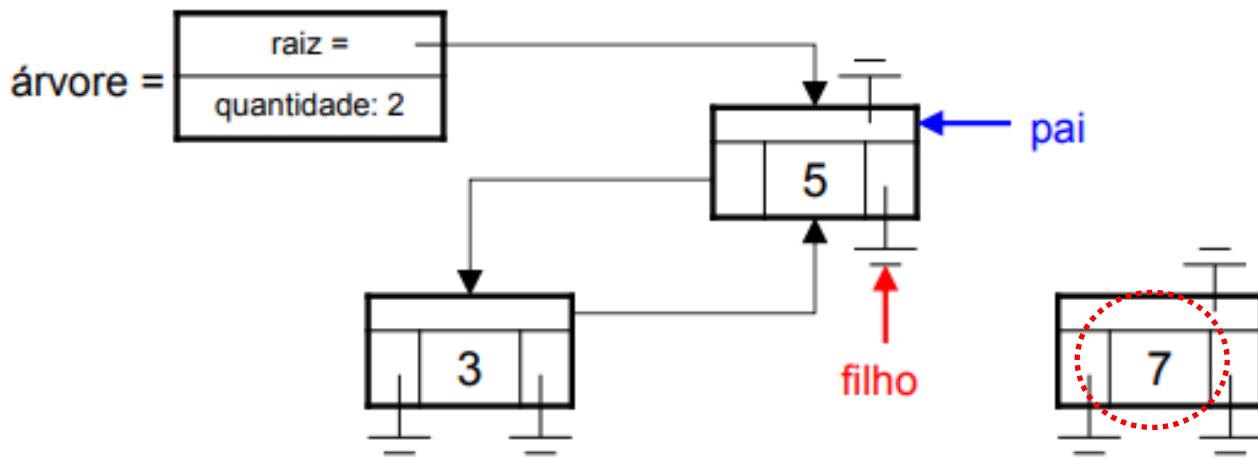
- Funcionamento (Insere)



- • Inserir o elemento: 7
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

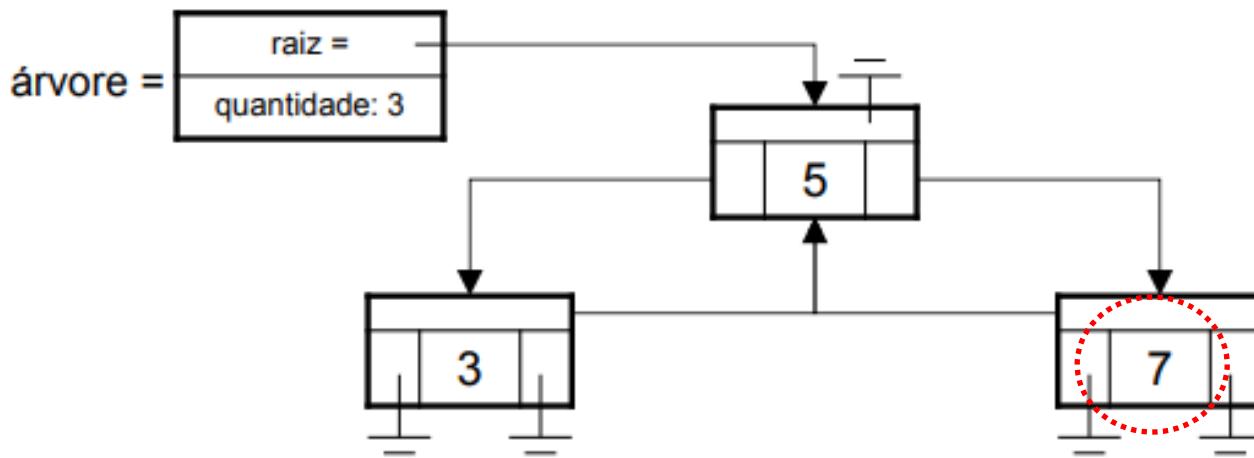
- Funcionamento (Insere)



- Inserir o elemento: 7
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- • Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - • Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - Faça o ponteiro pai do novo item apontar para o pai.
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

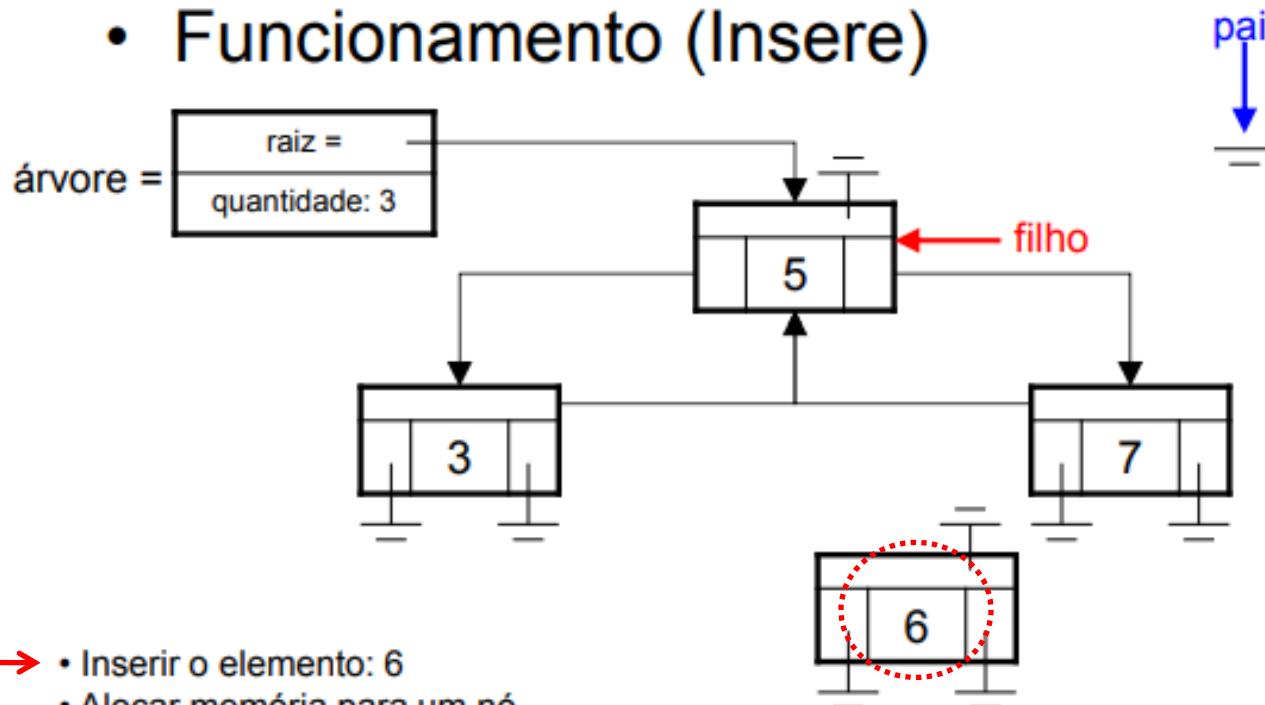
- Funcionamento (Insere)



- Inserir o elemento: 7
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- • Se a chave do pai for menor que a chave do novo item
 - • Faça ponteiro direita do pai apontar para o novo item
- • Faça o ponteiro pai do novo item apontar para o pai.
- • Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

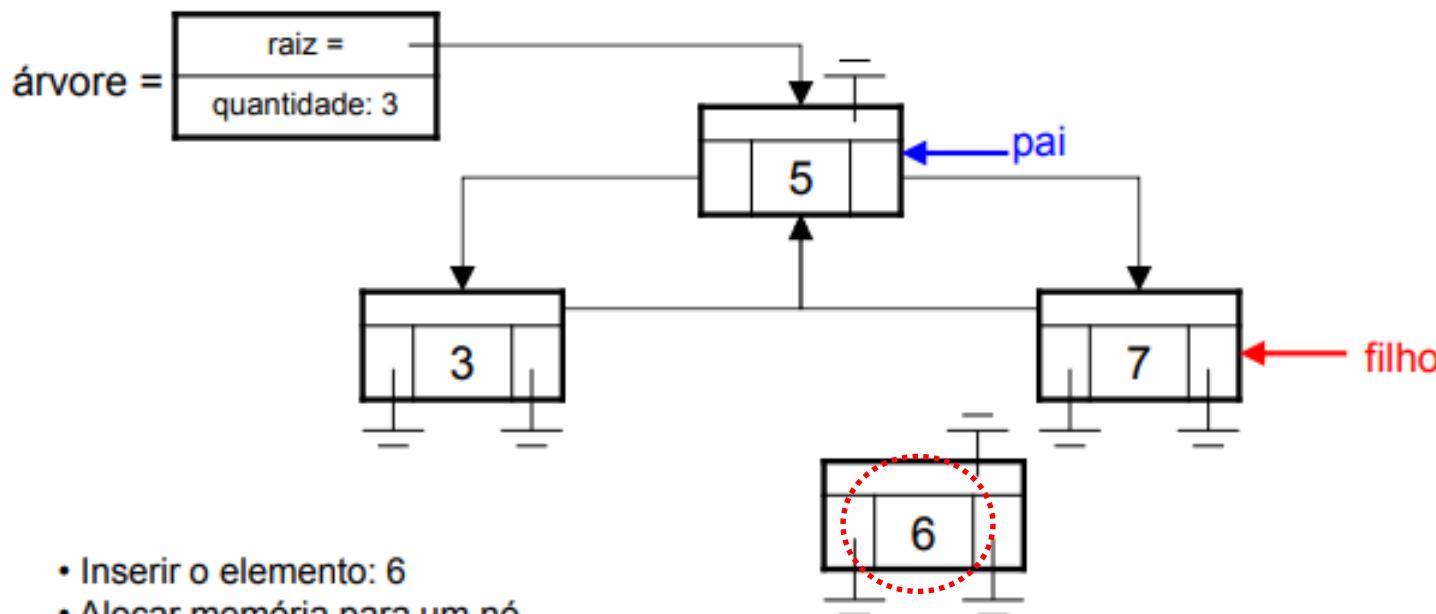
- Funcionamento (Insere)



