



UTFPR –DAINF  
Ciência da Computação  
Algoritmos e Estruturas de Dados 2

# Estruturas Lineares

---

- ▶ Estruturas estudadas até agora são **sequenciais**
  - ▶ A) Elementos contiguamente armazenados
    - ▶ Regra geral: Dado elemento armazenado na posição  $p$ :
      - Significado de  $p-1$ : posição do elemento que precede elemento em  $p$
      - Significado de  $p+1$ : posição do elemento que sucede elemento em  $p$
  - ▶ B) Elementos não contiguamente armazenados
    - ▶ Elementos **referenciam**, sucessor e predecessor. Ex.: Listas dinâmicas.
- ▶ Característica das estruturas lineares (caso contíguo):
  - ▶ Endereço/referência apenas indica local de armazenamento de outros elementos. Qual a implicação?
    - ▶ **A menos que sejam regularmente ordenados**, efetuar operações básicas pode levar a acessar todos os elementos.
- ▶ Ao ordenar elementos estamos tentando ...
  - ▶ Estabelecer relações de *hierarquia* entre eles
  - ▶ Vantagem: desempenho em procedimentos básicos



# Estruturas Lineares

---

- ▶ Conclusão: hierarquizar (ordenar) estruturas lineares é interessante para desempenho.
- ▶ Qual seria a vantagem de termos uma estrutura naturalmente hierárquica??
  - ▶ Em tese não precisaríamos re-hierarquizá-la regularmente
    - ▶ após cada modificação de inserção, atualização, etc.
  - ▶ Aumento de desempenho.
- ▶ Sugestões para uma estrutura capaz de armazenar dados hierarquicamente (i.e. Não necessariamente linear)?



# Estruturas Lineares x Não Lineares: **Ideia**

---

- ▶ Significado do endereço/referência:
- ▶ Em estruturas lineares:
  - ▶ Simplesmente o local de armazenamento de um dado
- ▶ Estruturas hierárquicas
  - ▶ Local de armazenamento de outro dado:
  - ▶ **Relação de hierarquia entre dados**

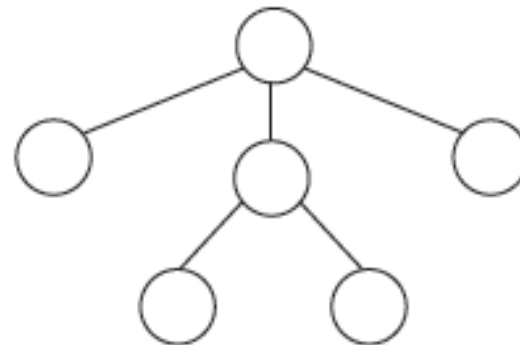
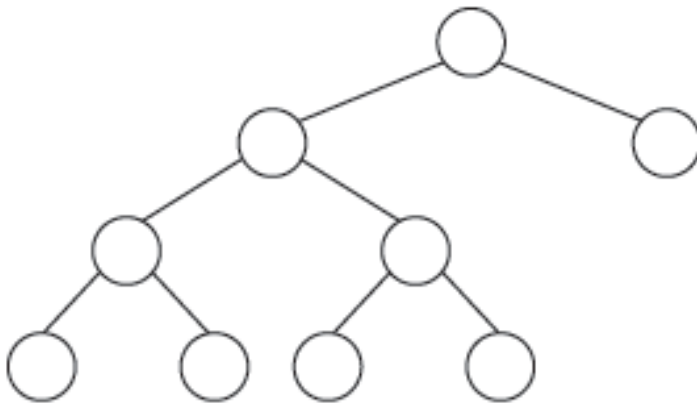


# Introdução - Árvores como Est. de Dados

---

## ▶ **Árvore:**

- ▶ Conseguem representação tanto linear como não-linear.
- ▶ *Possibilita* hierarquia na hora de “armazenar” os nós



# Árvore - Definição

---

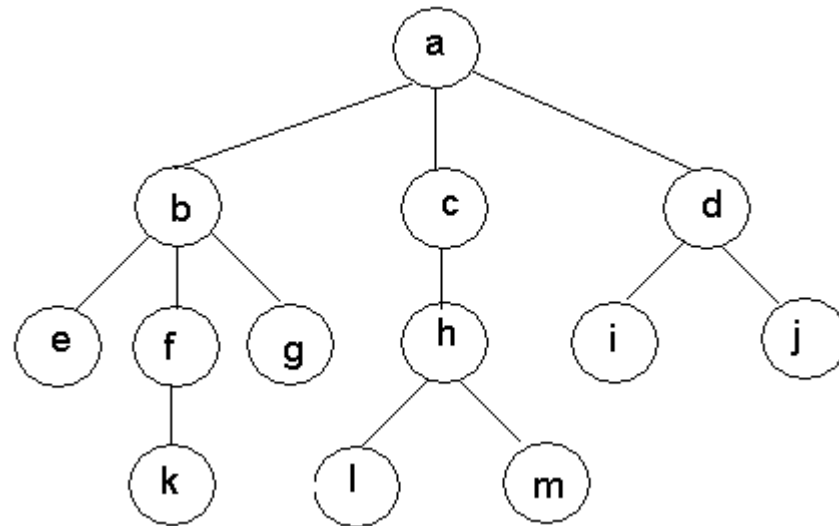
- ▶ É uma coleção finita de  $n \geq 0$  elementos.
  - ▶ Se  $n=0$ , dizemos que a árvore é vazia;
  - ▶ Se  $n>0$ 
    - ▶ Existe um nodo especial denominado **raiz**;
    - ▶ O nodo raiz contém referências para outras  $k$  árvores.  
Chamadas sub-árvores  $T_1, T_2, \dots, T_k$ :
    - ▶ Se  $T_k$  é subárvore de  $T$ , então  $T$  não é vazio



# Árvores – fundamentos...

---

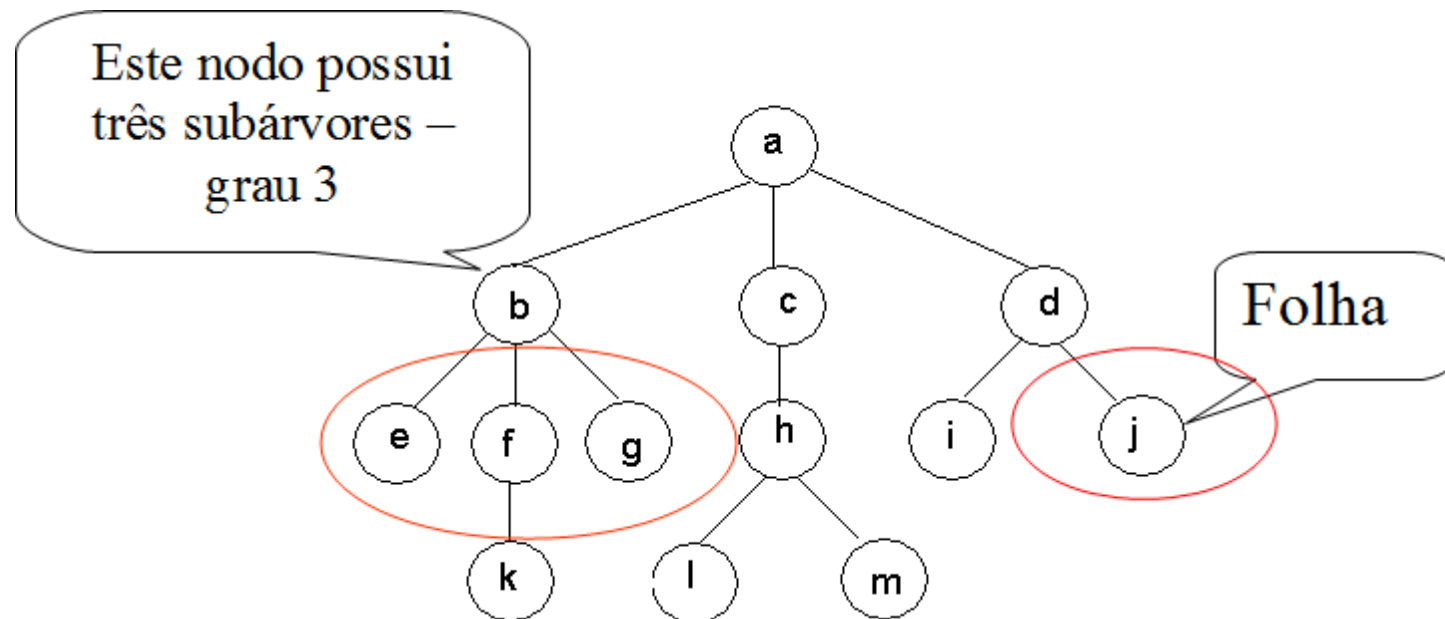
- ▶ Cada uma das estruturas  $T_i$  é organizada na forma de árvore – definição recursiva.



# Árvores – fundamentos...

## ► O **grau** de uma árvore

- é o número de subárvores do nó raiz.
- Um nó de grau igual a zero é chamado nó-folha ou terminal.



Podemos dizer que o maior grau de suas subárvores é o grau da árvore



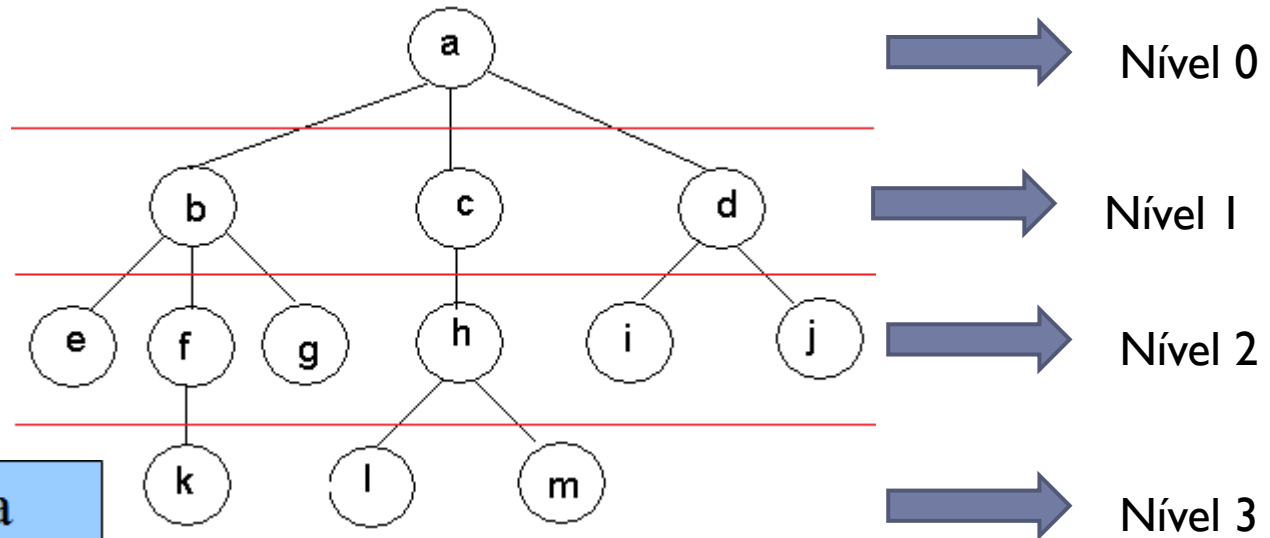
# Árvores – fundamentos...

## ► **Nível**

- distância (em número de ligações) de um nó até a raiz

## ► **Altura/profundidade**

- de uma árvore é o maior (i.e., mais profundo) nível da árvore.



Profundidade da  
árvore : 3

# Árvore – Fundamentos...

---

**Nó pai**: nó acima e com ligação direta a outro nó.

**Nó filho**: nó abaixo e com ligação direta a outro nó (o ***nó pai***).

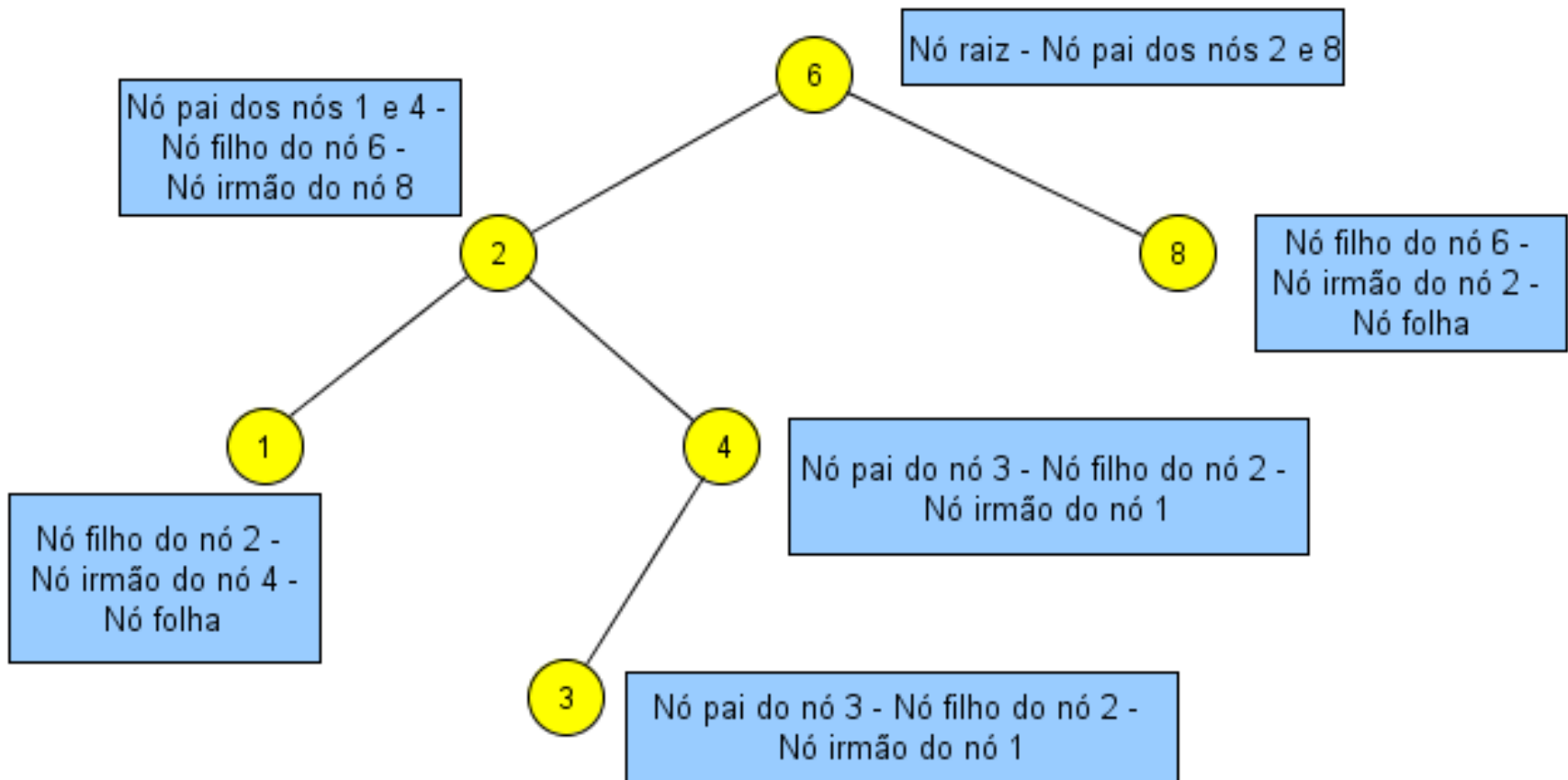
São os nós raízes das sub-árvores.

**Nós irmãos**: são nós que possuem o mesmo ***nó pai***.

**Nó folha ou terminal**: nó que ***não*** possui filhos.

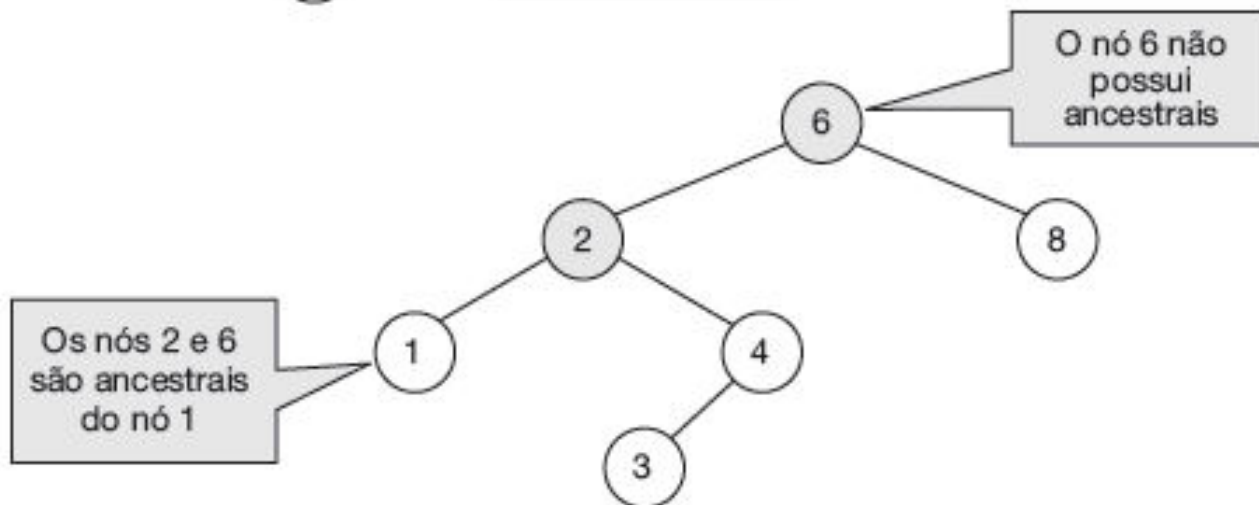
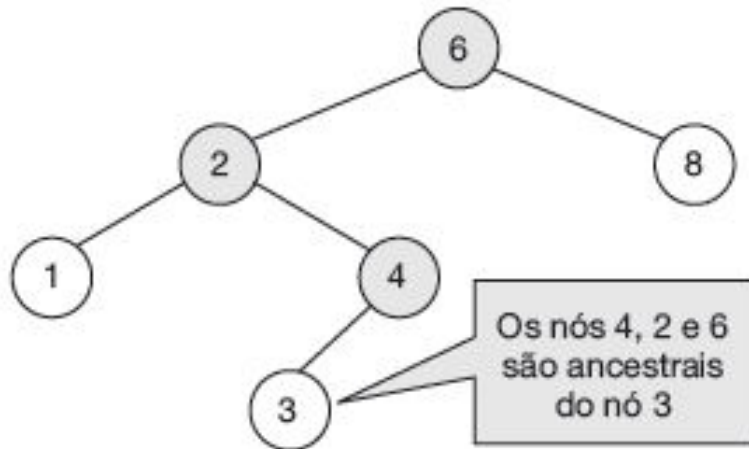


# Árvore – Fundamentos...



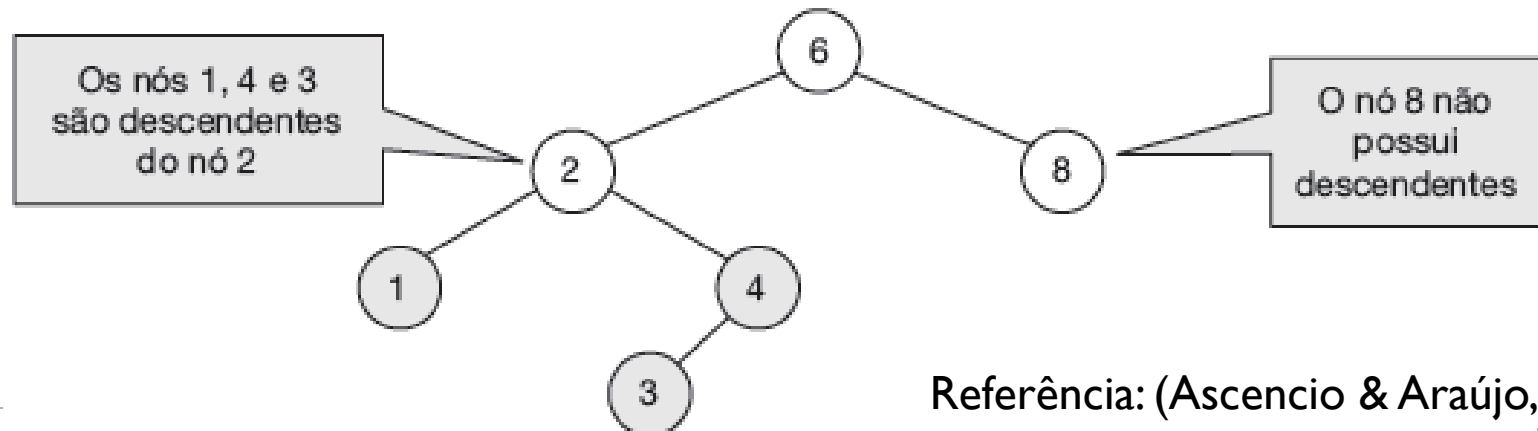
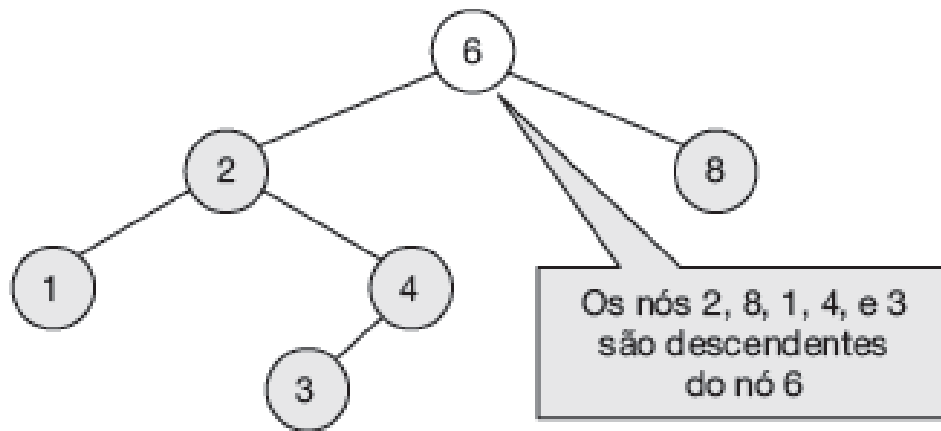
# Árvore - Fundamentos...

**Nós ancestrais:** nós que estão acima de um nó e têm ligação direta ou indireta.



# Árvore – Fundamentos...

**Nós descendentes**: nós estão abaixo de um nó e possuem ligação direta ou indireta.



Referência: (Ascencio & Araújo, 2011)

A vertical blue bar is located on the left side of the slide, partially enclosed by a thin white rectangular border.

# Árvores Binárias

# Árvore Binária - Definição

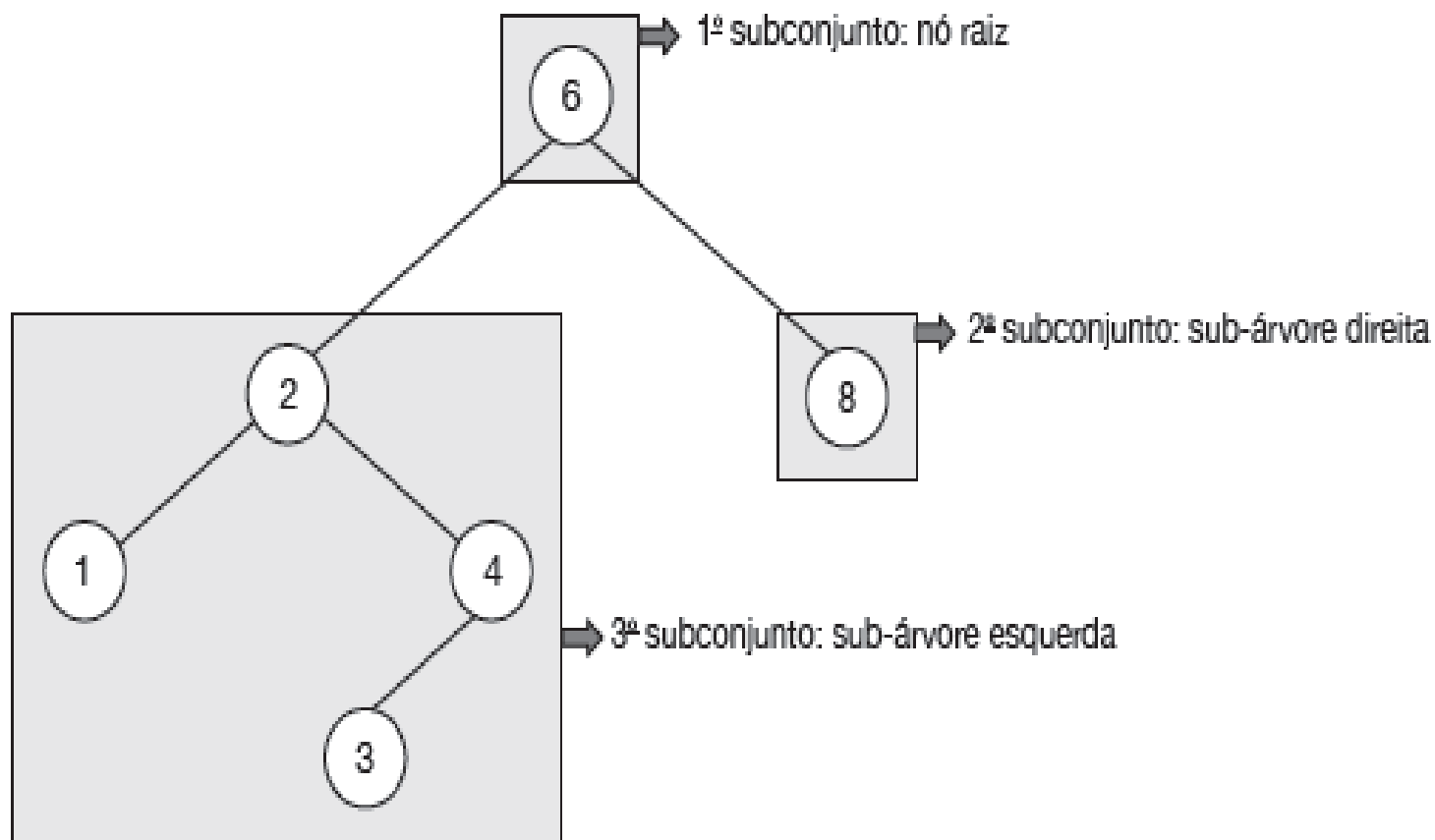
---

- ▶ Uma **Árvore Binária** é um caso particular de uma árvore em que cada nó tem grau máximo 2.
- ▶ **Definição Recursiva:**
  - ▶ Um conjunto  $T$  tal que :
    - ▶  $T$  contém, **no máximo**, 1 elemento.
    - ▶  $T$  possui duas subárvores: uma à esquerda e outra à direita ( $T_1, T_2$ ), respect.
    - ▶  $T_1, T_2$  são árvores binárias
- ▶ Uma árvore binária tem **grau máximo** igual a **2**.



# Árvore Binária - Definição

---



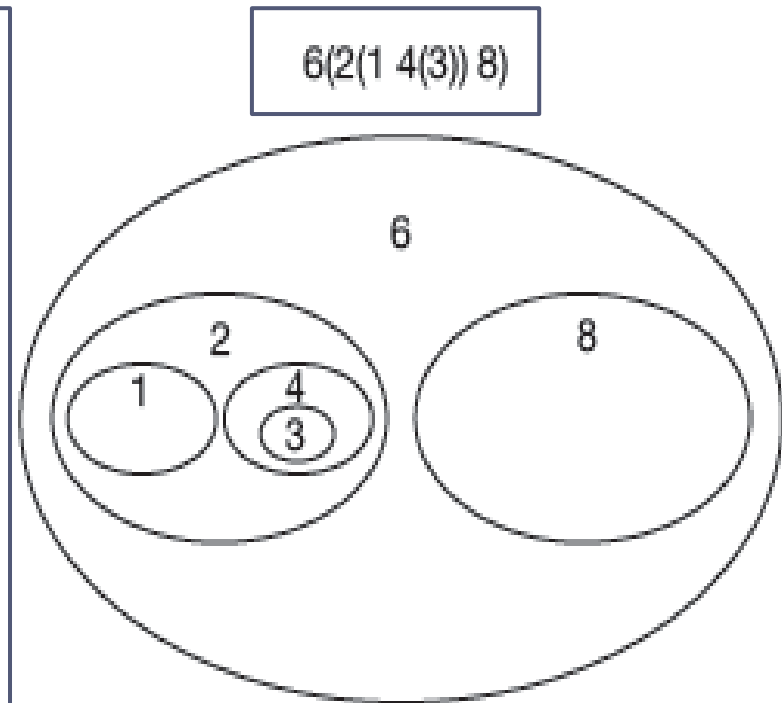
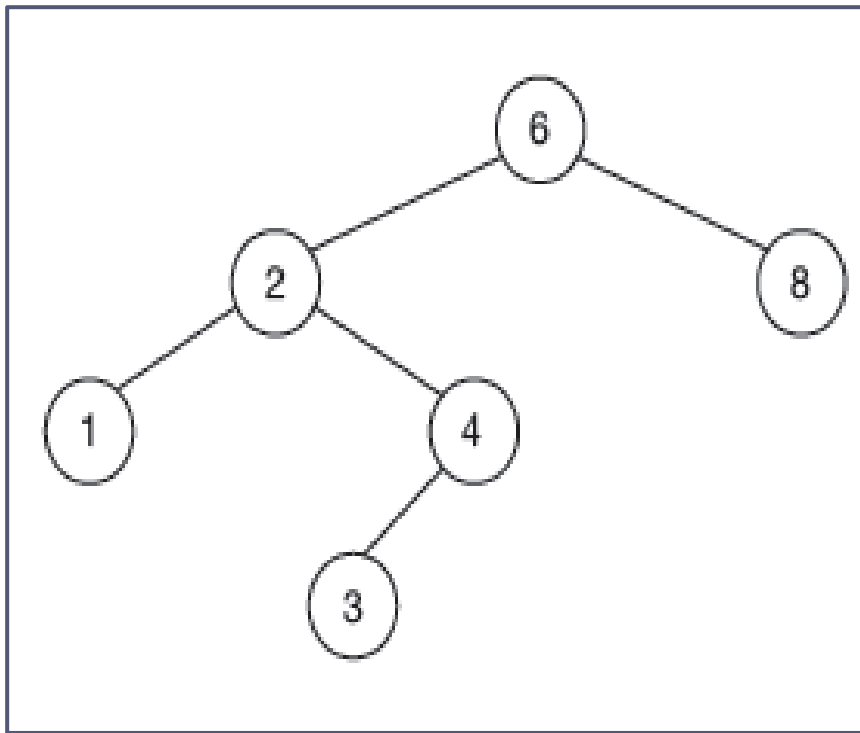
Referência: (Ascencio & Araújo, 2011)





# Árvore Binária - Ilustrações

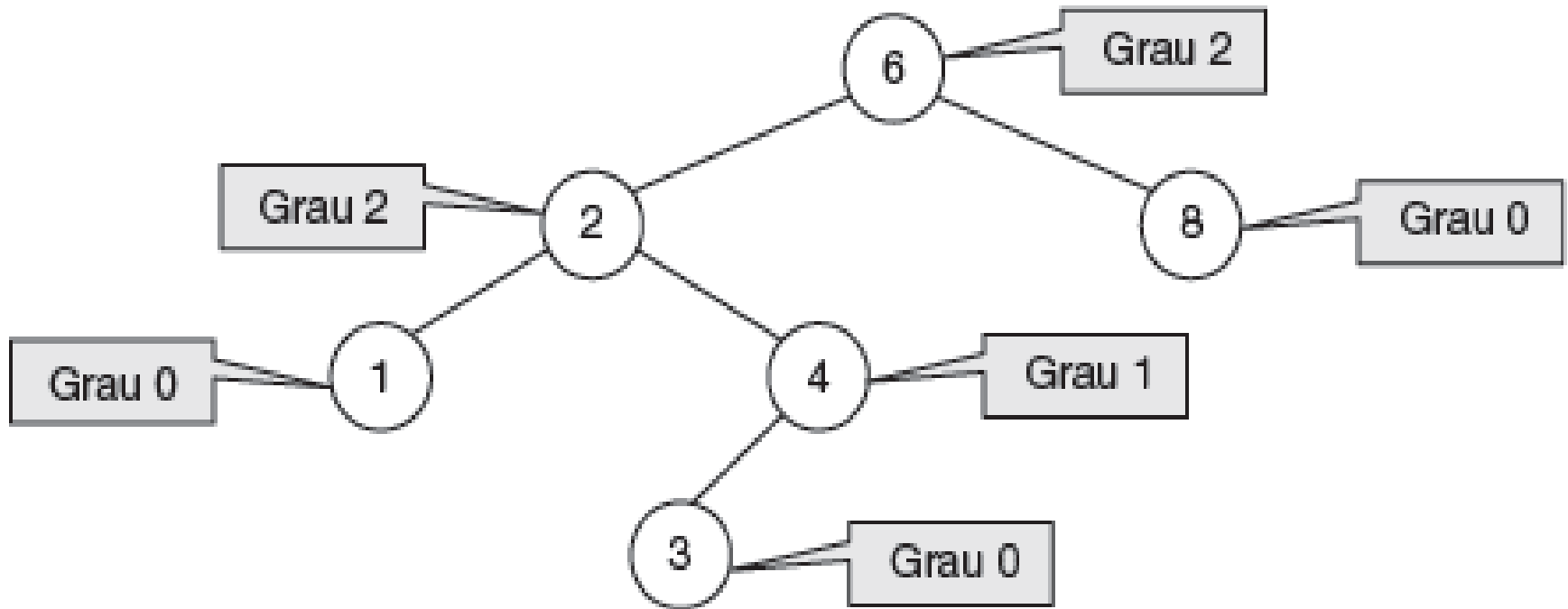
- ▶ As árvores binárias podem ser ilustradas de três formas distintas: Grafos, parentização raiz( $T_1, T_2$ ), diagramas de Ven



Referência: (Ascencio & Araújo, 2011)

# Árvore Binária - Propriedades & Definições

---



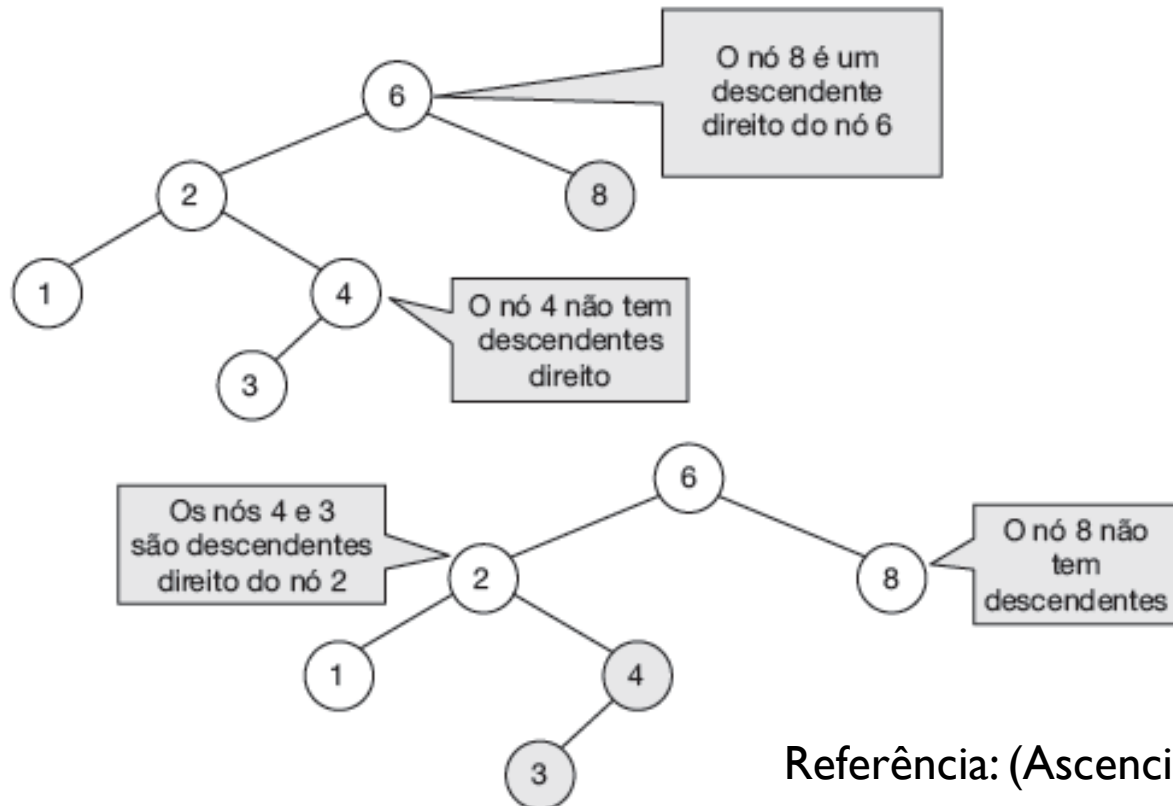
Referência: (Ascencio & Araújo, 2011)