- • Inserir o elemento: 6
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

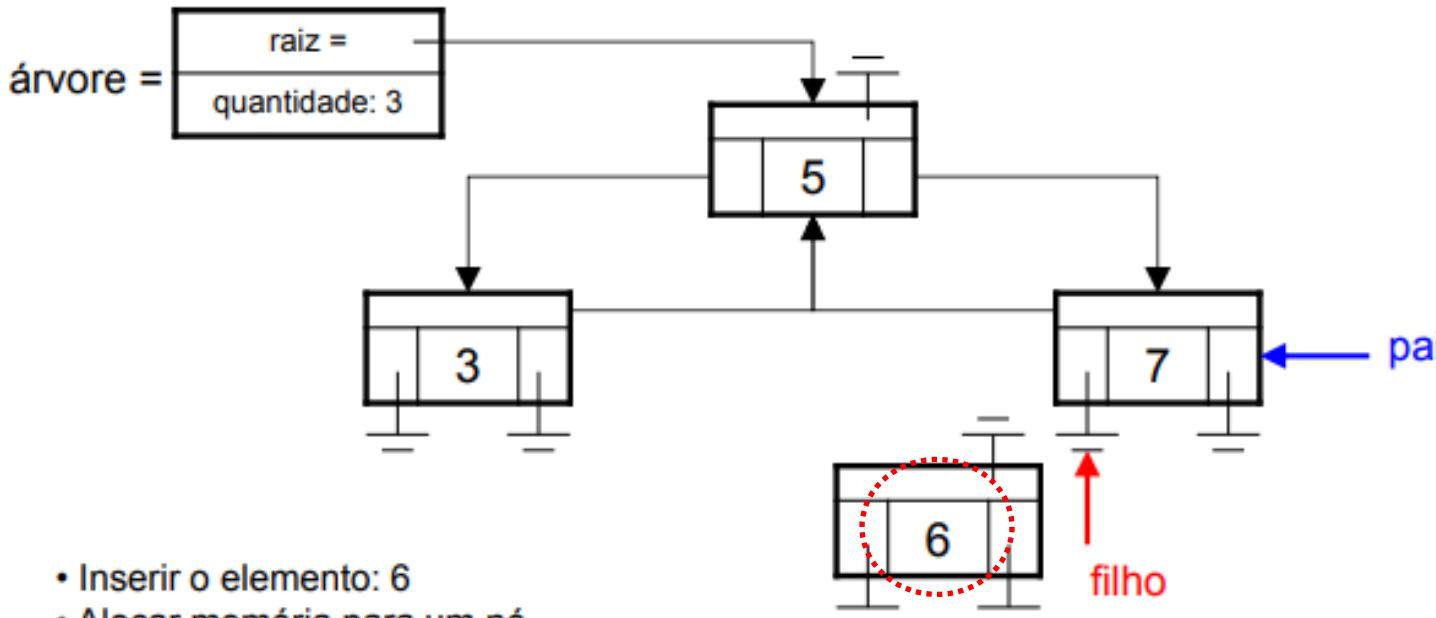
- Funcionamento (Insere)



- Inserir o elemento: 6
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
 - Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - Faça o ponteiro pai do novo item apontar para o pai.
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

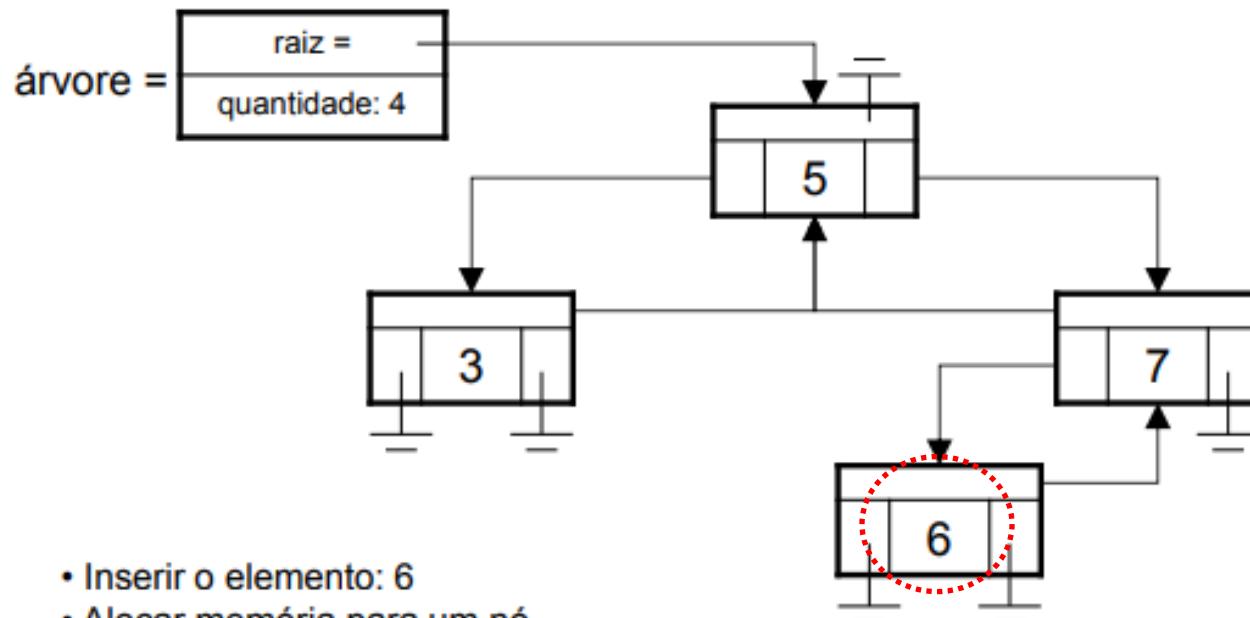
- Funcionamento (Insere)



- Inserir o elemento: 6
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
 - Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - Faça o ponteiro pai do novo item apontar para o pai.
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

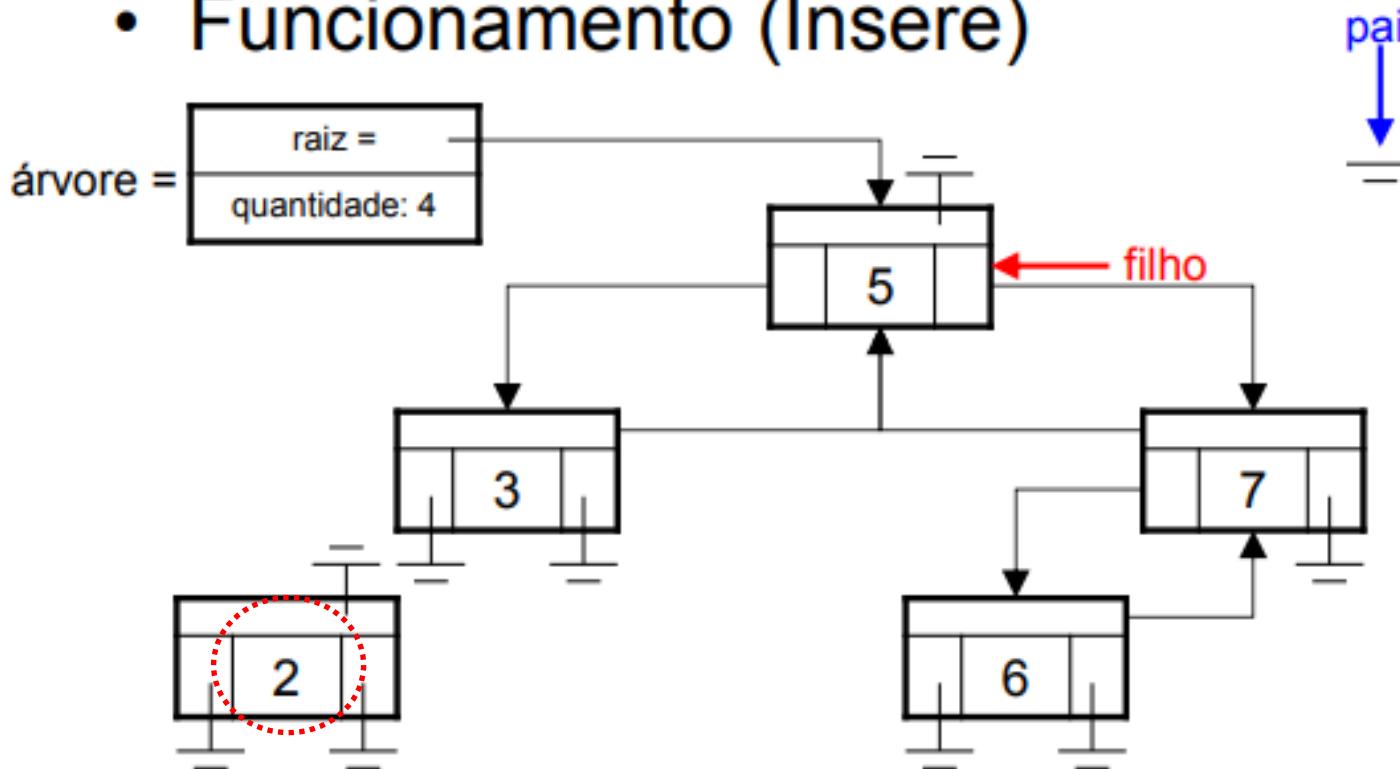
- Funcionamento (Insere)



- Inserir o elemento: 6
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
 - Ao encontrar o lugar.
 - • Se a chave do pai for maior que a chave do novo item.
 - • Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - • Faça o ponteiro pai do novo item apontar para o pai.
 - • Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

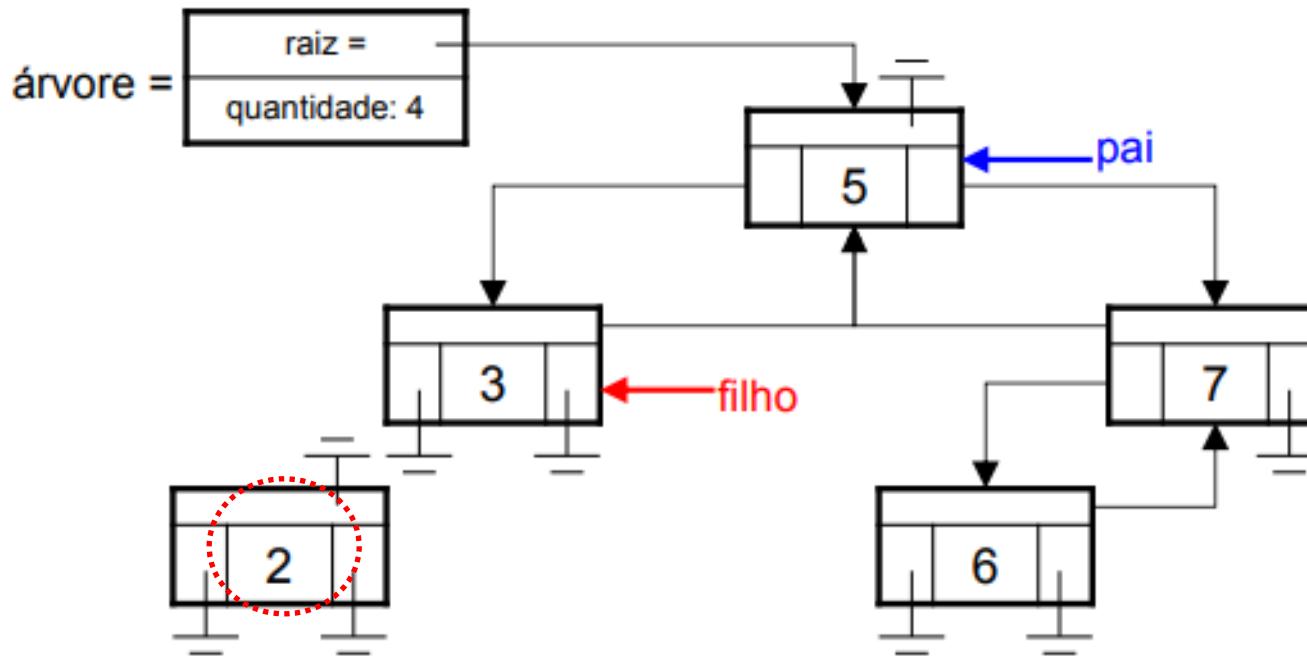
- Funcionamento (Insere)