# Árvore Binária- Fundamentos

## Nós descendentes direito:

descendentes da subárvore à direita Tl

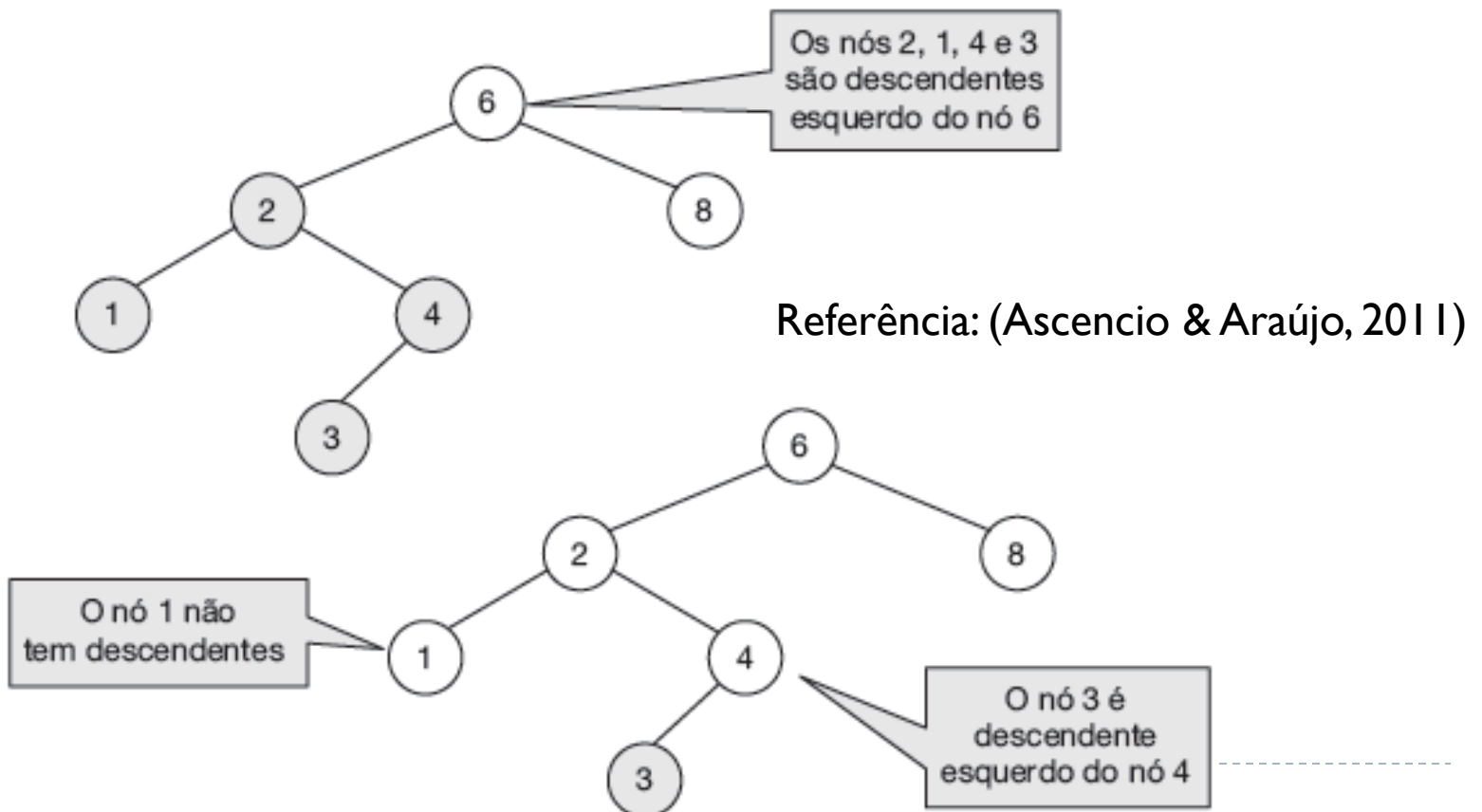


Referência: (Ascencio & Araújo, 2011)

# Árvore – Fundamentos...

## Nós descendentes esquerdo:

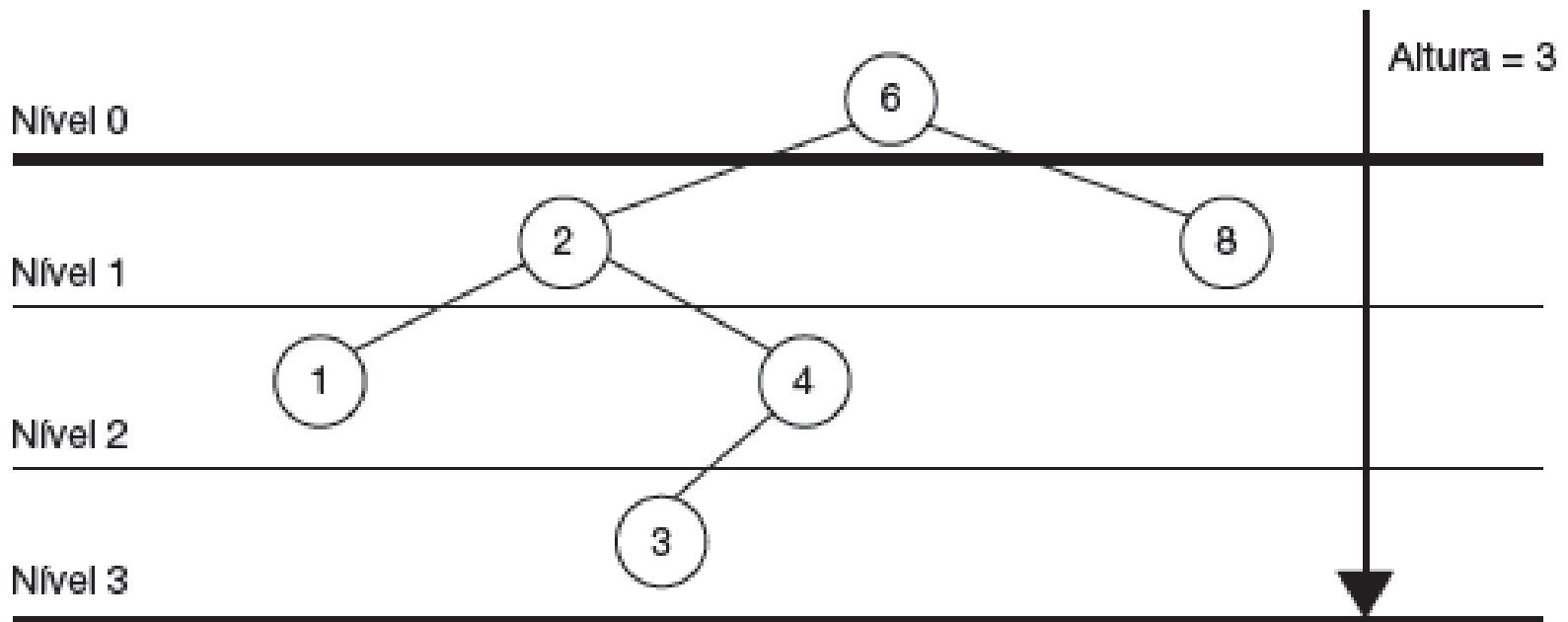
Descendente da sub-árvore *esquerda T2*



# Árvore Binária - Propriedades & Definições

**Nível de um nó**: número de ligação entre um nó e o raiz. Logo, o nível do nó raiz é sempre **zero**.

**Altura ou profundidade da árvore**: é o nível mais distante da raiz.



# Propriedades

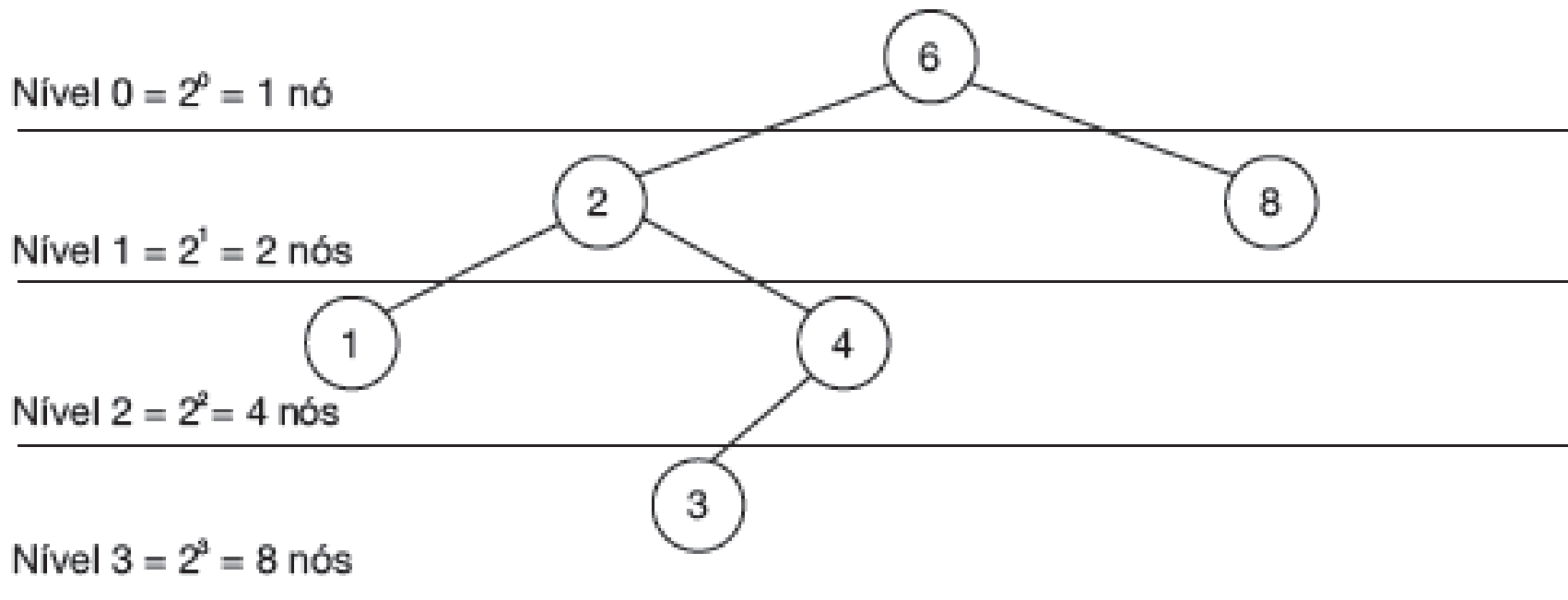
---

- ▶ Qual a **quantidade máxima** de nós que uma árvore binária apresenta no  $m$ -ésimo nível?



# Árvore Binária - Propriedades & Definições

Expressão que representa o *número máximo de nós em um nível da árvore binária* é definida por  $2^m$ , onde  $m$  é o nível em questão.

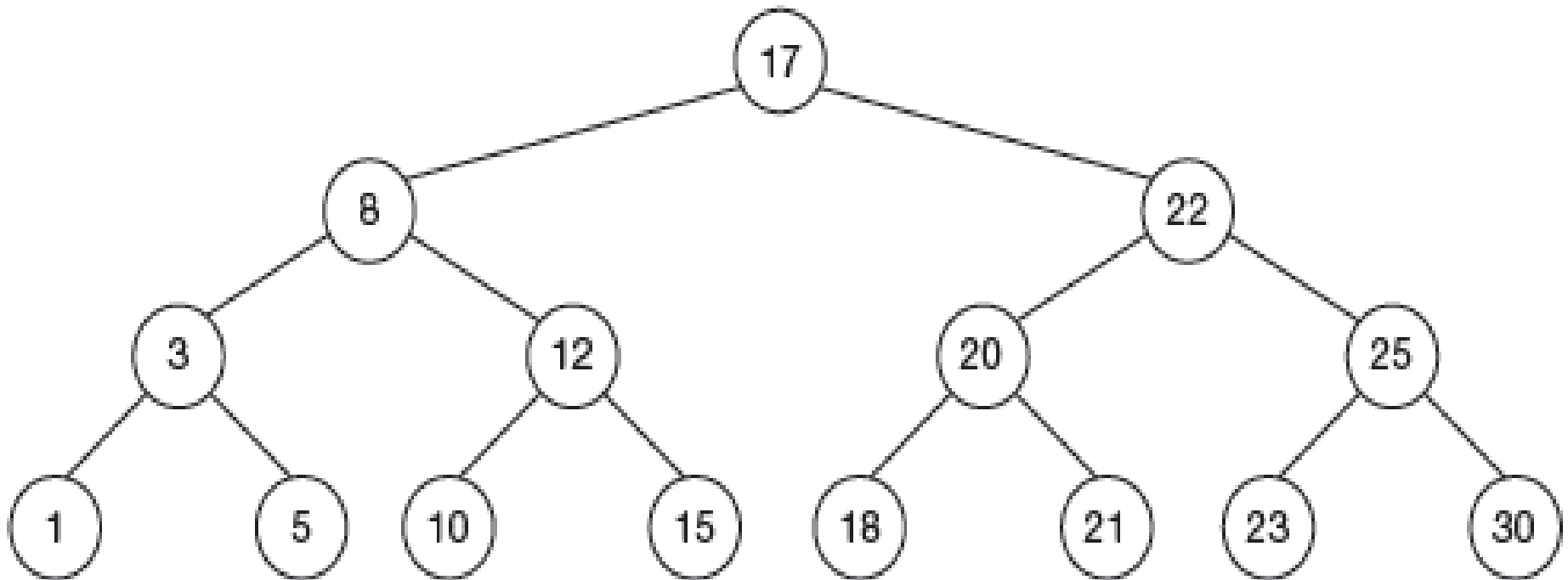


# Árvore Binária - Propriedades & Definições

---

## Árvore cheia (ou *binária completa*):

- Todos os níveis estão com a quantidade máxima de nós





# Árvore Binária

---

- ▶ Árvore é um formalismo não uma estrutura de dados
- ▶ Contudo, podemos usar a ideia das árvores para implementar uma estrutura de dados com características hierárquicas
- ▶ Alguma ideia?



# Árvore de Busca Binária – Propriedades

---

## ► Propriedades & Definições:

- Uma árvore binária, cuja raiz armazena o elemento  $r$ , é denominada **árvore de busca binária** se :
  - Todo elemento armazenado na subárvore esquerda é menor que  $r$ ;
  - Todo elemento armazenado na subárvore direita é maior que  $r$ ;
  - As subárvores esquerda e direita também são árvores de busca binária.
- Como implementamos uma árvore binária de busca (de agora em diante apenas “árvore binária”)?



# Árvore Binária: Estrutura

---

```
struct NoArvoreBin{  
    int chave;  
    struct NoArvoreBin* esq; //subarvore esquerda  
    struct NoArvoreBin* dir; //subarvore direita  
};
```

► **Note que:**

- Referências à `esq` e `dir` expressam não somente endereço do nó raiz das respectivas sub-árvores, mas também indicam uma relação de hierarquia pois é verdade que:
  - $chave > T1 \rightarrow chave$
  - $chave < T2 \rightarrow chave$



# Exercício em sala

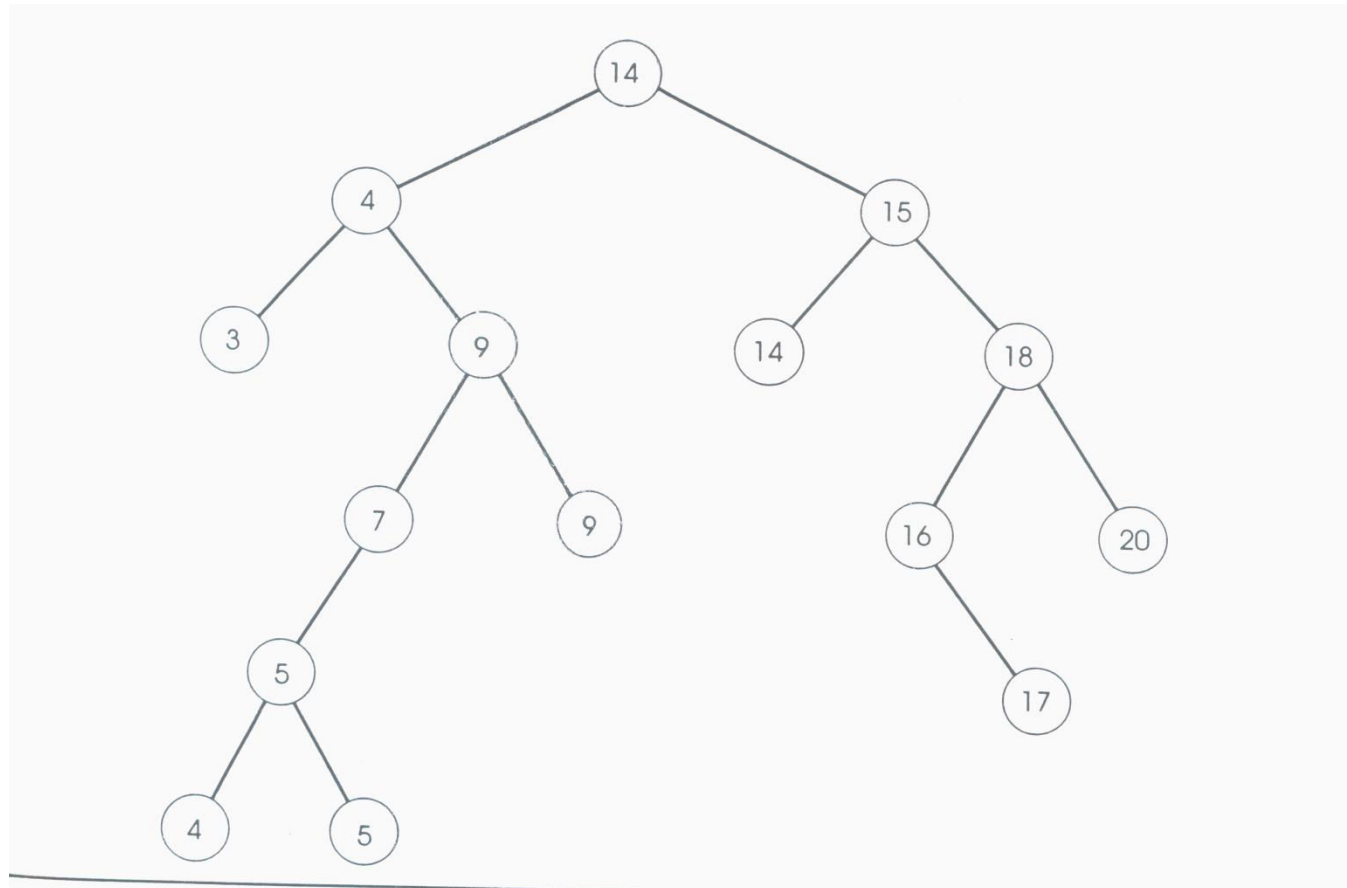
---


- Construa a árvore binária de pesquisa para a sequência: 14 – 15 – 4 – 9 – 7 – 18 – 3 – 5 – 16 – 4 – 20 – 17 – 9 – 14 – 5
- Sejam :
  - `struct NoArvoreBin * raiz;`: um ponteiro para o nó raiz de uma árvore binária de pesquisa
  - `int k;`: um valor chave a ser pesquisado
  - Escreva um algoritmo capaz de decidir se `k` está na árvore dada.