- • Inserir o elemento: 2
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
 - Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
 - Faça o ponteiro pai do novo item apontar para o pai.
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

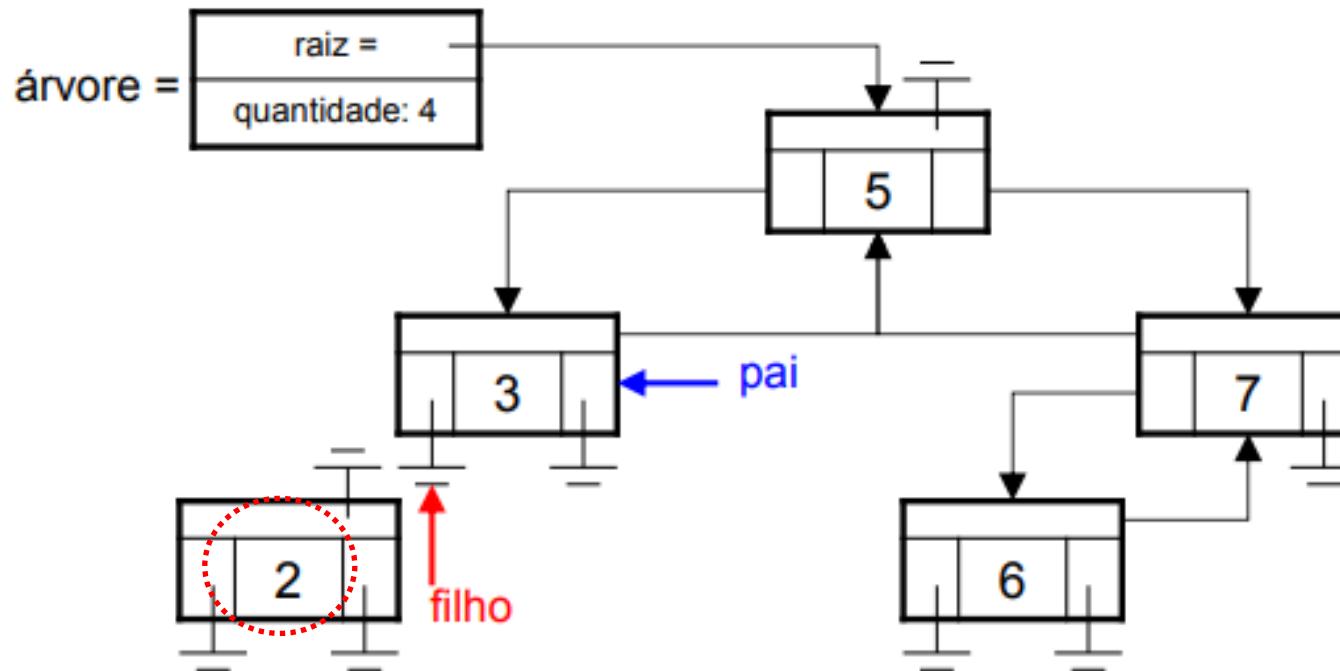
- Funcionamento (Insere)



- Inserir o elemento: 2
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

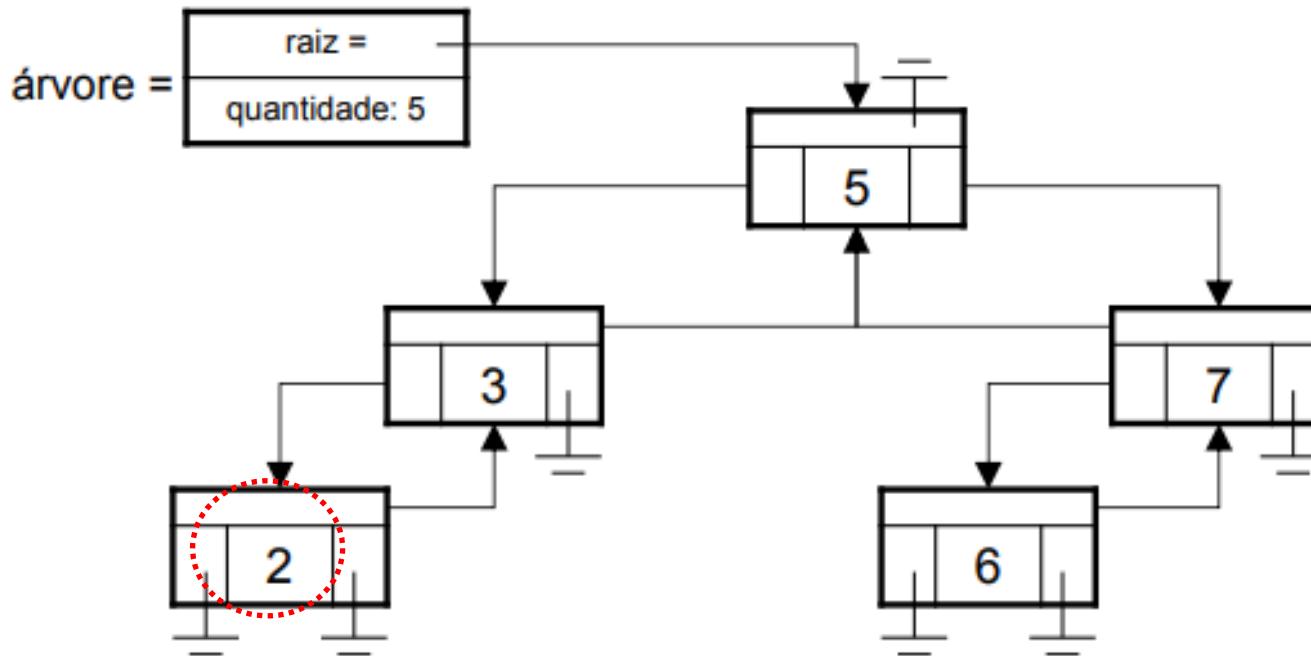
- Funcionamento (Insere)



- Inserir o elemento: 2
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- • Se a chave do pai for maior que a chave do novo item.
 - • Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

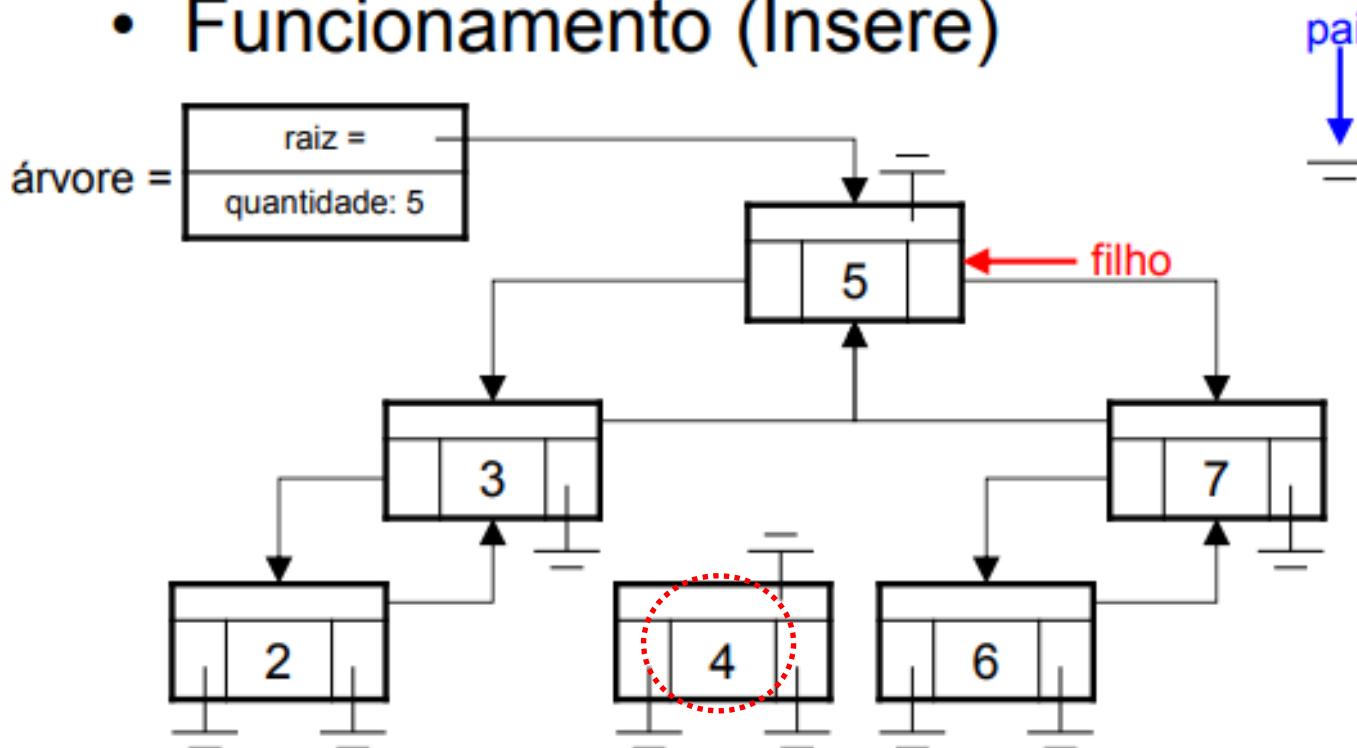
- Funcionamento (Insere)



- Inserir o elemento: 2
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
 - Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
 - Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incrementa o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

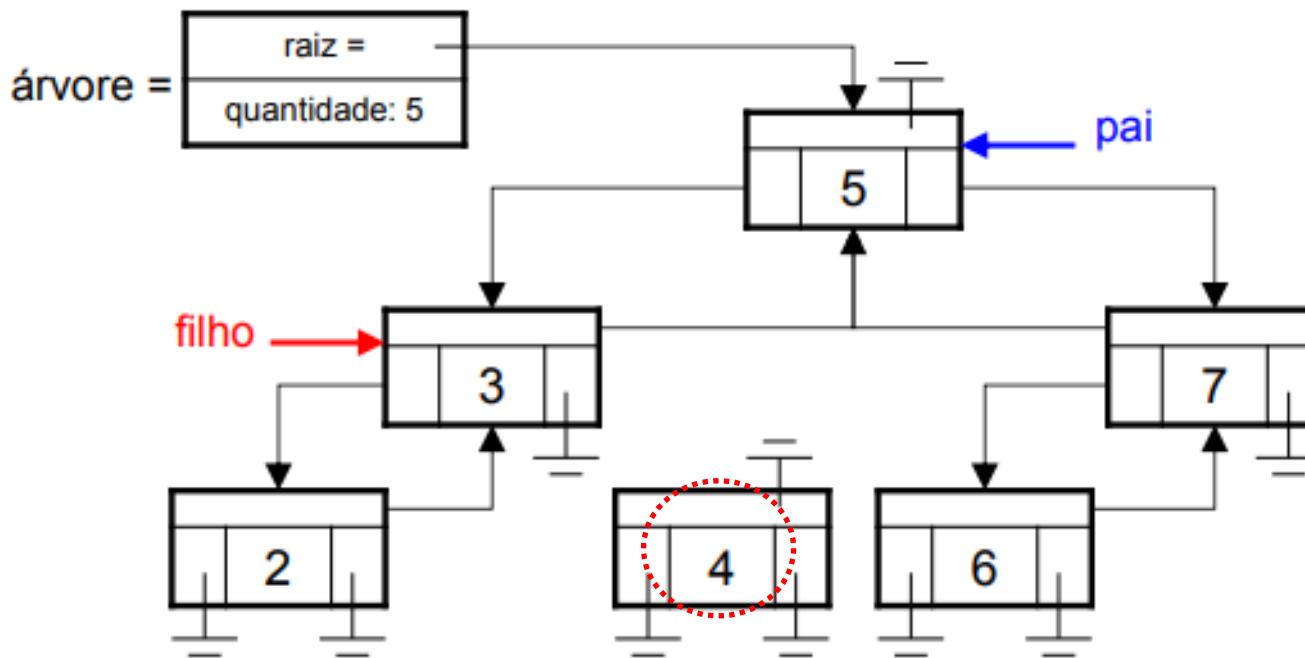
- Funcionamento (Insere)



- • Inserir o elemento: 4
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

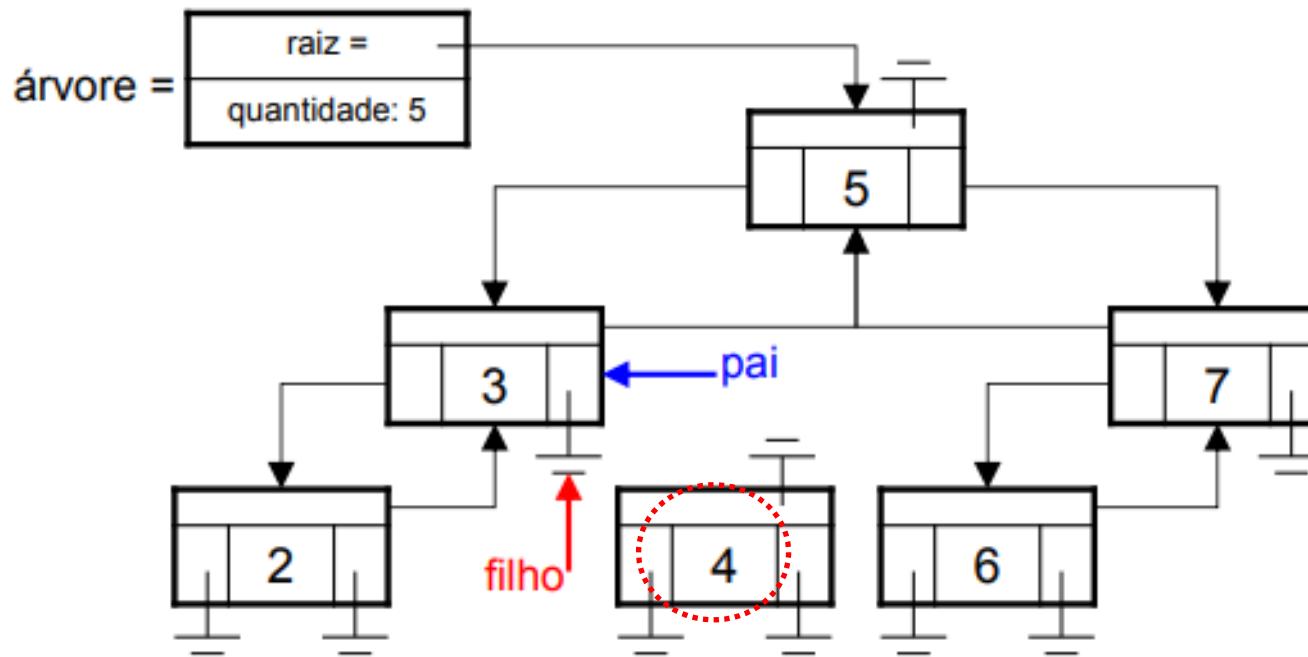
- Funcionamento (Insere)



- Inserir o elemento: 4
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

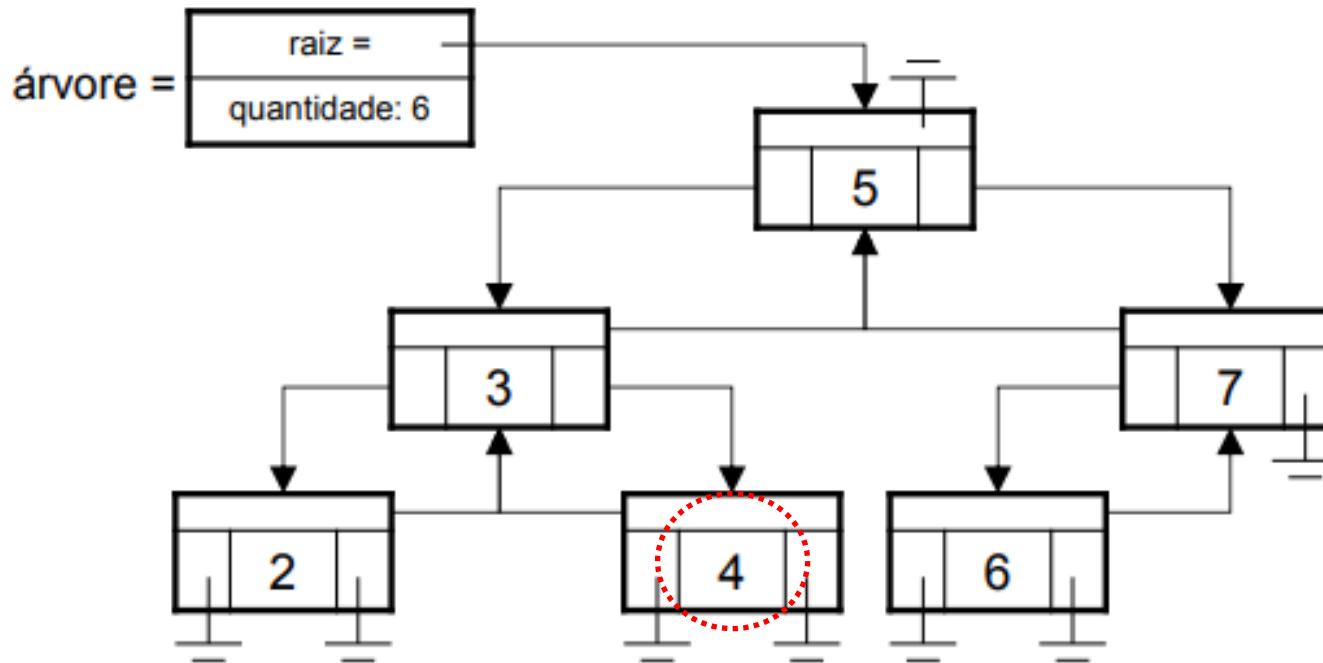
- Funcionamento (Insere)



- Inserir o elemento: 4
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- • Percorra a árvore buscando o lugar para inserir o item novo. Guaarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- • Se a chave do pai for menor que a chave do novo item
 - • Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

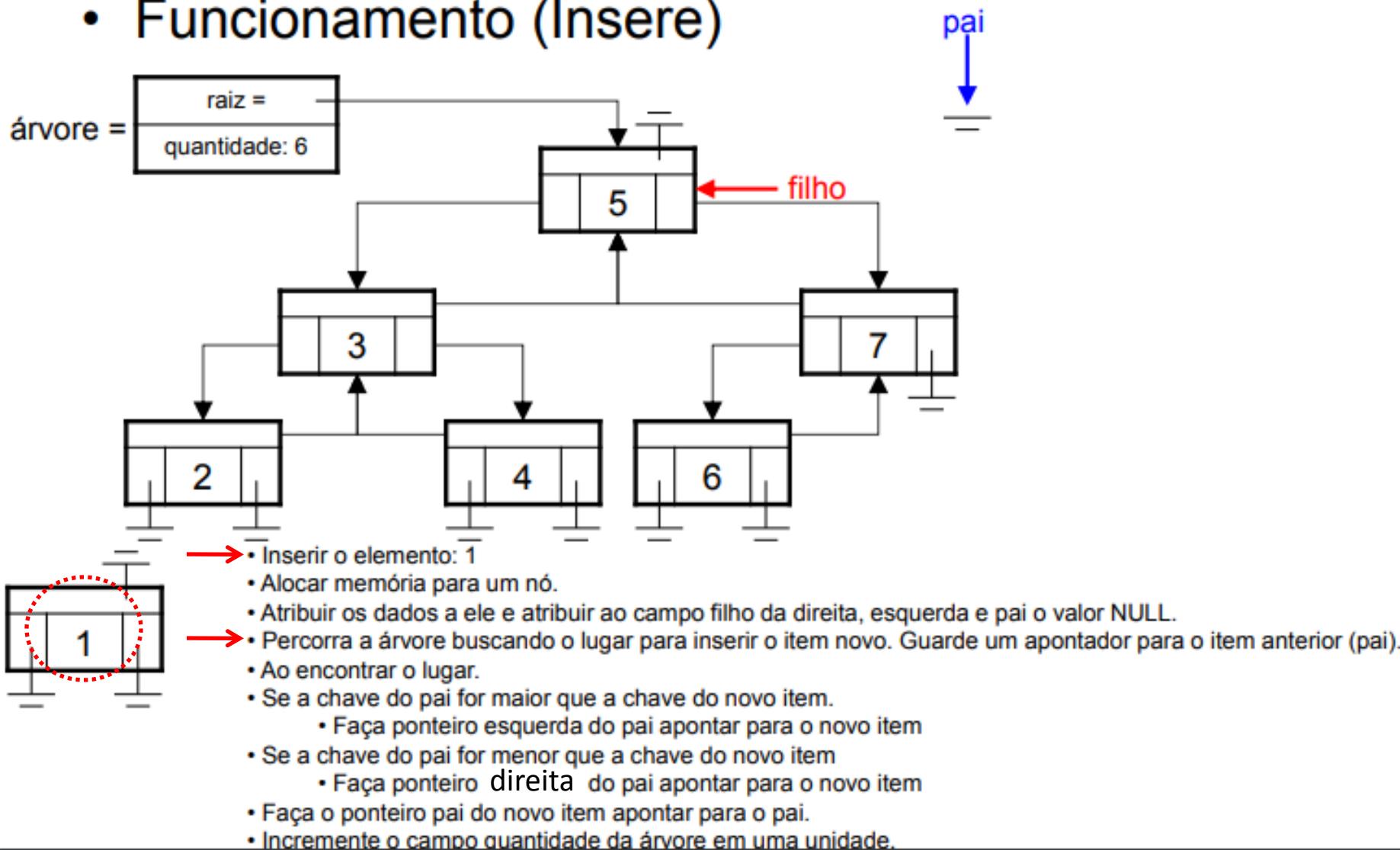
- Funcionamento (Insere)



- Inserir o elemento: 4
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - • Faça ponteiro direita do pai apontar para o novo item
- • Faça o ponteiro pai do novo item apontar para o pai.
- • Incremente o campo quantidade da árvore em uma unidade.