# Desenhe a árvore de busca binária

- Dada a seqüência: 14 – 15 – 4 – 9 – 7 – 18 – 3 – 5 – 16 – 4 – 20 – 17 – 9 – 14 – 5





## Procedimentos: Inserção e Remoção.

# Árvore Binária: inserção e remoção de nós

---

## ▶ Na inserção:

- ▶ as propriedades da árvore devem ser obedecidas e todo novo nó é sempre uma folha.

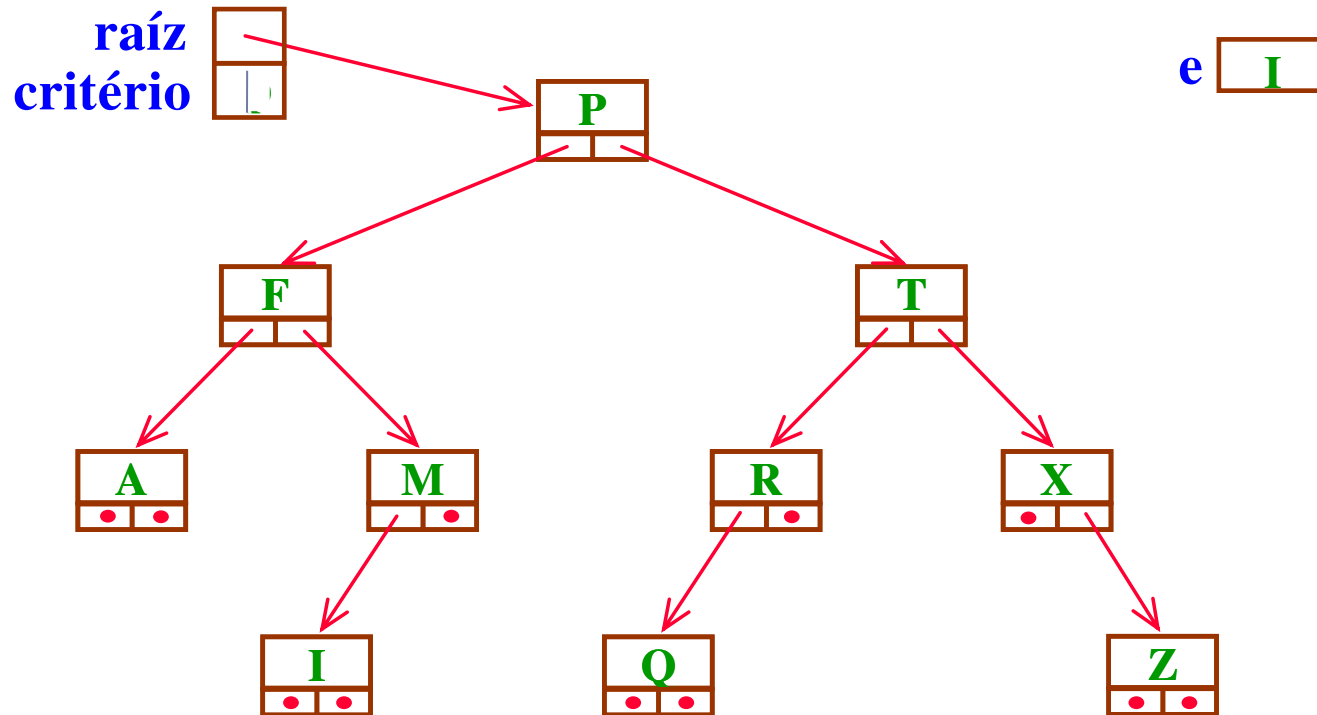
## ▶ Na remoção:

- ▶ Se nó for folha, apenas remova-o
- ▶ Se nó for não folha
  - ▶ É preciso determinar um novo nó para o lugar do removido