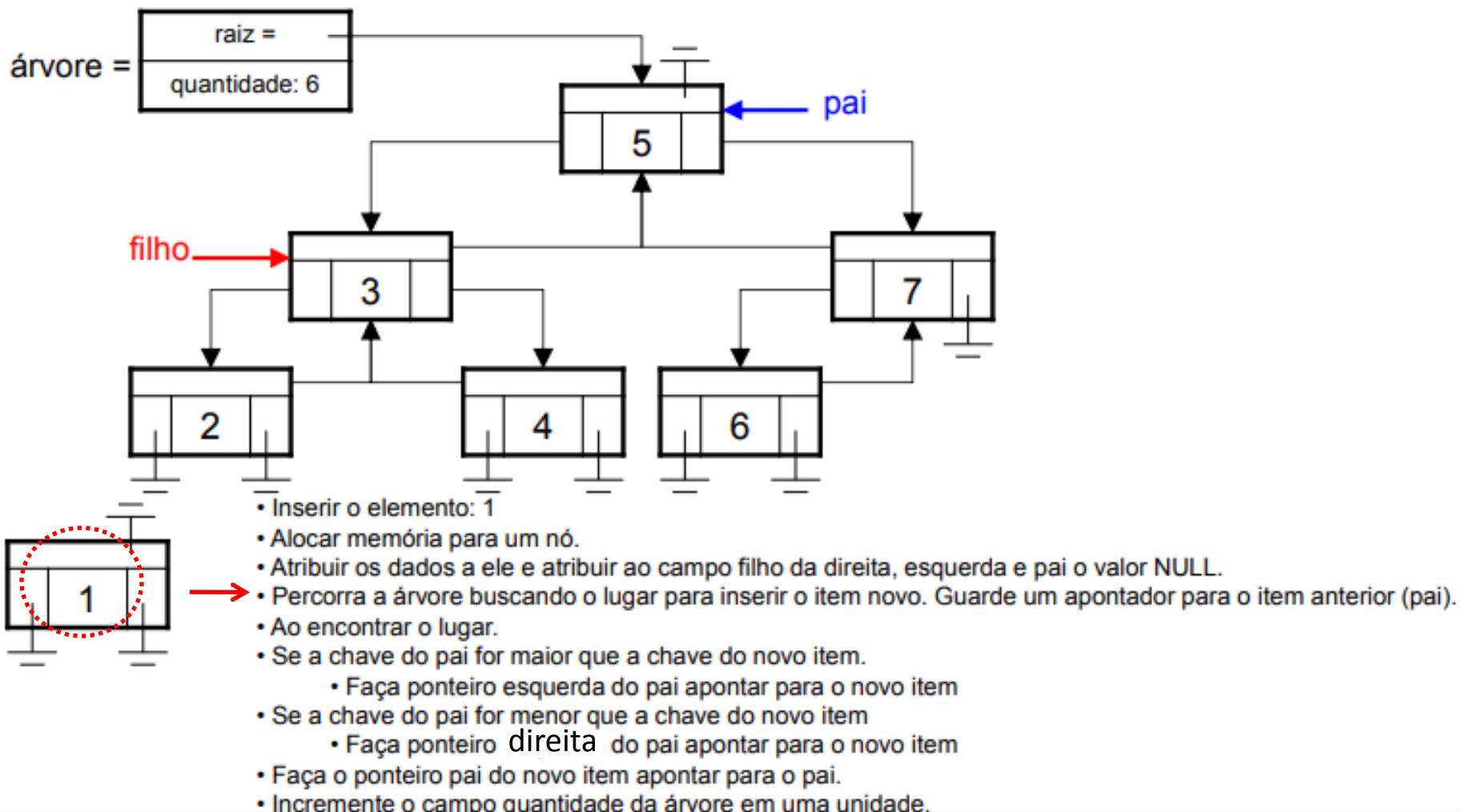
Árvore Binária de Busca

- Funcionamento (Insere)



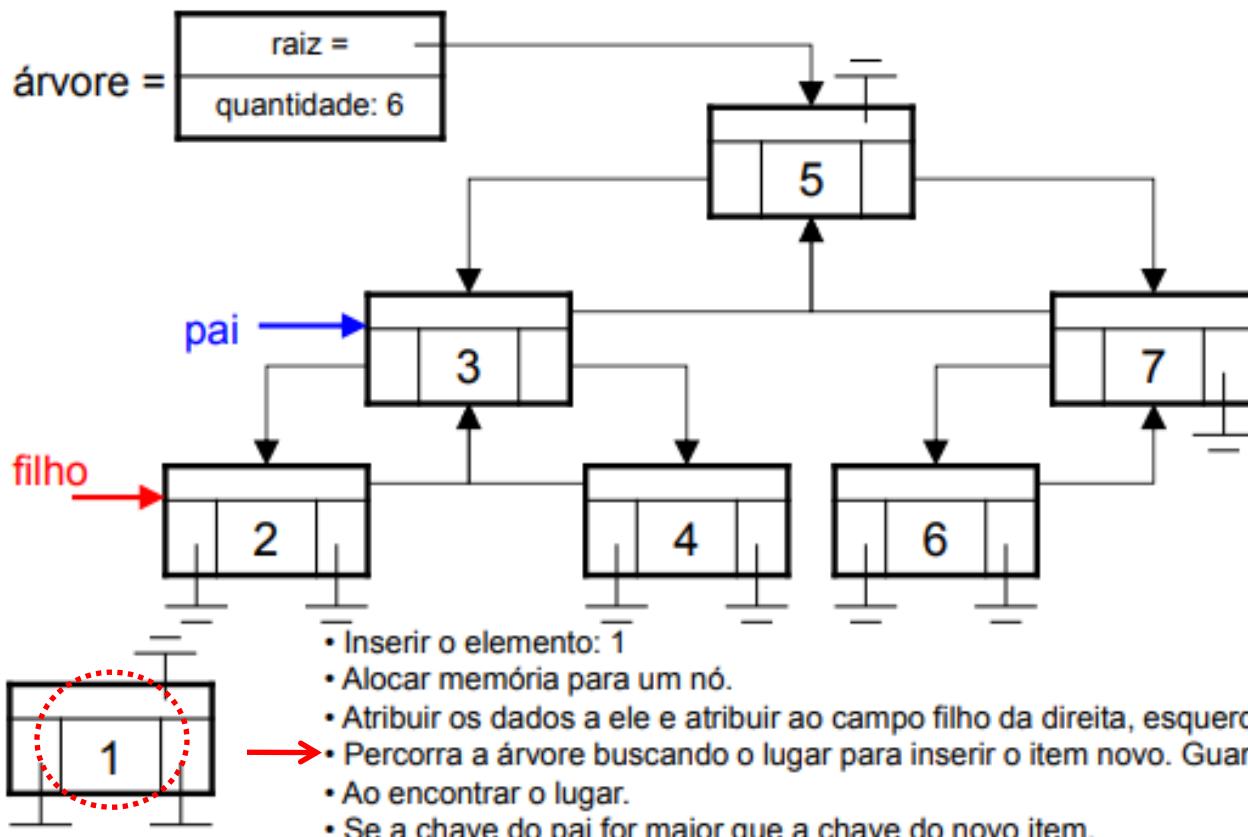
Árvore Binária de Busca

- Funcionamento (Insere)