# Árvores Binárias de Pesquisa: Busca

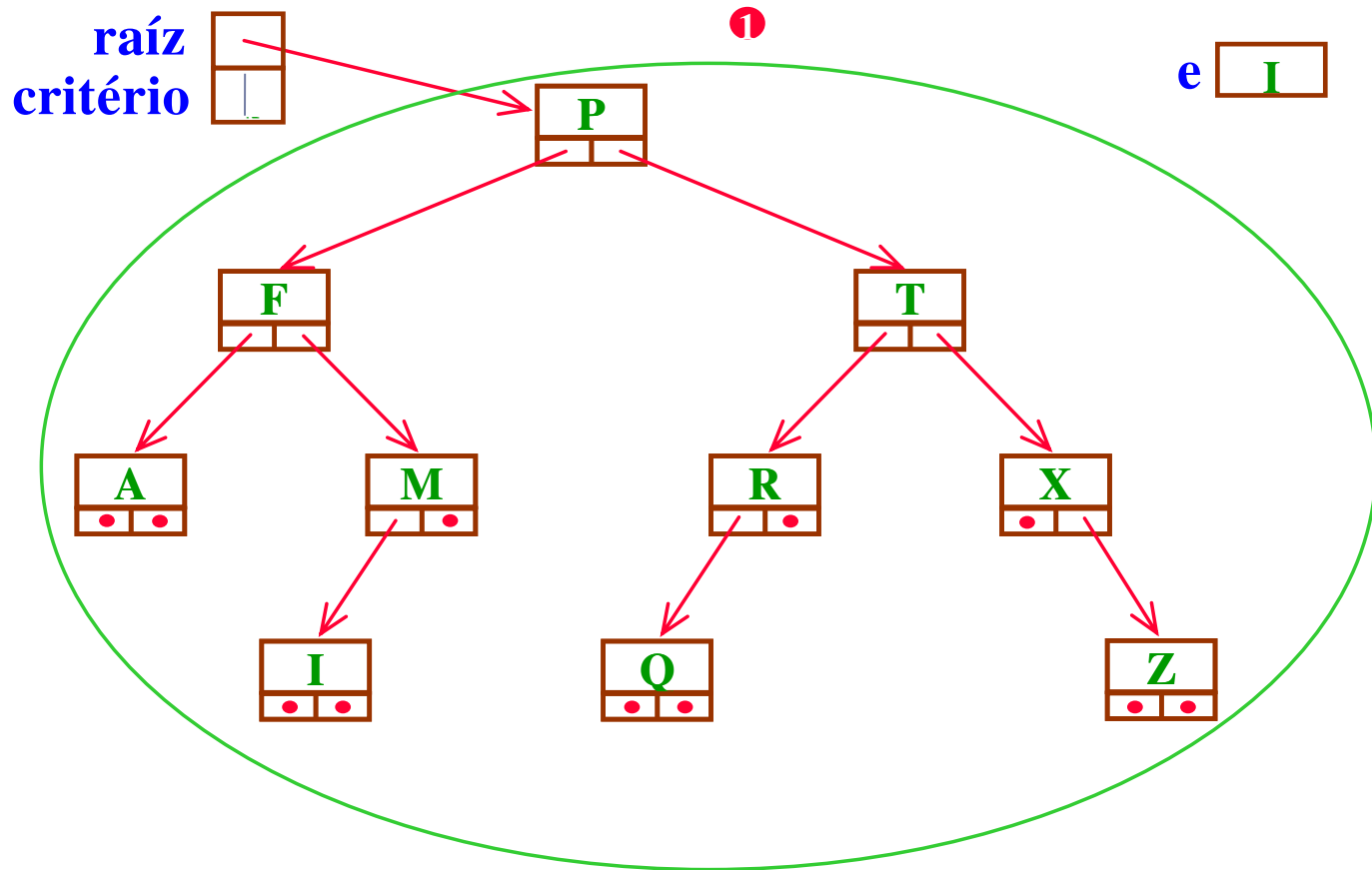
## ► Consulta





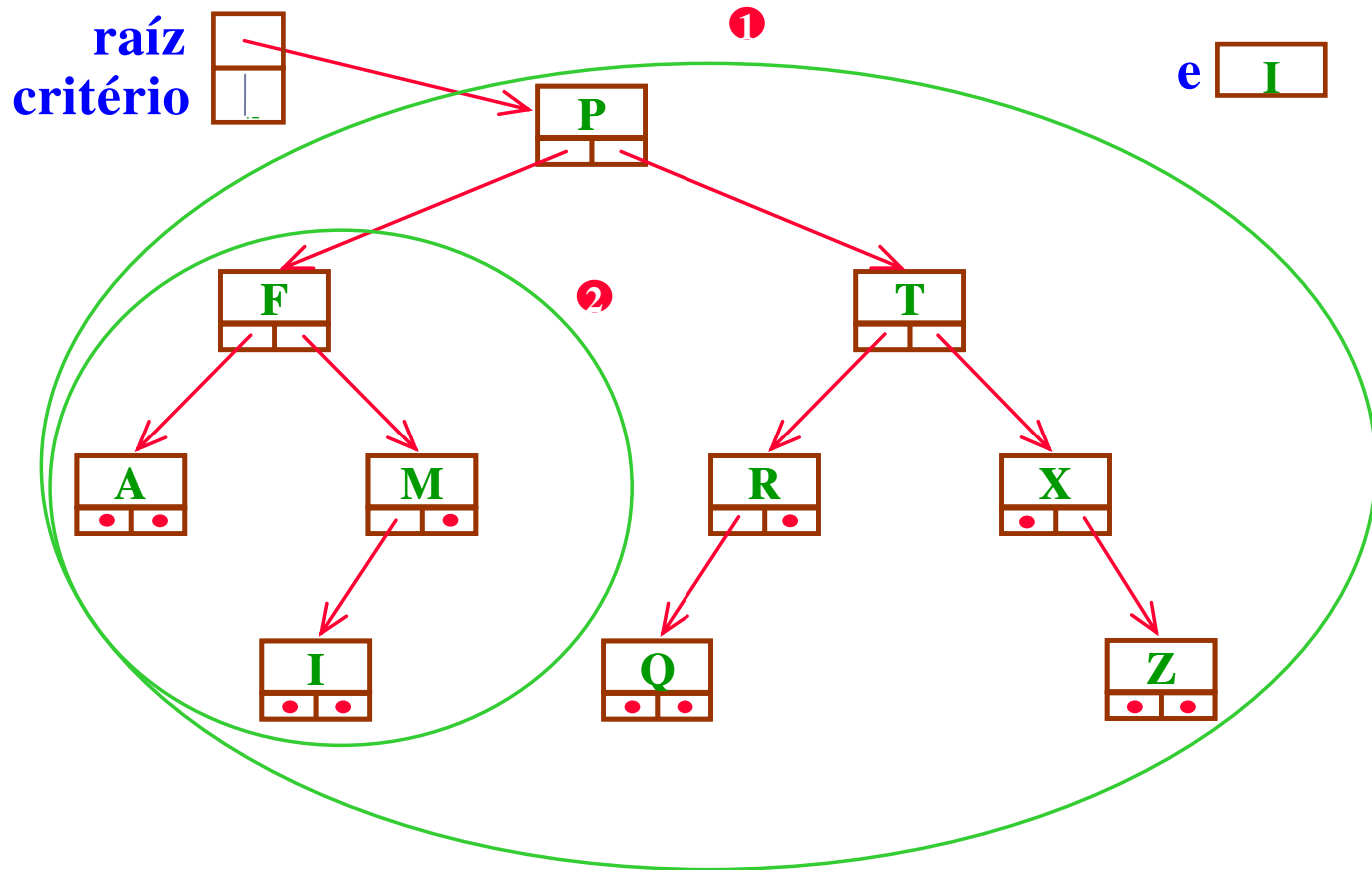
# Árvores Binárias de Pesquisa: Busca

## ► Consulta



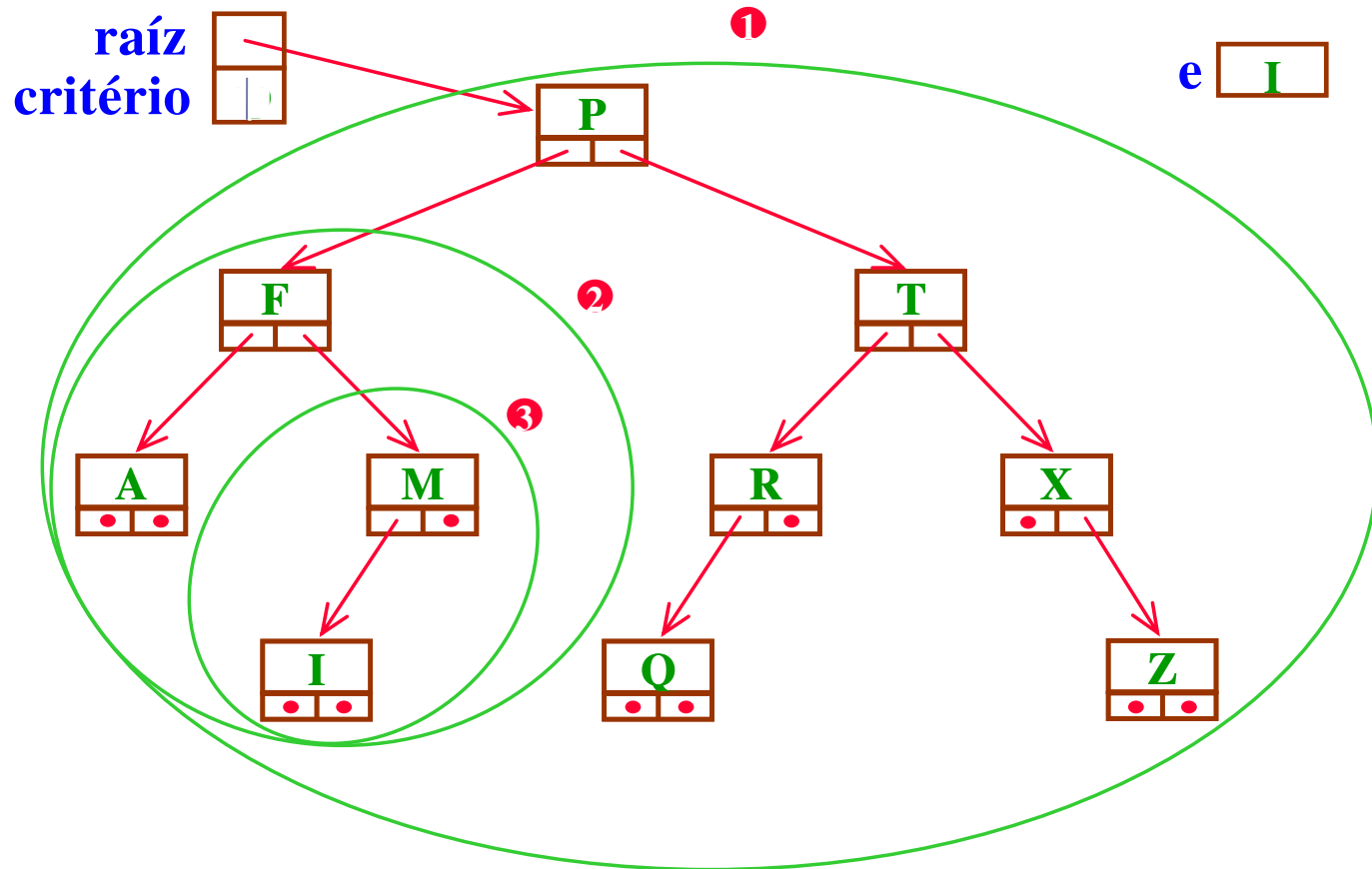
# Árvores Binárias de Pesquisa: Busca

## ► Consulta



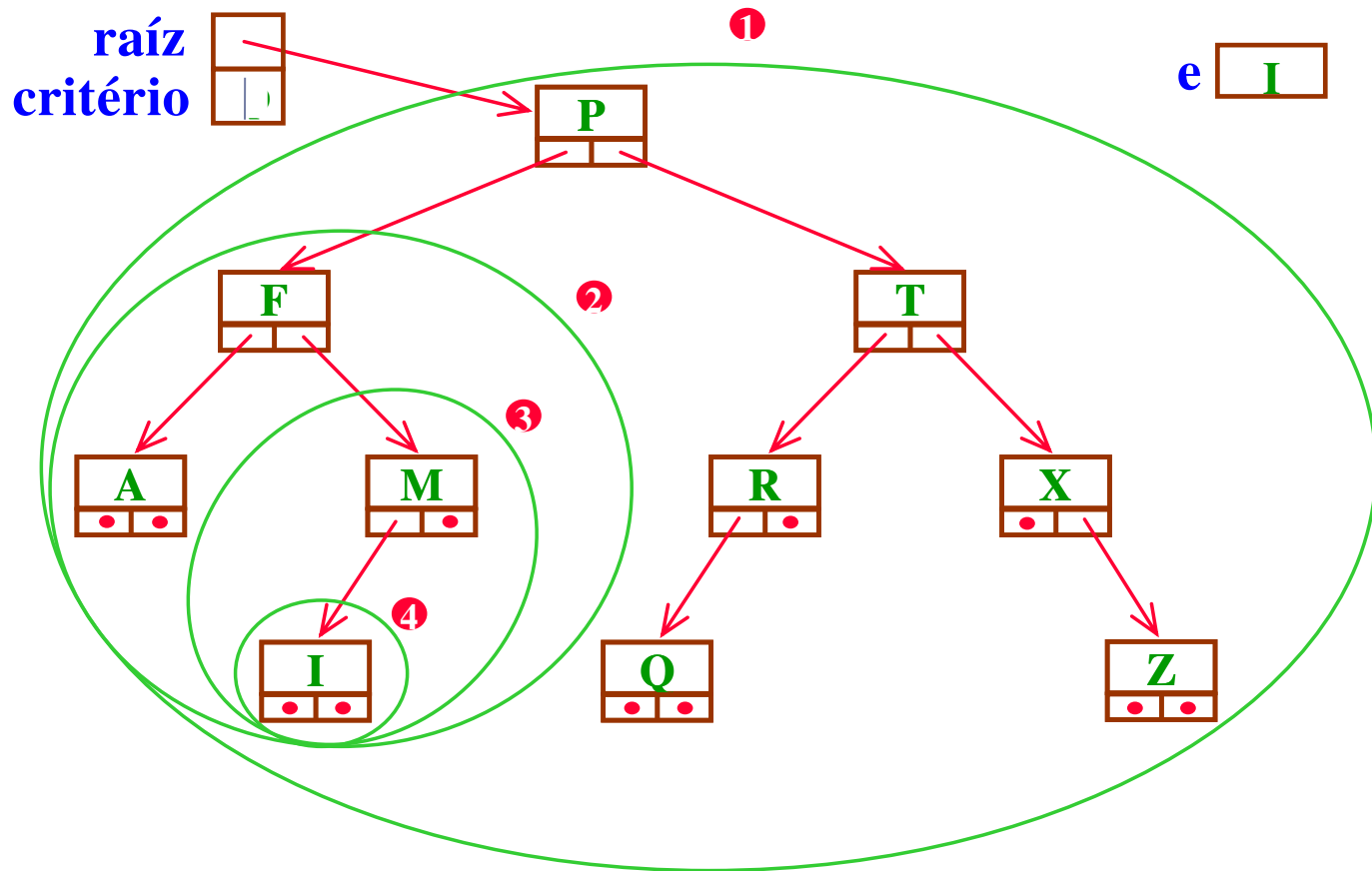
# Árvores Binárias de Pesquisa: Busca

## ► Consulta



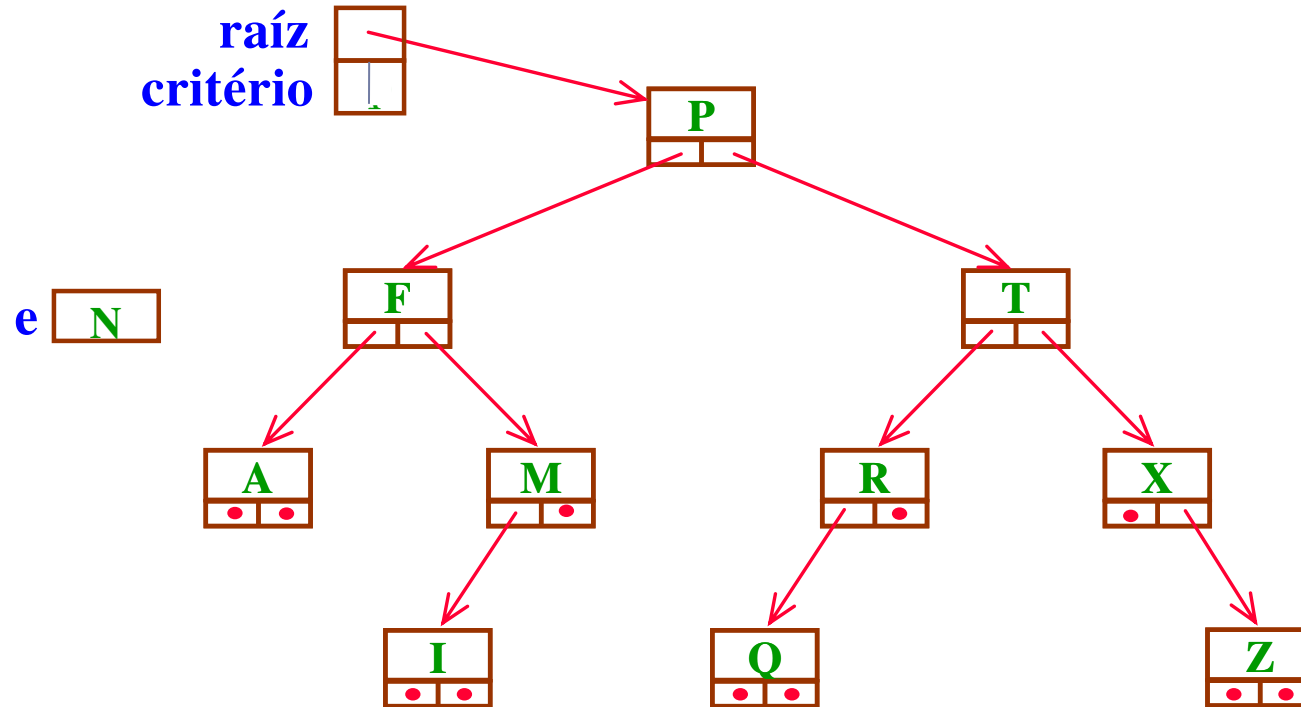
# Árvores Binárias de

## ► Consulta



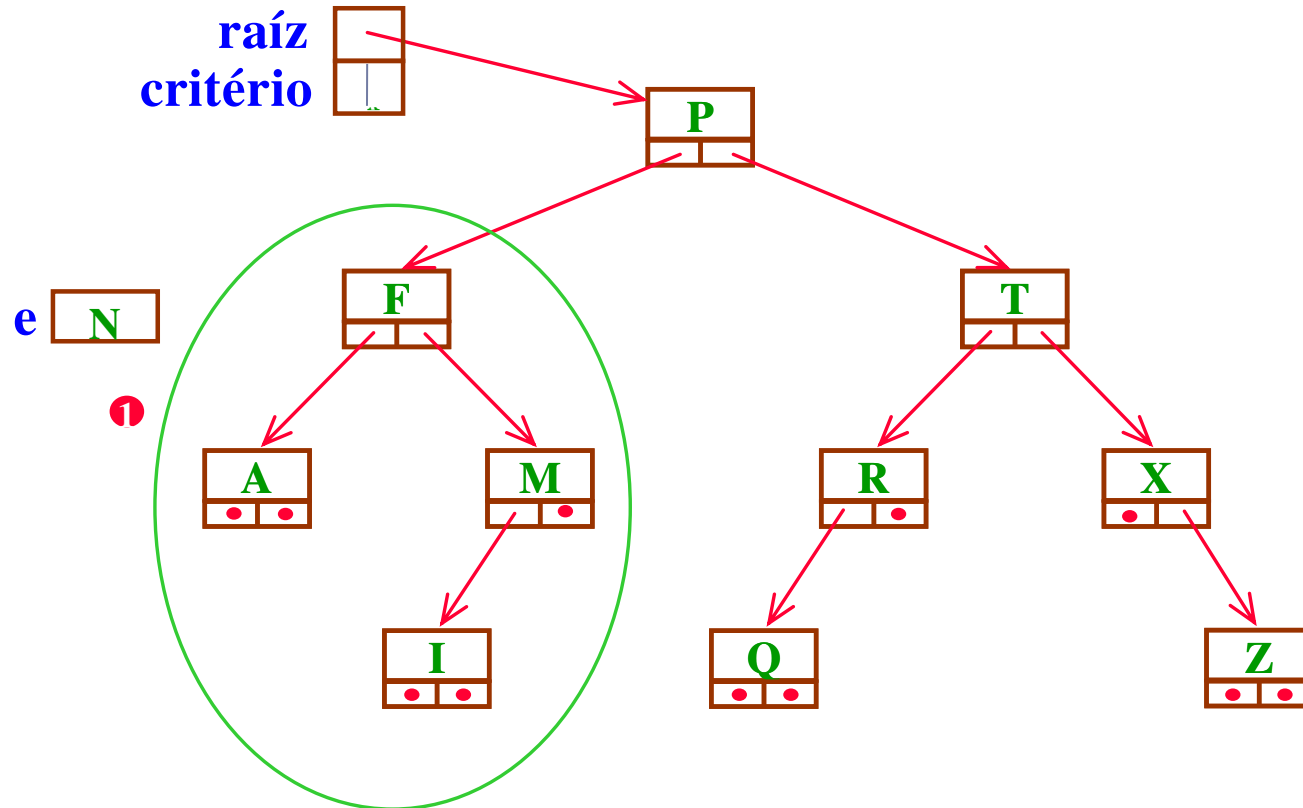
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



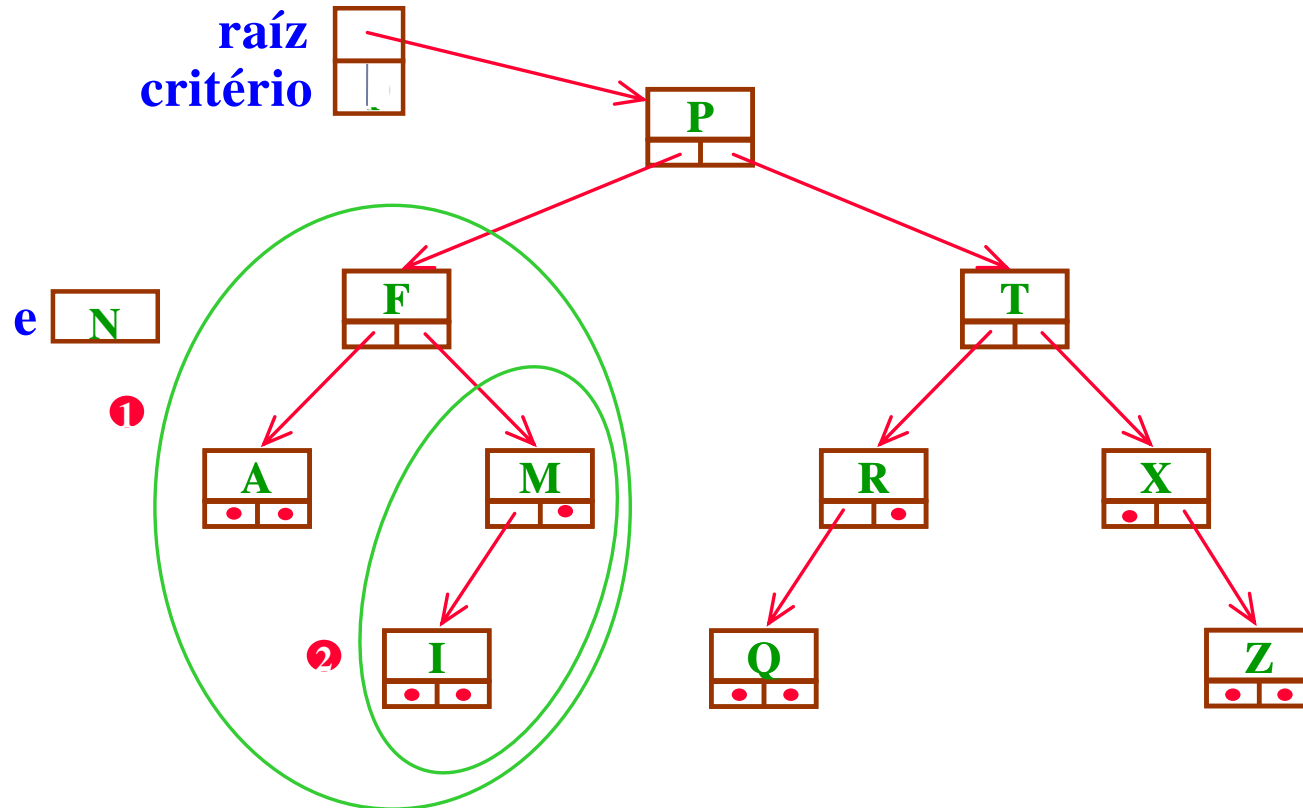
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



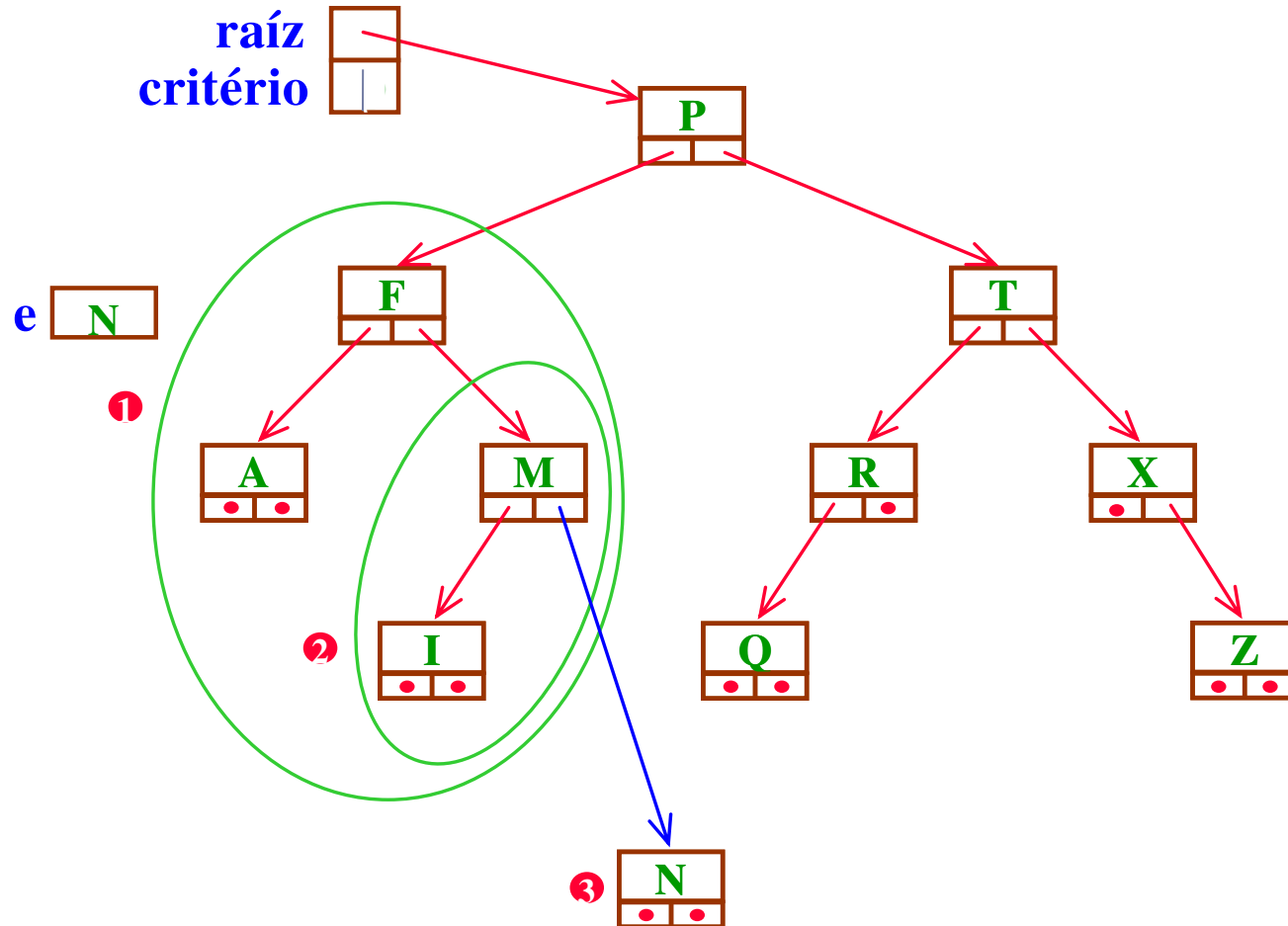
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



# Árvores Binárias de Pesquisa: Inserção

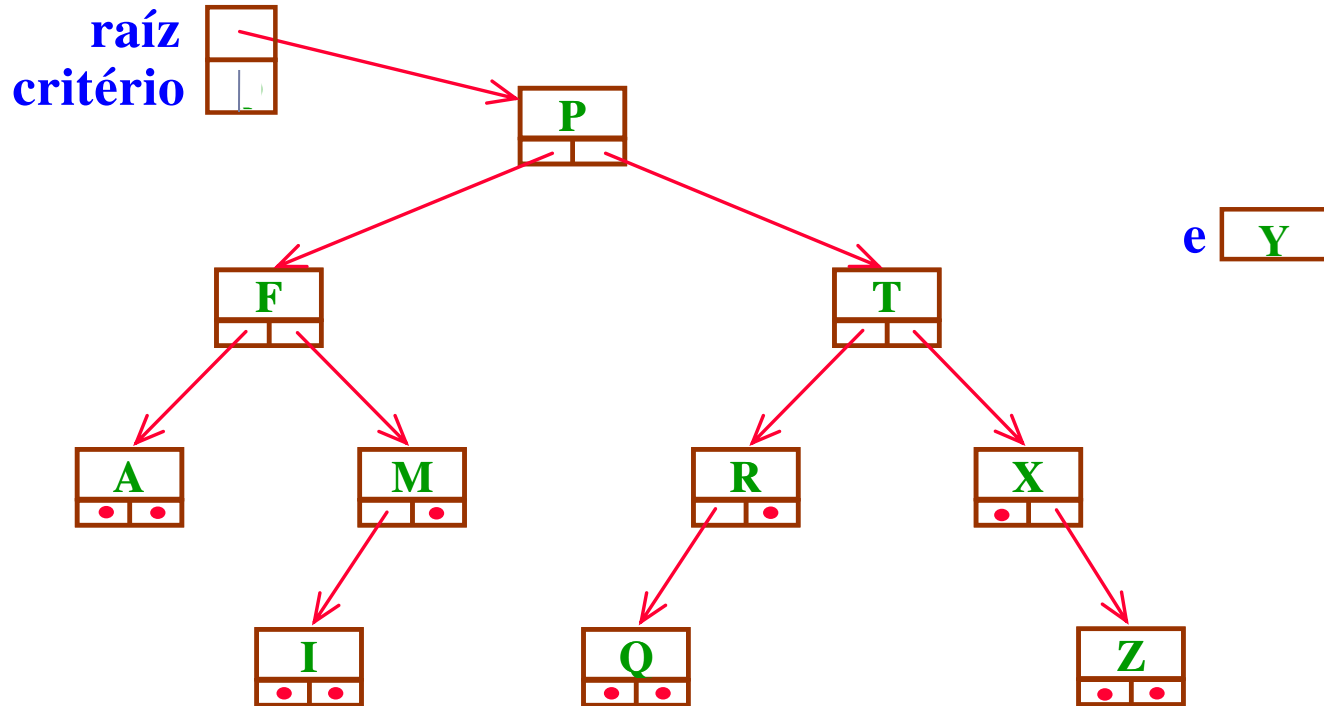
## ► Inserção





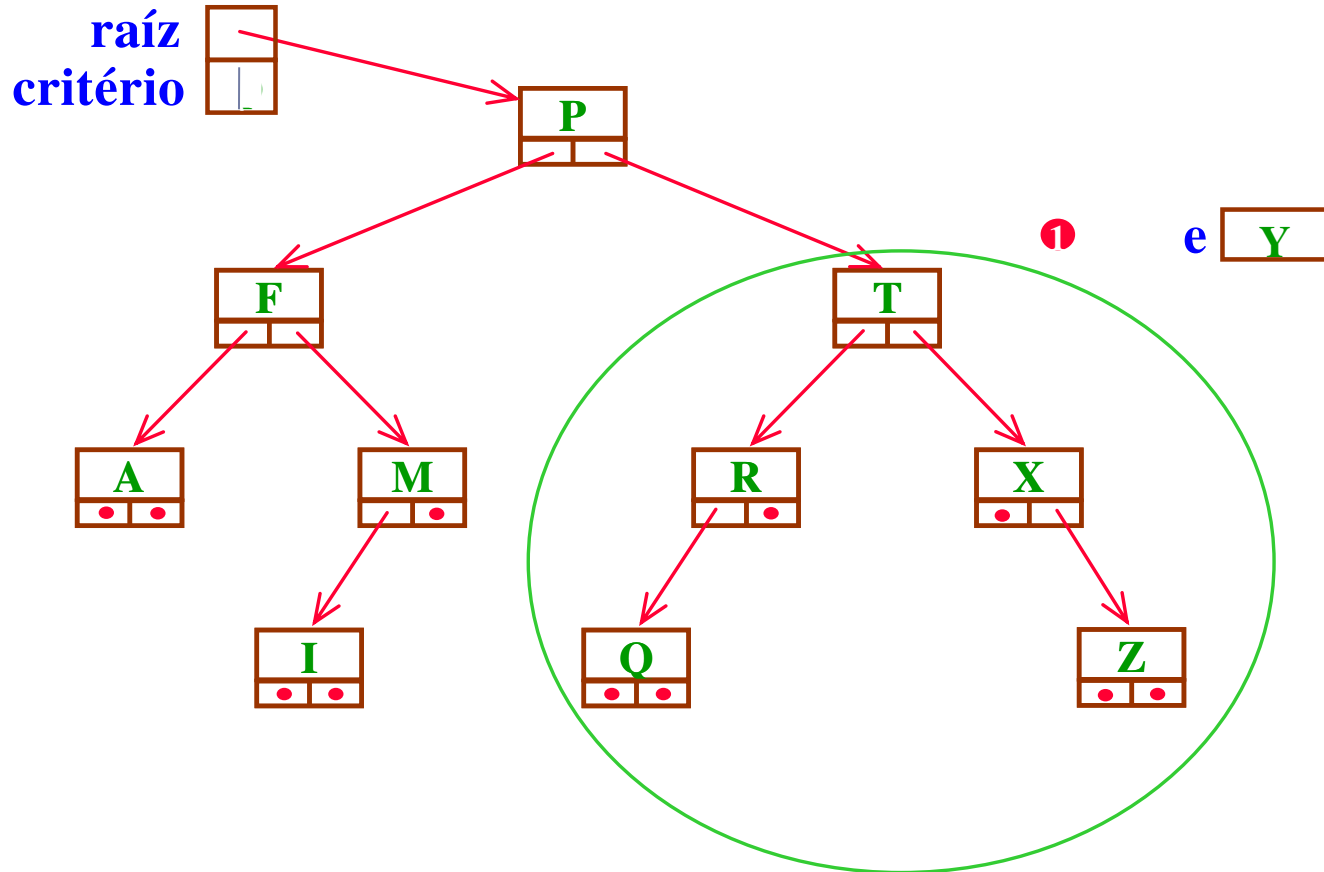
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



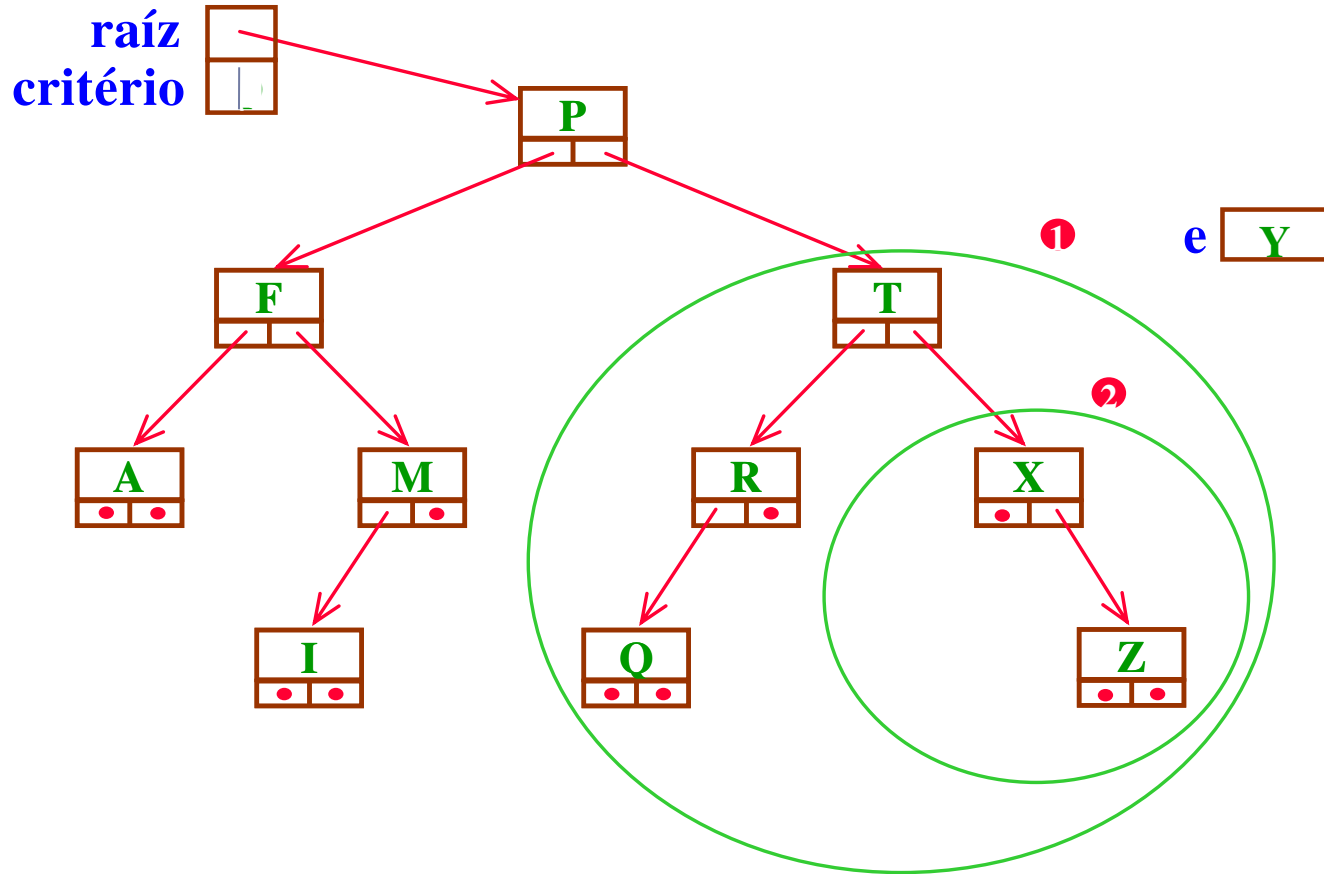
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



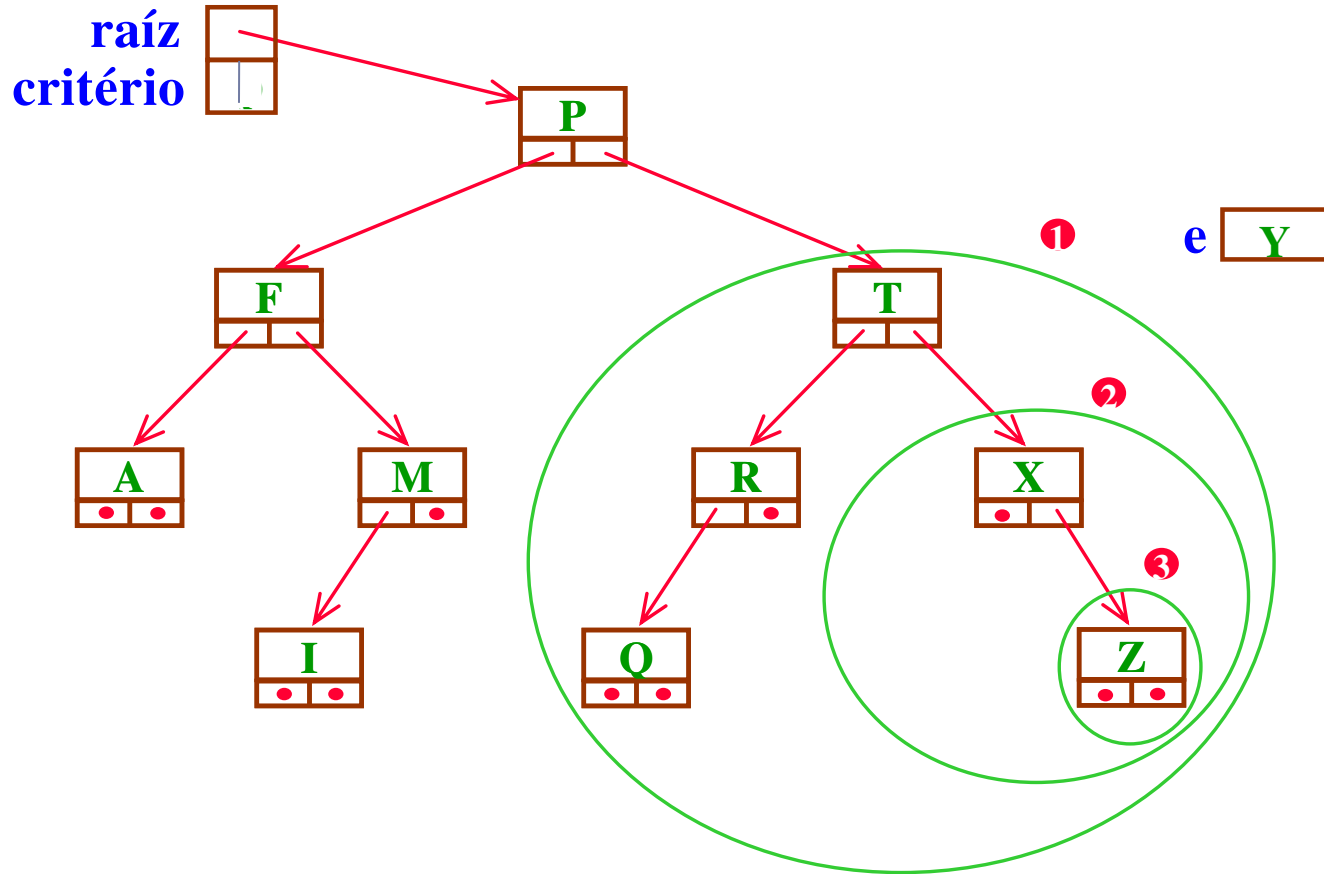
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



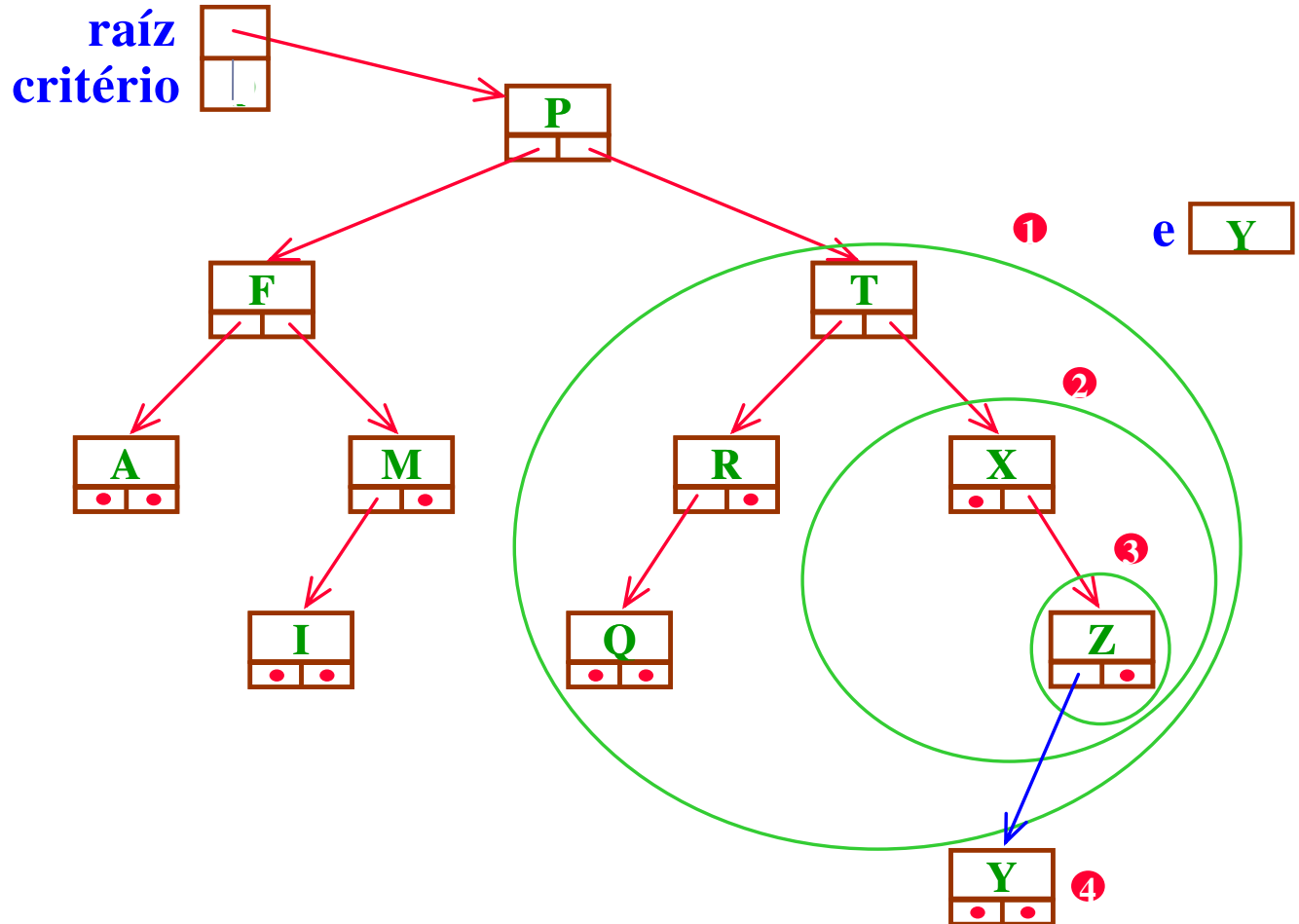
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



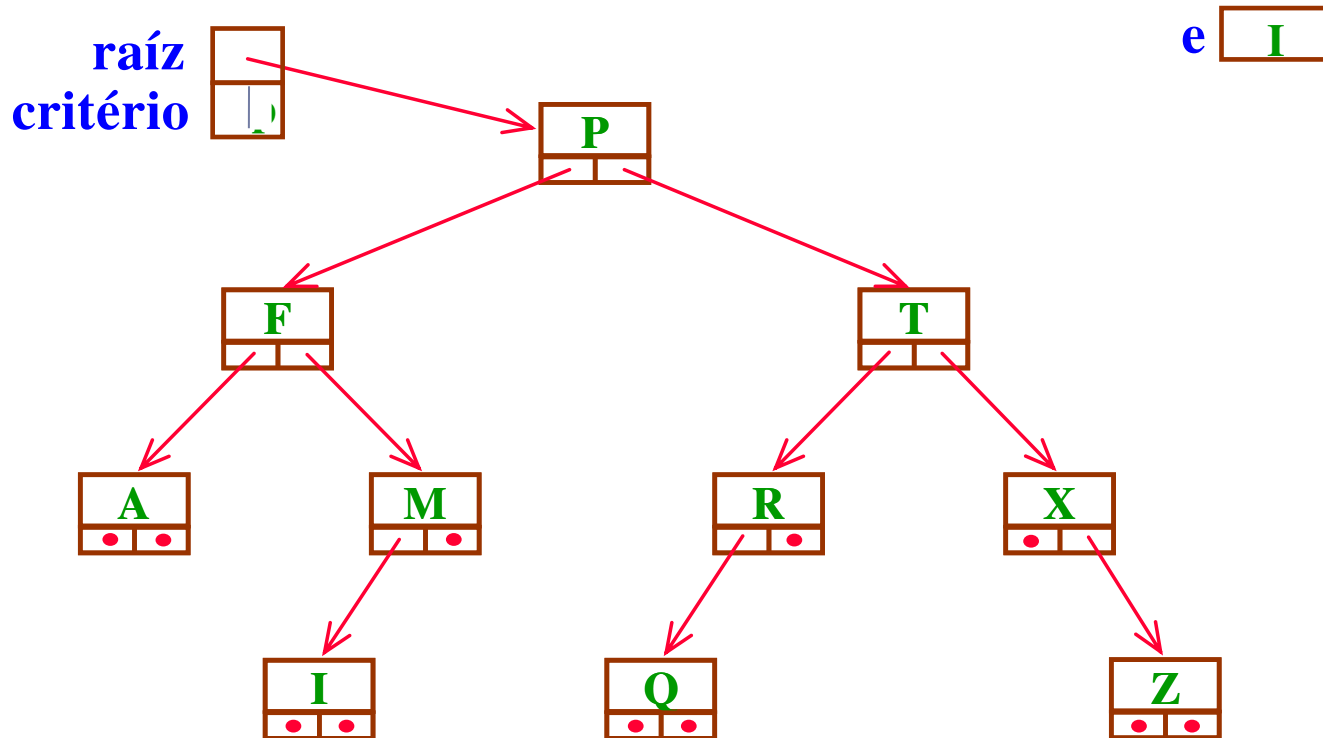
# Árvores Binárias de Pesquisa: Inserção