Árvore Binária de Busca

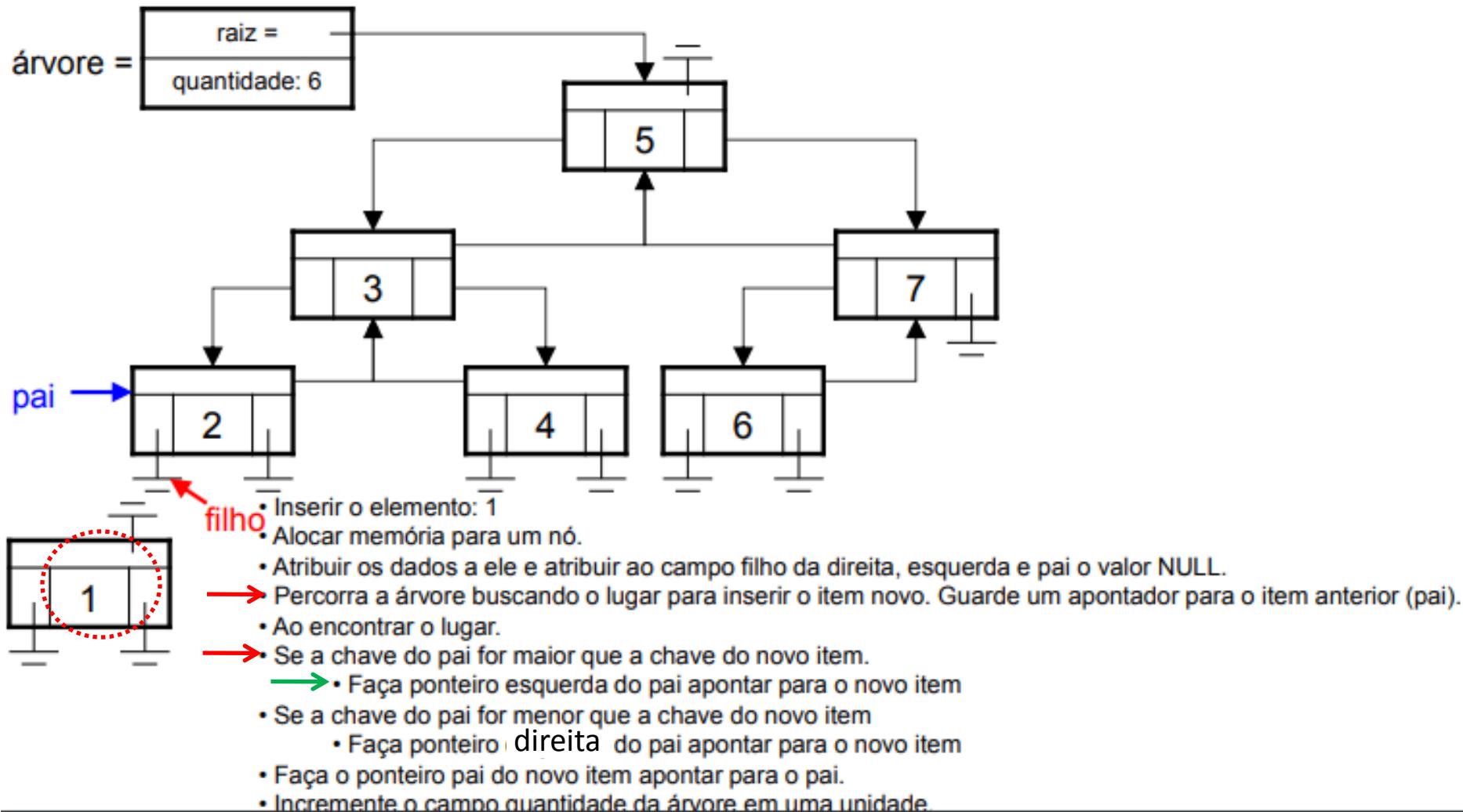
- Funcionamento (Insere)



- Inserir o elemento: 1
- Alocar memória para um nó.
- Atribuir os dados a ele e atribuir ao campo filho da direita, esquerda e pai o valor NULL.
- Percorra a árvore buscando o lugar para inserir o item novo. Guarde um apontador para o item anterior (pai).
- Ao encontrar o lugar.
- Se a chave do pai for maior que a chave do novo item.
 - Faça ponteiro esquerda do pai apontar para o novo item
- Se a chave do pai for menor que a chave do novo item
 - Faça ponteiro direita do pai apontar para o novo item
- Faça o ponteiro pai do novo item apontar para o pai.
- Incremente o campo quantidade da árvore em uma unidade.

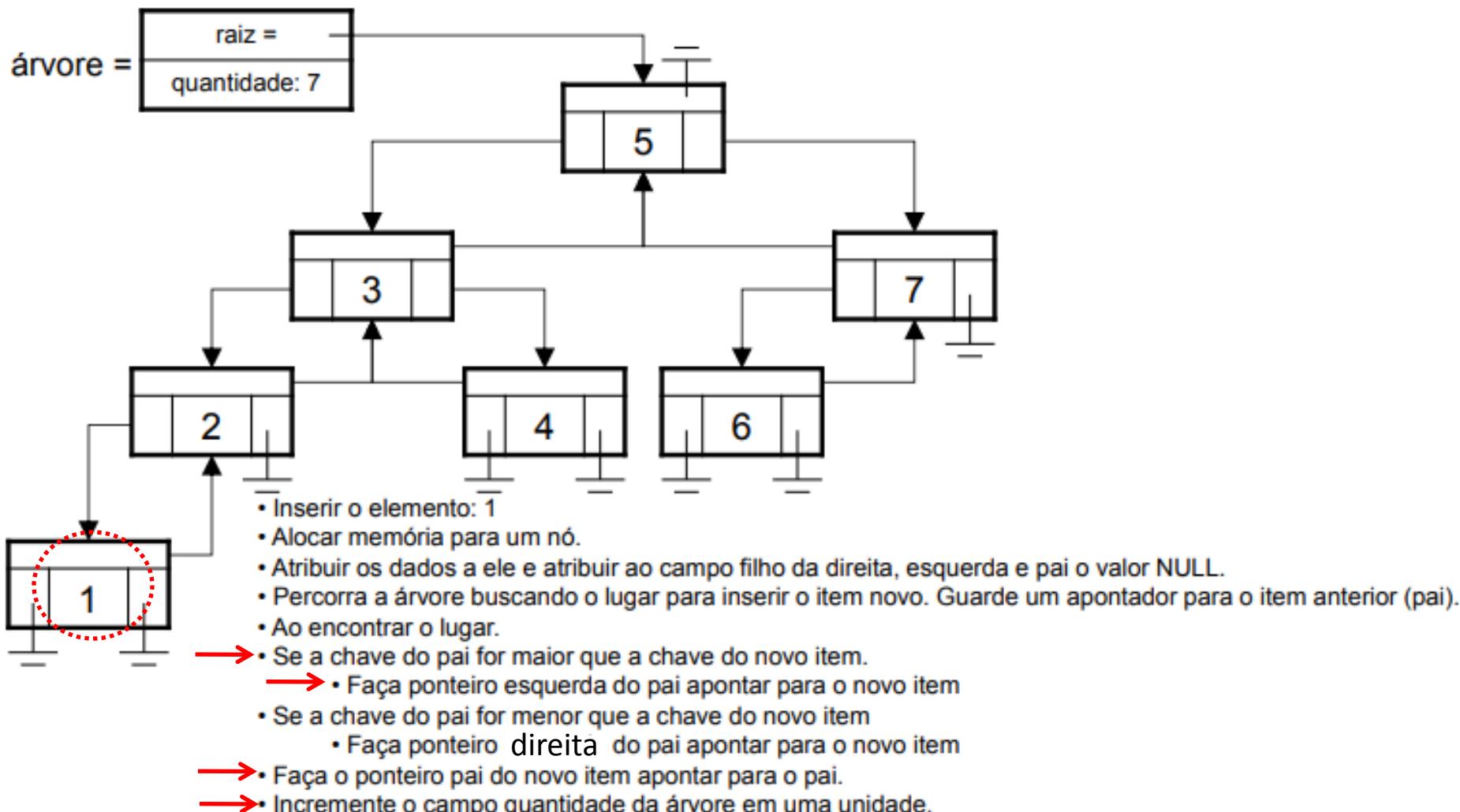
Árvore Binária de Busca

- Funcionamento (Insere)



Árvore Binária de Busca

- Funcionamento (Insere)



Árvore Binária de Busca

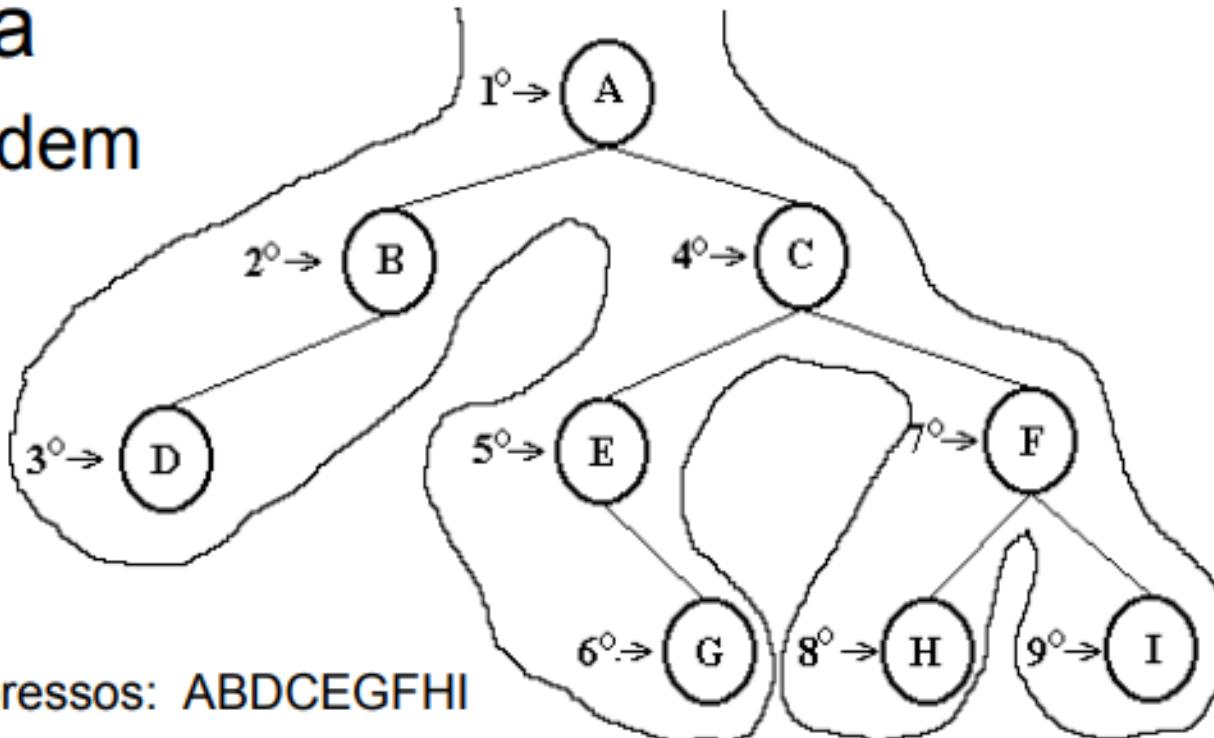
- Inserir (Lembretes)
 - Caso Particular: (Primeiro item)
 - Se a árvore estiver vazia.
 - Aloque espaço para o item.
 - Atribua-lhe os dados.
 - Faça com que o campo raiz da árvore aponte para o novo item.
 - Caso Geral:
 - Se houver itens na árvore.
 - Percorrer a partir da raiz todos os itens checando:
 - » Se campo chave do item for maior que a chave do novo item
 - » Vá para o filho esquerdo do item
 - » Se campo chave do item for menor que a chave do novo item
 - » Vá para o filho direito do item
 - Até encontrar um item nulo.
 - Ao encontrar
 - » Se campo chave do pai for maior que a chave do novo item
 - » Faça o campo esquerda do pai apontar para o novo item
 - » Se campo chave do pai for menor que a chave do novo item
 - » Faça o campo direita do pai apontar para o novo item
 - » Faça campo pai do novo item apontar para o pai
 - Incremente o campo quantidade da árvore em uma unidade.