## ► Inserção



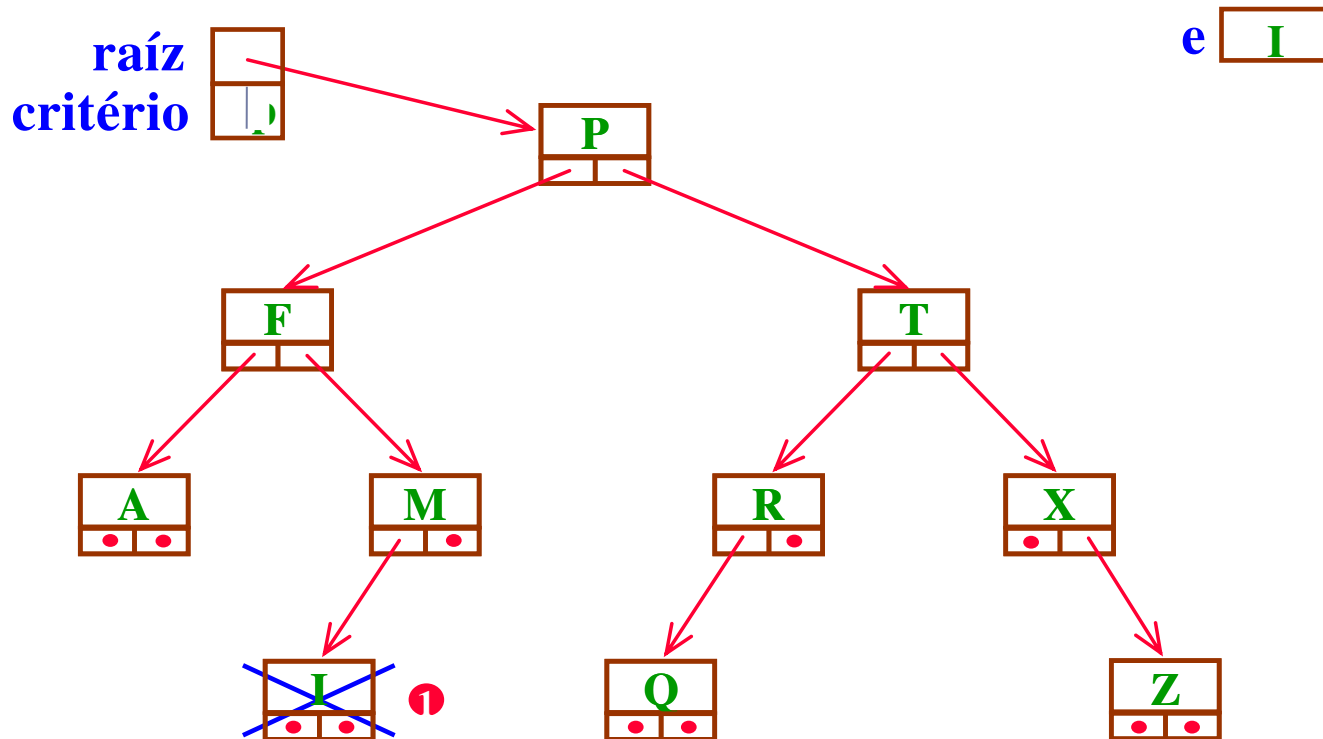
# Árvores Binárias de Pesquisa: Remoção

## ► Remoção de uma folha (nó sem filhos)



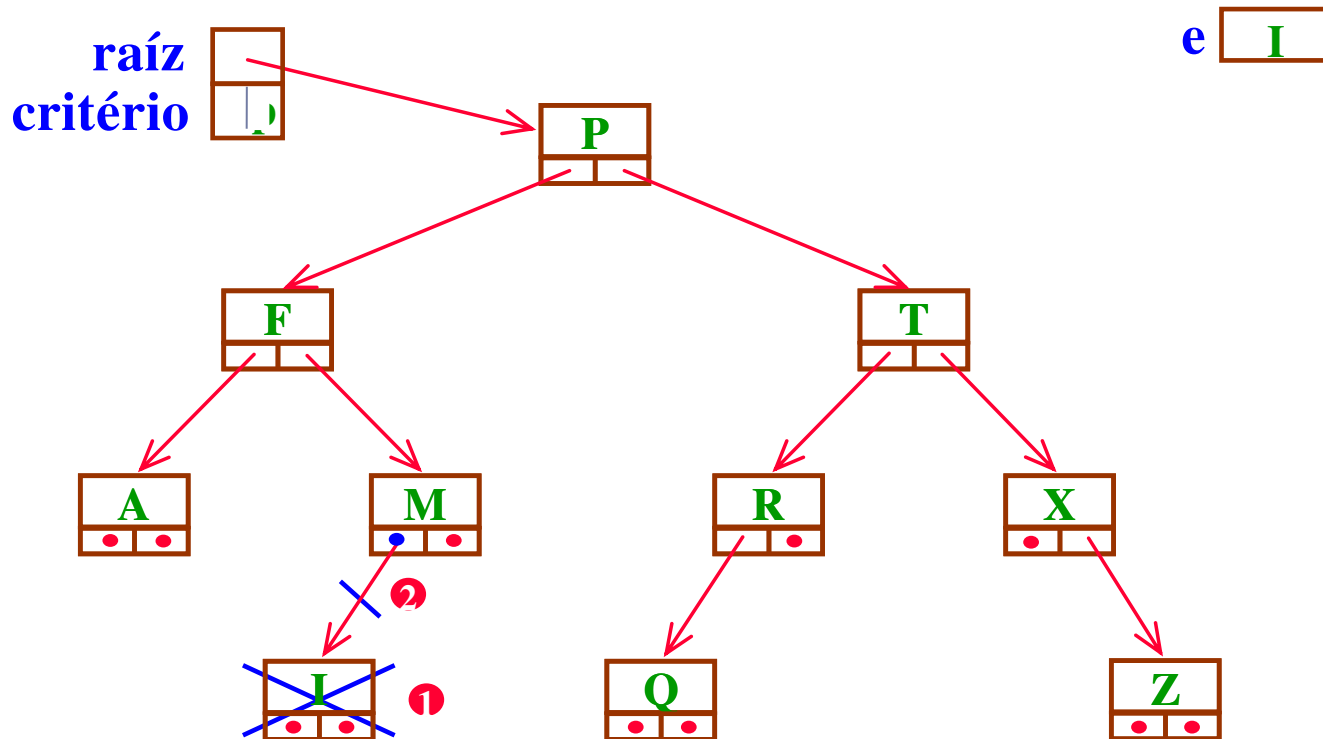
# Árvores Binárias de

## ► Remoção de uma folha (nó sem filhos)



# Árvores Binárias de

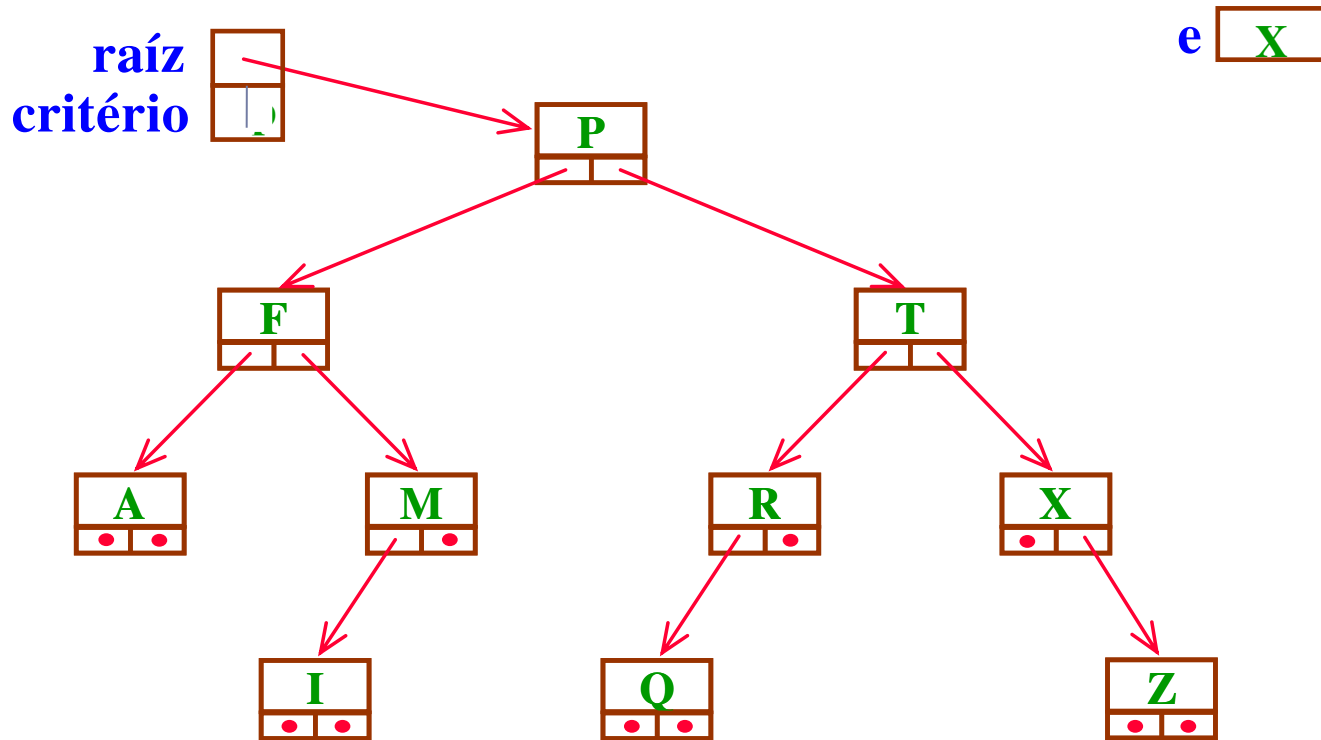
## ► Remoção de uma folha (nó sem filhos)