Árvore Binária de Busca

- Pesquisa
 - Há 3 formas de se efetuar a pesquisa em uma árvore:
 - Pré-Ordem
 - In-Ordem
 - Pós-Ordem

Árvore Binária de Busca

- Pesquisa
 - Pré-Ordem

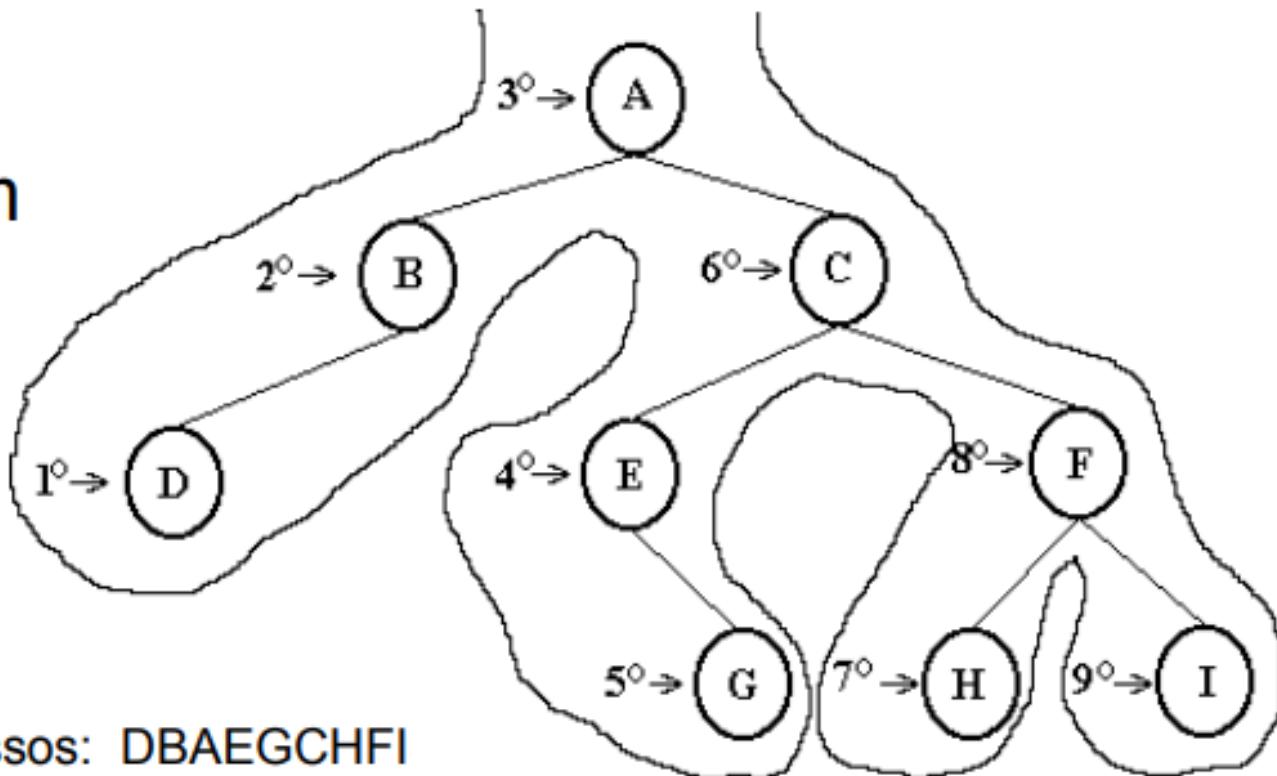


Ordem dos nós impressos: ABDCEGFHI

- Se a árvore estiver vazia, então fim
 - Visite a Raiz.
 - Percorra em pré-ordem a subárvore da esquerda.
 - Percorra em pré-ordem a subárvore da direita.

Árvore Binária de Busca

- Pesquisa
 - + In-Ordem

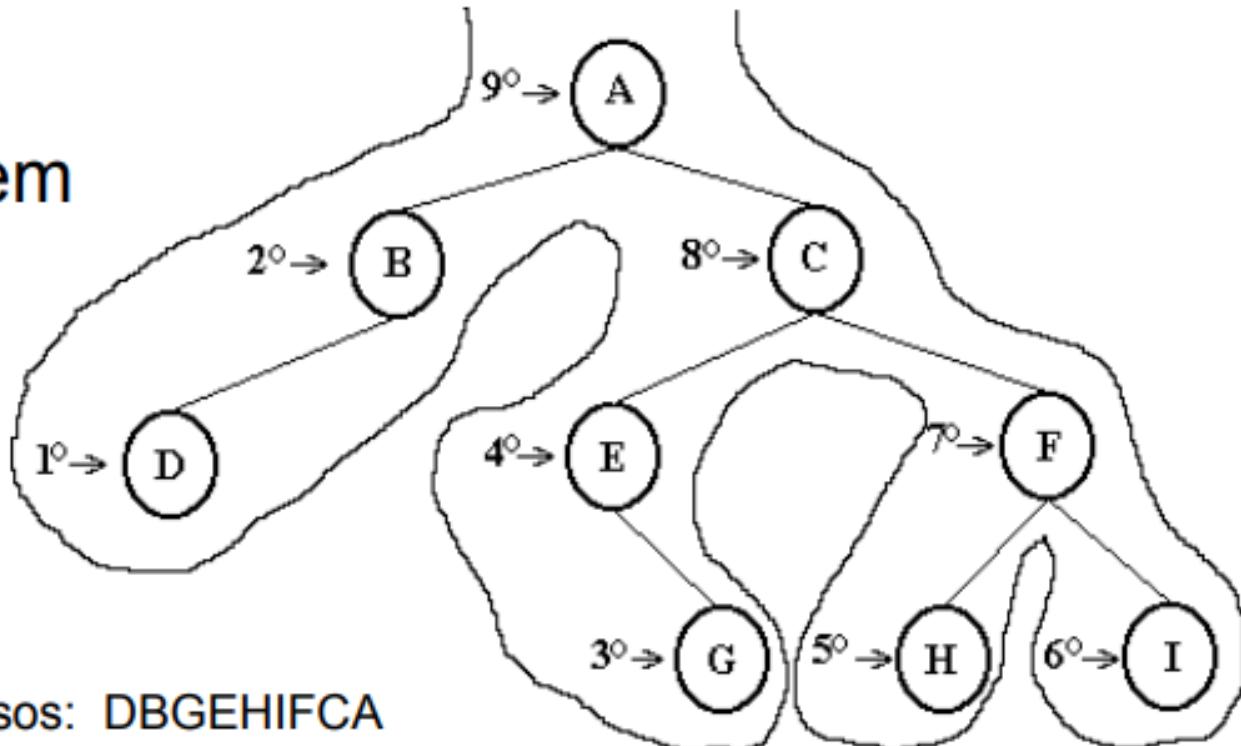


Ordem dos nós impressos: DBAEGCHFI

- Se a árvore estiver vazia, então fim
 - Percorra em In-ordem a subárvore da esquerda.
 - Visite a Raiz.
 - Percorra em In-ordem a subárvore da direita.

Árvore Binária de Busca

- Pesquisa
 - Pós-Ordem



Ordem dos nós impressos: DBGEHIFCA

- Se a árvore estiver vazia, então fim
 - Percorra em pós-ordem a subárvore da esquerda.
 - Percorra em pós-ordem a subárvore da direita.
 - Visite a Raiz.

Árvore Binária de Busca

- Estruturas:
 - tArvore
 - tNo

Árvore Binária de Busca

- Funções:

- struct tArvore * criaArvore(void);
- void lerNo(struct tNo *);
- void criaRaiz(struct tArvore *);
- struct tNo * criaNo(void);
- struct tNo * pai(struct tNo *);
- struct tNo * filhoDireita(struct tNo *);
- struct tNo * filhoEsquerda(struct tNo *);
- struct tNo * antecessor(struct tNo *);
- struct tNo* sucessor(struct tNo *);
- int ehFilhoDireita(struct tNo *);
- int ehFilhoEsquerda(struct tNo *);
- void insere(struct tArvore *);
- void remover(struct tArvore *);
- void preOrdem(struct tNo *);
- void inOrdem(struct tNo *);
- void posOrdem(struct tNo *);
- void listar(struct tArvore *);

Árvore Binária de Busca

- Implementação da Estrutura no Nó

```
1. struct tNo  
2. {  
3.     int chave;  
4.     struct tNo *pai, *esquerda, *direita;  
5. };
```

Por simplificação do exemplo nossos itens terão apenas um valor inteiro, aqui pode-se acrescentar outros campos para a estrutura.

- Implementação da Estrutura da Árvore

```
1. struct tArvore  
2. {  
3.     struct tNo *raiz;  
4.     int quantidade;  
5. };
```

Árvore Binária de Busca

- Inicialização da Árvore

```
1. struct tArvore * criaArvore(void) {  
2.     struct tArvore * arvore = (struct tArvore *) malloc(sizeof(struct tArvore));  
3.     if(arvore != NULL) {  
4.         arvore->raiz = NULL;  
5.         arvore->quantidade = 0;  
6.     }  
7.     else {  
8.         printf("\nDificuldade na alocacao de memoria!\n");  
9.         system("pause");  
10.    }  
11.    return arvore;  
12. }
```

- Inicialização do Raiz

```
1. void criaRaiz(struct tArvore *arvore) {  
2.     struct tNo *raiz = criaNo();  
3.     if(raiz != NULL) {  
4.         arvore->raiz = raiz;  
5.         arvore->quantidade ++;  
6.     }  
7. }
```

Árvore Binária de Busca

- Inicialização do Nó

```
1. struct tNo * criaNo(void) {  
2.     struct tNo *no = (struct tNo *) malloc(sizeof(struct tNo));  
3.     if(no != NULL) {  
4.         lerNo(no);  
5.         no->pai = NULL;  
6.         no->direita = NULL;  
7.         no->esquerda = NULL;  
8.     }  
9.     else {  
10.         printf("\nDificuldade na alocacao de memoria!\n");  
11.         system("pause");  
12.     }  
13.     return no;  
14. }
```

- Leitura do Nó

```
1. void lerNo(struct tNo *no) {  
2.     printf("\nInforme o valor da chave: \n");  
3.     scanf("%d", &no->chave);  
4.     /*ler demais campos da estrutura aqui!*/  
5. }
```

Árvore Binária de Busca

- Encontra o pai de um nó

```
1. struct tNo * pai(struct tNo *filho) {  
2.     return filho->pai;  
3. }
```

- Encontra o Filho da Direita

```
1. struct tNo * filhoDireita(struct tNo *raiz) {  
2.     return raiz->direita;  
3. }
```

- Encontra o Filho da Esquerda

```
1. struct tNo * filhoEsquerda(struct tNo *raiz) {  
2.     return raiz->esquerda;  
3. }
```

Árvore Binária de Busca

- **Retorna se é um filho da direita ou filho da esquerda**

```
1. int ehFilhoDireita(struct tNo *no) {  
2.     struct tNo *no_pai = pai(no);  
3.     if(no_pai == NULL) {  
4.         return 0;  
5.     }  
6.     else {  
7.         if(filhoDireita(no_pai) == no) {  
8.             return 1;  
9.         }  
10.    else {  
11.        return 0;  
12.    }  
13. }  
14. }
```

```
1. int ehFilhoEsquerda(struct tNo *no) {  
2.     struct tNo *no_pai = pai(no);  
3.     if(no_pai == NULL) {  
4.         return 0;  
5.     }  
6.     else {  
7.         if(filhoEsquerda(no_pai) == no) {  
8.             return 1;  
9.         }  
10.    else {  
11.        return 0;  
12.    }  
13. }  
14. }
```

Árvore Binária de Busca

- Percurso Pré-Ordem

```
1. void preOrdem(struct tNo* raiz)  {
2.     if(raiz != NULL)  {
3.         printf("\n%d\n", raiz->chave);
4.         preOrdem(raiz->esquerda);
5.         preOrdem(raiz->direita);
6.     }
7. }
```

Árvore Binária de Busca

- Percurso In-Ordem

```
1. void inOrdem(struct tNo* raiz)  {
2.     if(raiz != NULL)  {
3.         inOrdem(raiz->esquerda);
4.         printf("\n%d\n", raiz->chave);
5.         inOrdem(raiz->direita);
6.     }
7. }
```

Árvore Binária de Busca

- Percurso Pós-Ordem

```
1. void posOrdem(struct tNo* raiz)  {
2.     if(raiz != NULL)  {
3.         posOrdem(raiz->esquerda);
4.         posOrdem(raiz->direita);
5.         printf("\n%d\n", raiz->chave);
6.     }
7. }
```

Árvore Binária de Busca

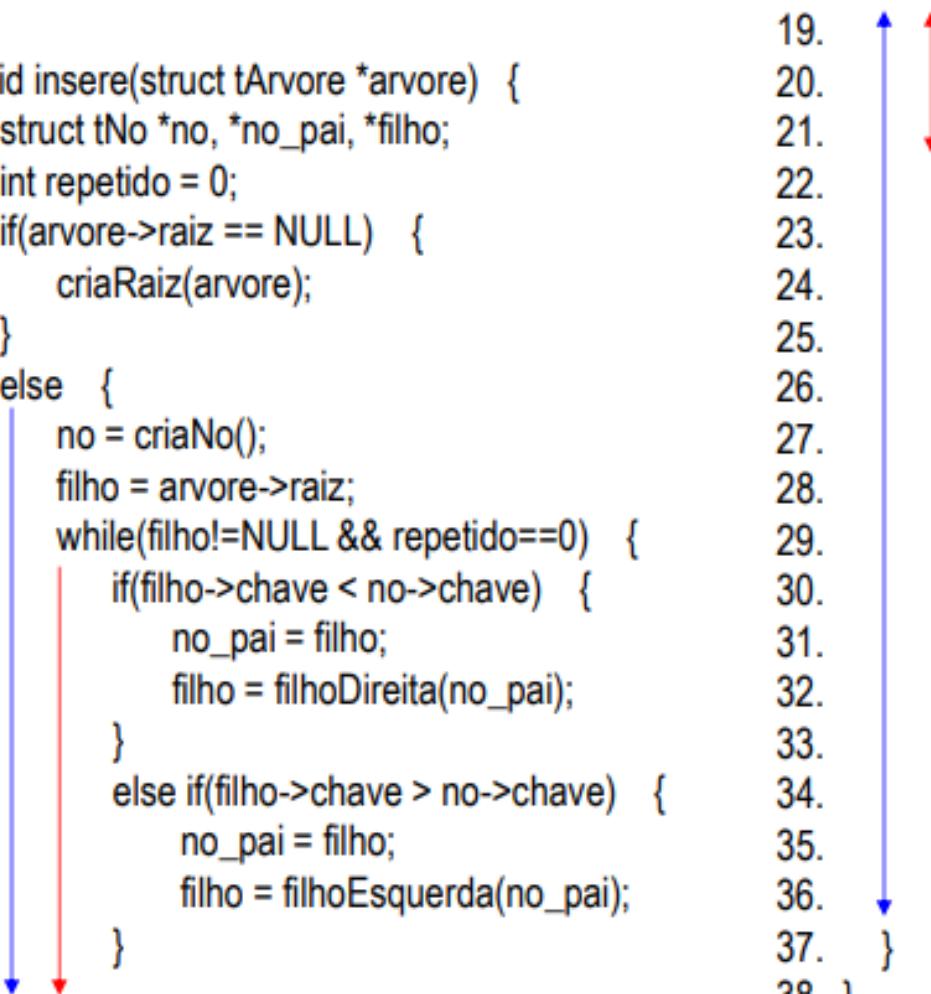
- Listar elementos da Árvore.

```
1. void listar(struct tArvore *arvore) {  
2.     if(arvore->raiz != NULL) {  
3.         /* invoque alguma das funções para percorrer árvore aqui*/  
4.     }  
5.     else {  
6.         printf("\nA arvore estah vazia!!!\n");  
7.         system("pause");  
8.     }  
9. }
```

Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {  
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }  
19.        }  
20.    }  
21.    else {  
22.        repetido = 1;  
23.    }  
24.    if(repetido == 0) {  
25.        if(no_pai->chave < no->chave) {  
26.            no_pai->direita = no;  
27.        }  
28.        else if(no_pai->chave > no->chave) {  
29.            no_pai->esquerda = no;  
30.        }  
31.        no->pai = no_pai;  
32.        arvore->quantidade++;  
33.    }  
34.    else {  
35.        printf("\nChave %d repetida na arvore!\n", ...);  
36.        system("pause");  
37.    }  
38. }
```



Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {
```

```
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }
```

Árvore Vazia!
Inserindo a raiz

```
19.     ↑   else {  
20.         ↑   repetido = 1;  
21.         ↓   }  
22.     }  
23.     if(repetido == 0) {  
24.         if(no_pai->chave < no->chave) {  
25.             no_pai->direita = no;  
26.         }  
27.         else if(no_pai->chave > no->chave) {  
28.             no_pai->esquerda = no;  
29.         }  
30.         no->pai = no_pai;  
31.         arvore->quantidade ++;  
32.     }  
33.     else {  
34.         printf("\nChave %d repetida na arvore!\n", ...);  
35.         system("pause");  
36.     }  
37. }  
38. }
```

Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {  
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }  
19.        }  
20.    }  
21.    else {  
22.        if(repetido == 0) {  
23.            if(no_pai->chave < no->chave) {  
24.                no_pai->direita = no;  
25.            }  
26.            else if(no_pai->chave > no->chave) {  
27.                no_pai->esquerda = no;  
28.            }  
29.            no->pai = no_pai;  
30.            arvore->quantidade++;  
31.        }  
32.        else {  
33.            printf("\nChave %d repetida na arvore!\n", ...);  
34.            system("pause");  
35.        }  
36.    }  
37. }
```

Buscando a
posição de
inserção

```
19.     }  
20. }  
21. }  
22. }  
23. }  
24. }  
25. }  
26. }  
27. }  
28. }  
29. }  
30. }  
31. }  
32. }  
33. }  
34. }  
35. }  
36. }  
37. }
```

Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {  
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }  
19.        }  
20.    }  
21.    else {  
22.        if(repetido == 0) {  
23.            if(no_pai->chave < no->chave) {  
24.                no_pai->direita = no;  
25.            }  
26.            else if(no_pai->chave > no->chave) {  
27.                no_pai->esquerda = no;  
28.            }  
29.            no->pai = no_pai;  
30.            arvore->quantidade++;  
31.        }  
32.        else {  
33.            printf("\nChave %d repetida na arvore!\n", ...);  
34.            system("pause");  
35.        }  
36.    }  
37.}
```

Buscando a
posição de
inserção

```
19.    }  
20.    else {  
21.        repetido = 1;  
22.    }  
23.    if(repetido == 0) {  
24.        if(no_pai->chave < no->chave) {  
25.            no_pai->direita = no;  
26.        }  
27.        else if(no_pai->chave > no->chave) {  
28.            no_pai->esquerda = no;  
29.        }  
30.        no->pai = no_pai;  
31.        arvore->quantidade++;  
32.    }  
33.    else {  
34.        printf("\nChave %d repetida na arvore!\n", ...);  
35.        system("pause");  
36.    }  
37.}
```

Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {  
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }  
19.        }  
20.    }  
21.    else {  
22.        repetido = 1;  
23.    }  
24.    if(repetido == 0) {  
25.        if(no_pai->chave < no->chave) {  
26.            no_pai->direita = no;  
27.        }  
28.        else if(no_pai->chave > no->chave) {  
29.            no_pai->esquerda = no;  
30.        }  
31.        no->pai = no_pai;  
32.        arvore->quantidade ++;  
33.    }  
34.    else {  
35.        printf("\nChave %d repetida na arvore!\n", ...);  
36.        system("pause");  
37.    }  
38. }
```

Se não for chave repetida. Insere na posição localizada.

Árvore Binária de Busca

- Inclusão de itens na árvore

```
1. void insere(struct tArvore *arvore) {  
2.     struct tNo *no, *no_pai, *filho;  
3.     int repetido = 0;  
4.     if(arvore->raiz == NULL) {  
5.         criaRaiz(arvore);  
6.     }  
7.     else {  
8.         no = criaNo();  
9.         filho = arvore->raiz;  
10.        while(filho!=NULL && repetido==0) {  
11.            if(filho->chave < no->chave) {  
12.                no_pai = filho;  
13.                filho = filhoDireita(no_pai);  
14.            }  
15.            else if(filho->chave > no->chave) {  
16.                no_pai = filho;  
17.                filho = filhoEsquerda(no_pai);  
18.            }  
19.        }  
20.    }  
21.    else {  
22.        repetido = 1;  
23.    }  
24.    if(repetido == 0) {  
25.        if(no_pai->chave < no->chave) {  
26.            no_pai->direita = no;  
27.        }  
28.        else if(no_pai->chave > no->chave) {  
29.            no_pai->esquerda = no;  
30.        }  
31.        no->pai = no_pai;  
32.        arvore->quantidade++;  
33.    }  
34.    else {  
35.        printf("\nChave %d repetida na arvore!\n", ...);  
36.        system("pause");  
37.    }  
38. }
```

↑ ↑
↓ ↓

else {
 repetido = 1;
} **Se não for chave repetida.**
Insere na posição localizada.

if(repetido == 0) {
 if(no_pai->chave < no->chave) {
 no_pai->direita = no;
 }
 else if(no_pai->chave > no->chave) {
 no_pai->esquerda = no;
 }
 no->pai = no_pai;
 arvore->quantidade++;
}

else {
 printf("\nChave %d repetida na arvore!\n", ...);
 system("pause");
}