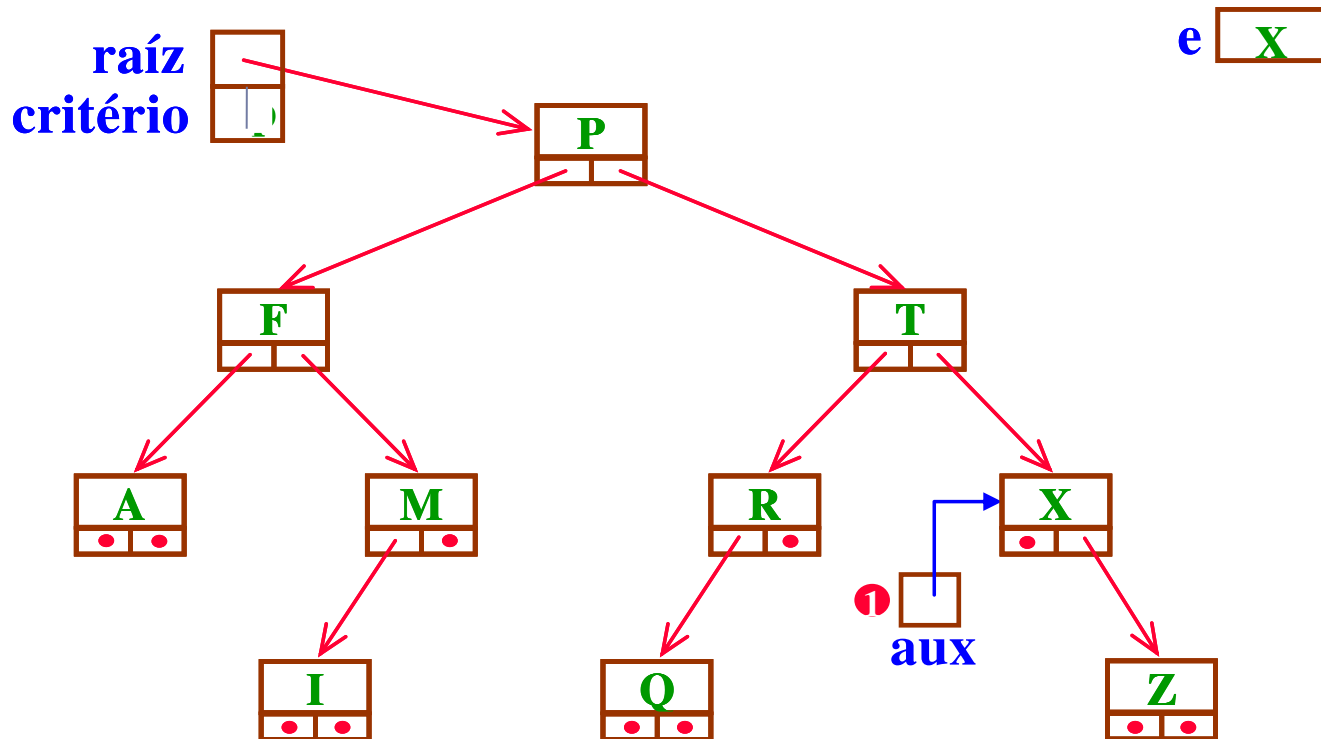
# Árvores Binárias de

## ► Remoção de nó só com filho direito



# Árvores Binárias de

## ► Remoção de nó só com filho direito

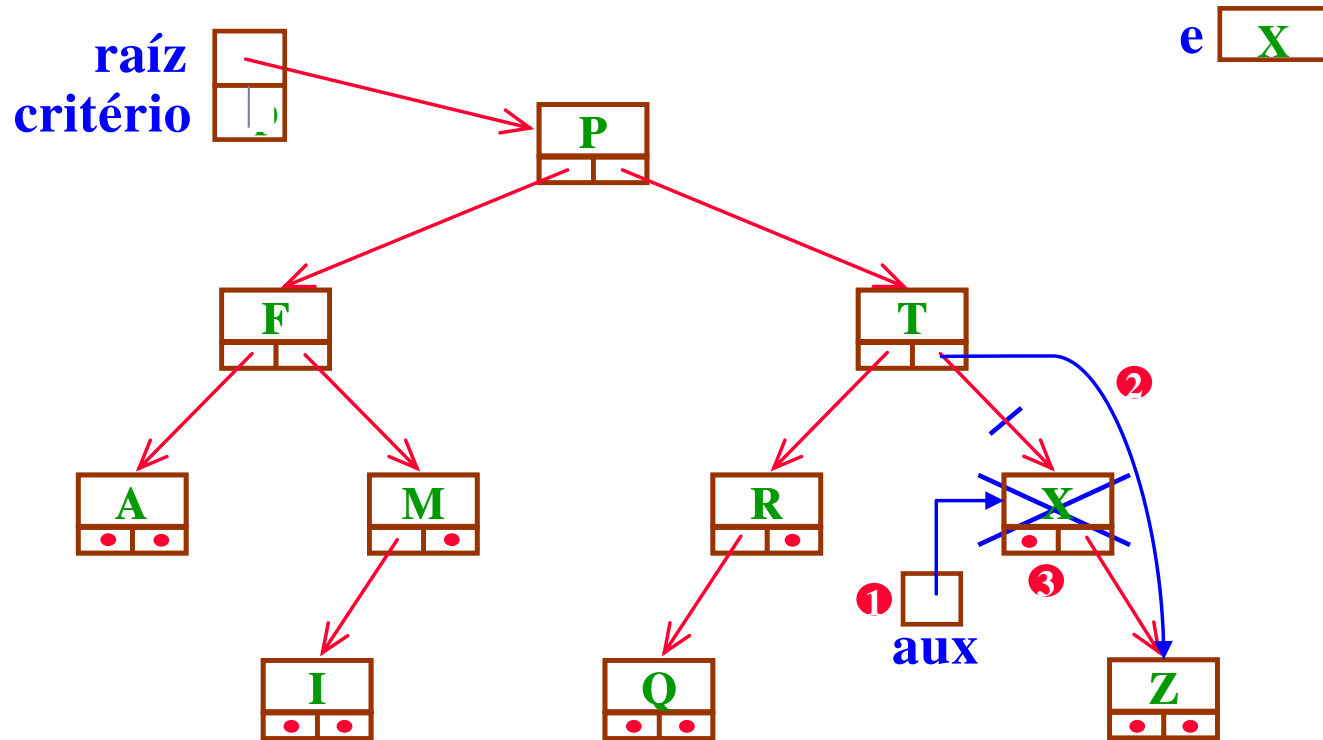


### ► Remoção de nó só com filho direito



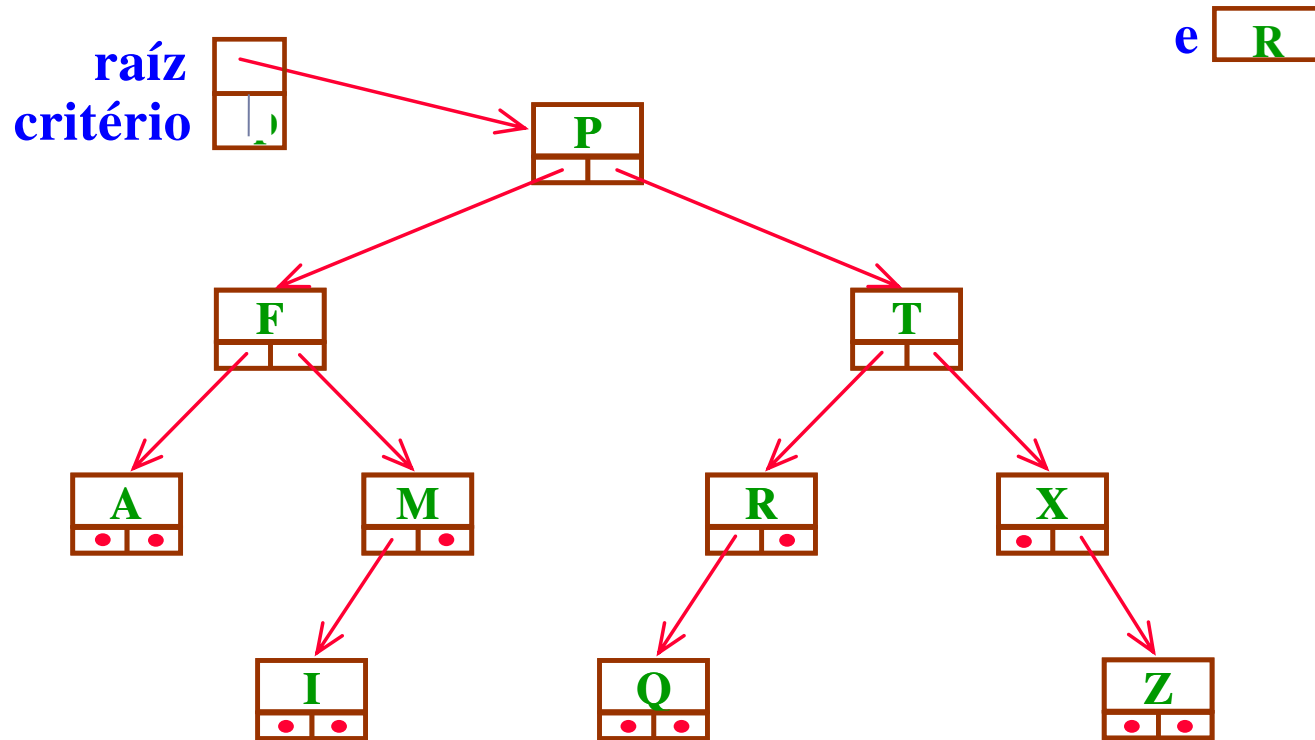
# Árvores Binárias de

## ► Remoção de nó só com filho direito



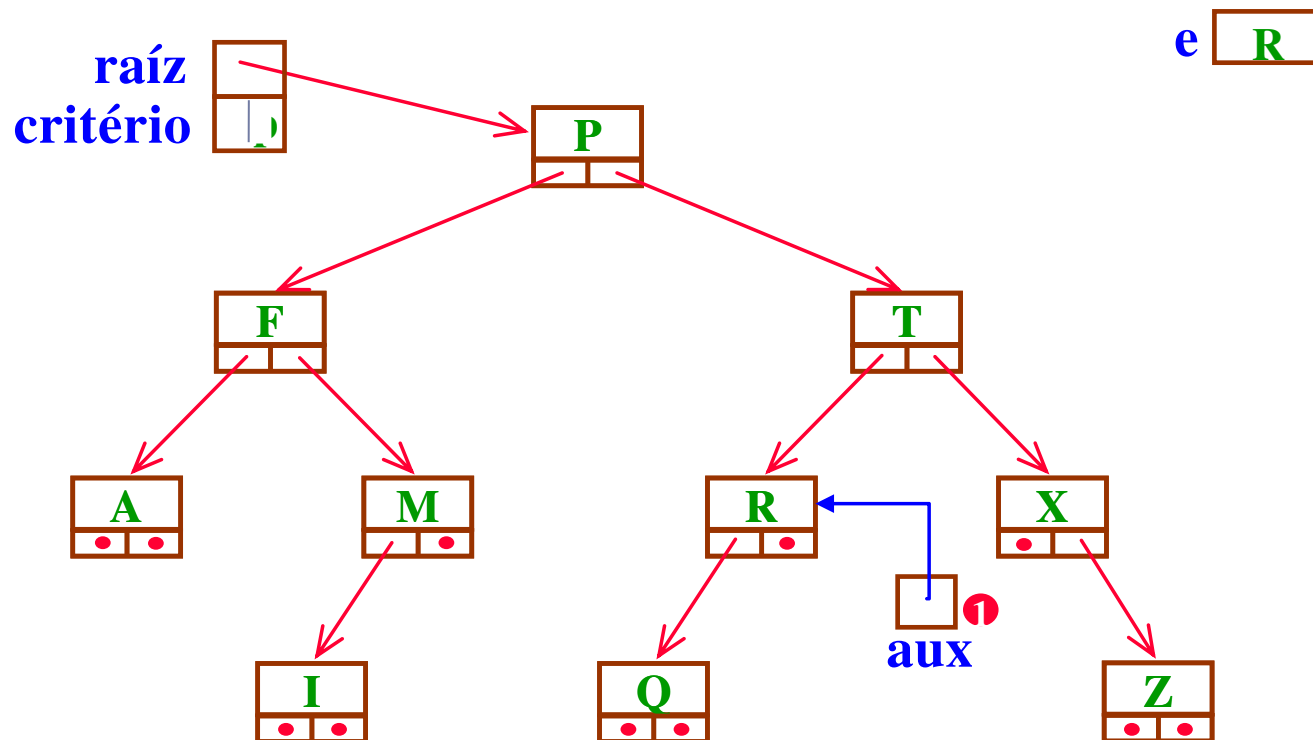
# Árvores Binárias de

## ► Remoção de nó só com filho esquerdo



# Árvores Binárias de

## ► Remoção de nó só com filho esquerdo

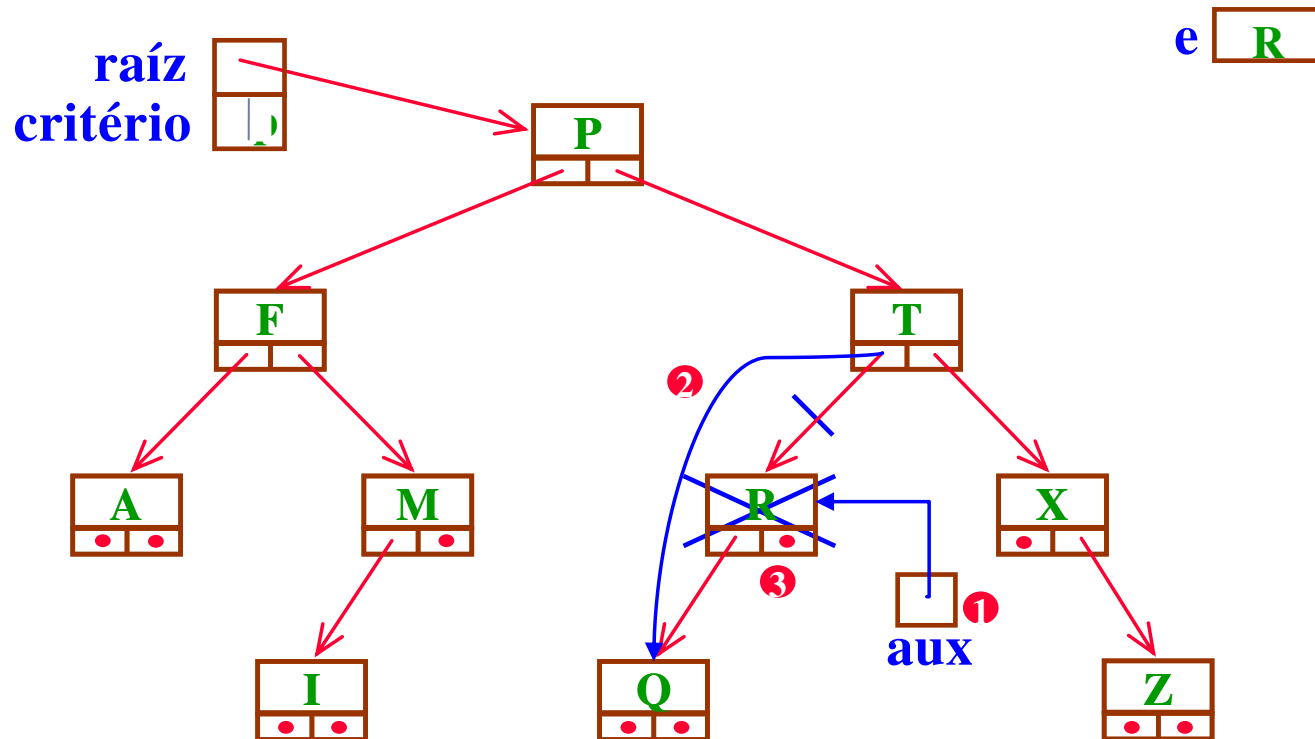


### ► Remoção de nó só com filho esquerdo



# Árvores Binárias de

## ► Remoção de nó só com filho esquerdo

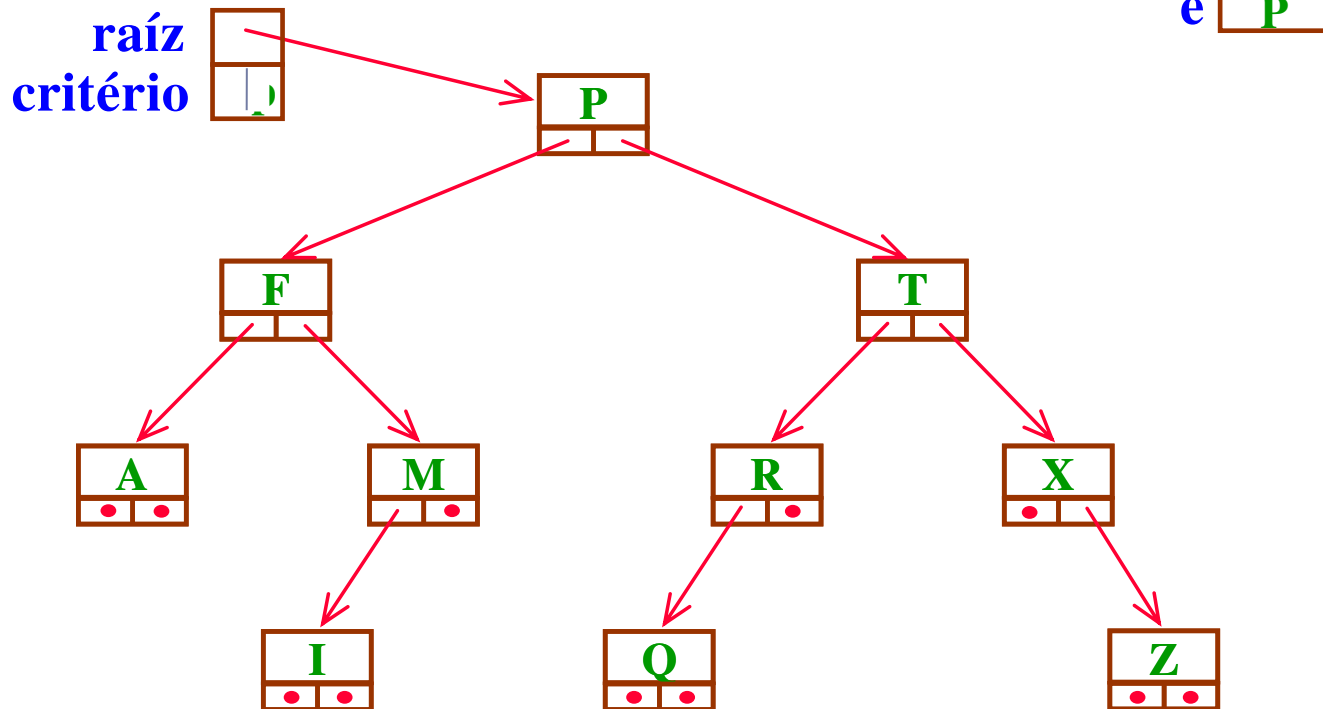




# Árvores Binárias de

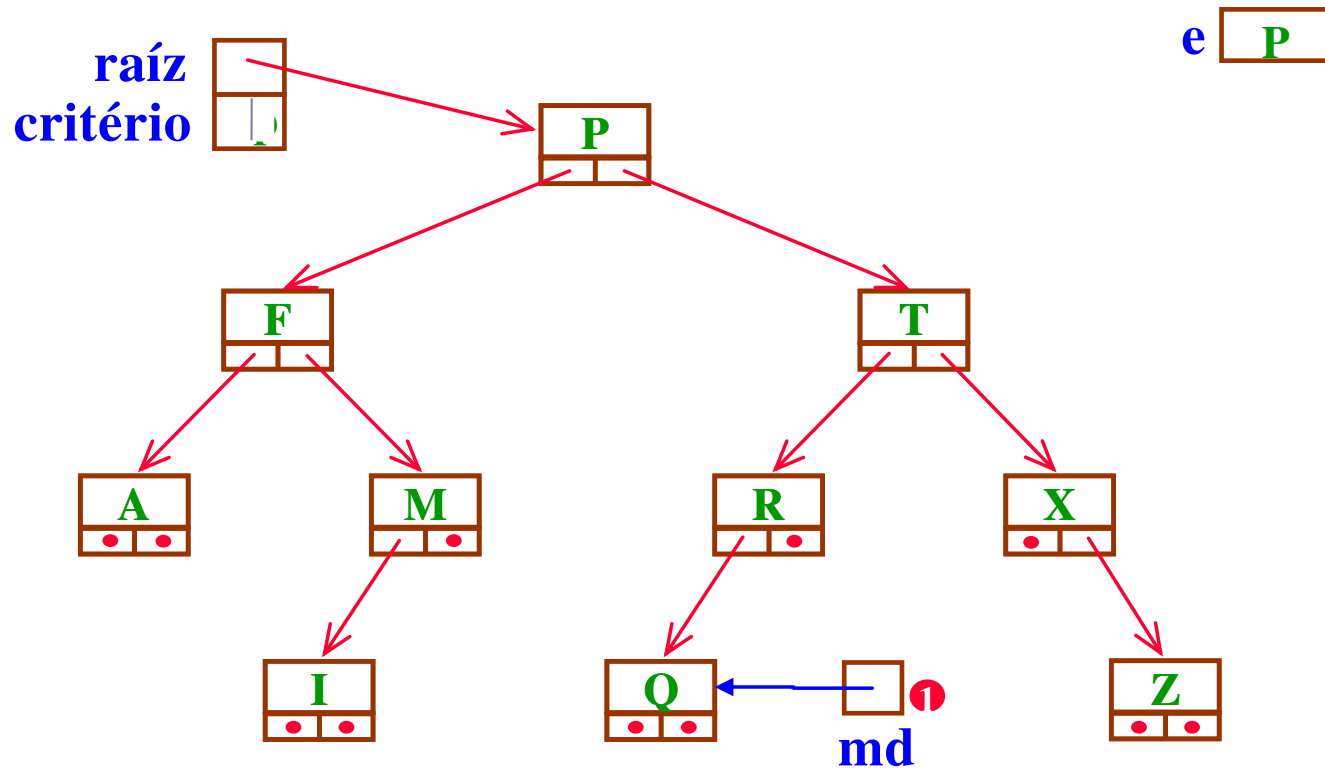
## ► Remoção de nó com dois filhos:

- Que chave é menor que todas da direita e maior que todas da esquerda??
- Substituir P por ela!



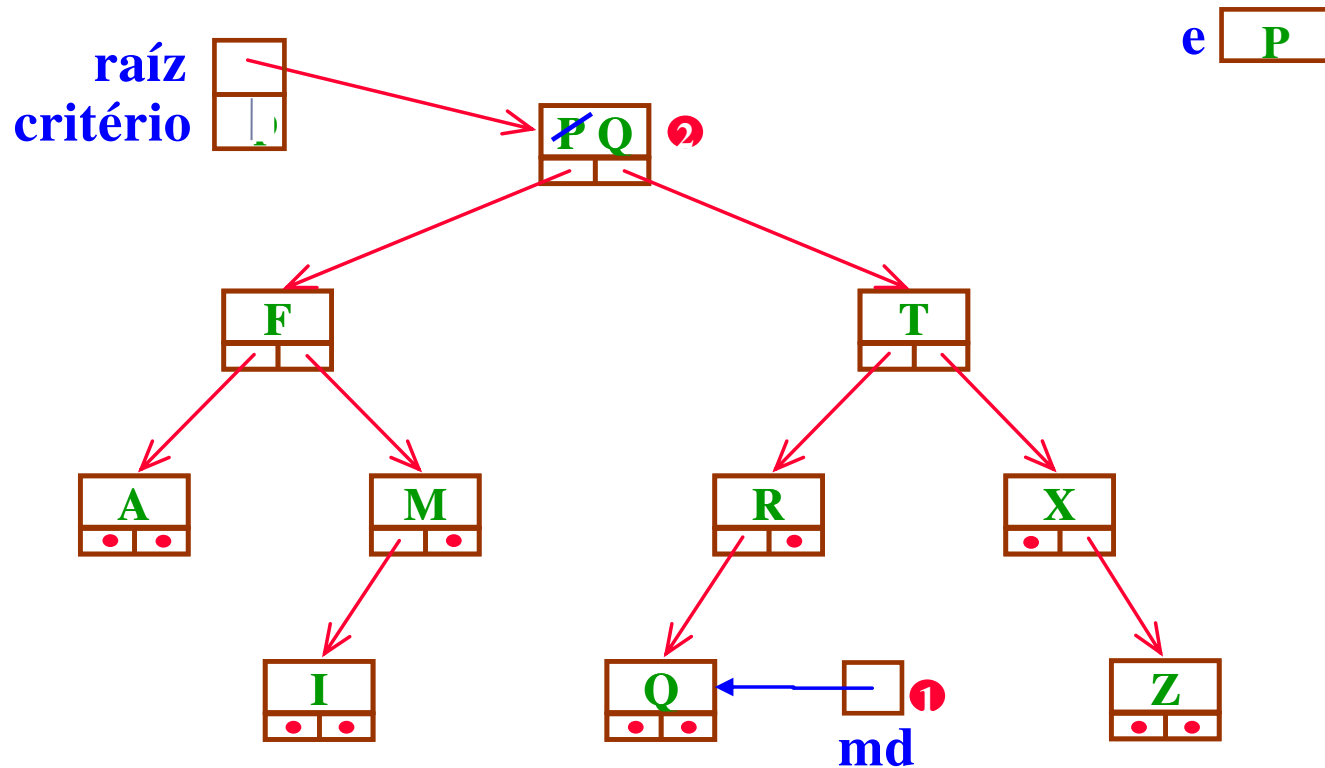
# Árvores Binárias de

## ► Remoção de nó com dois filhos



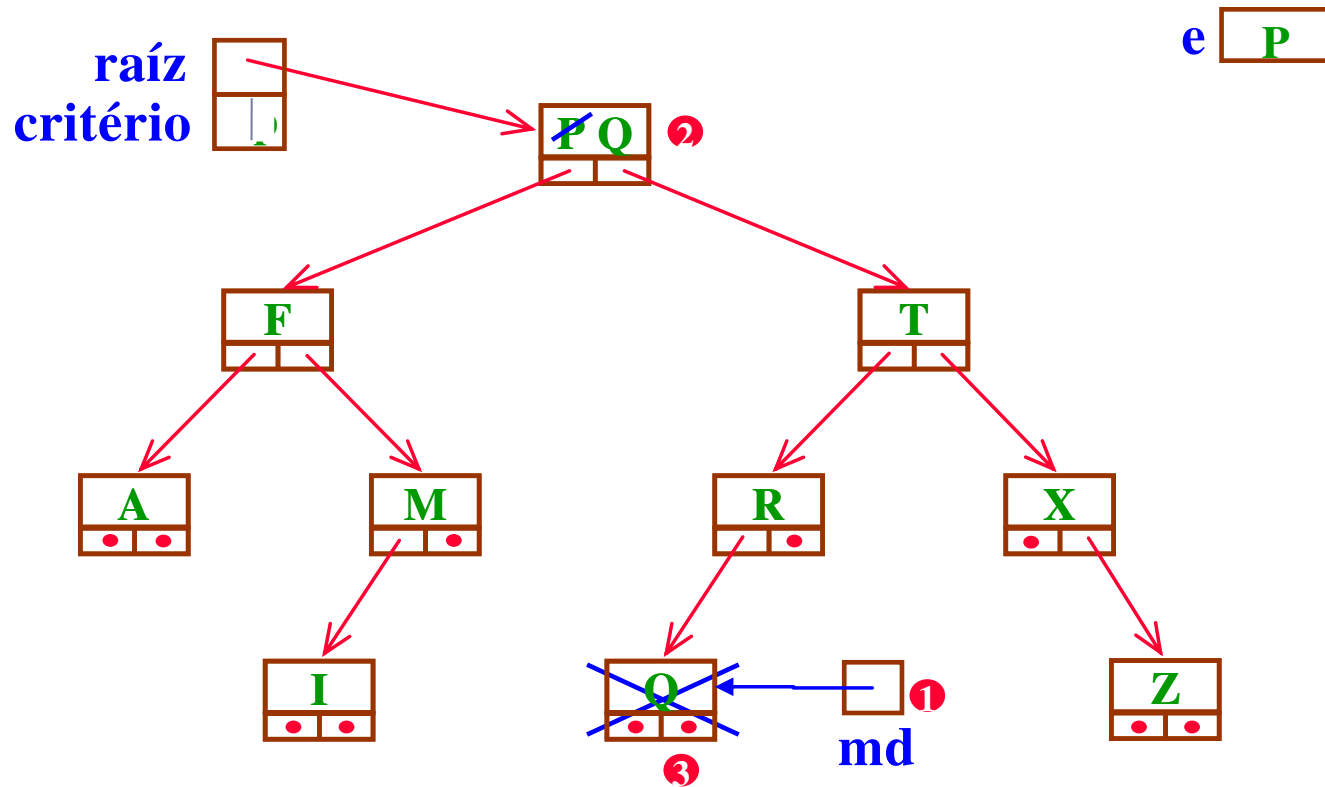
# Árvores Binárias de

## ► Remoção de nó com dois filhos



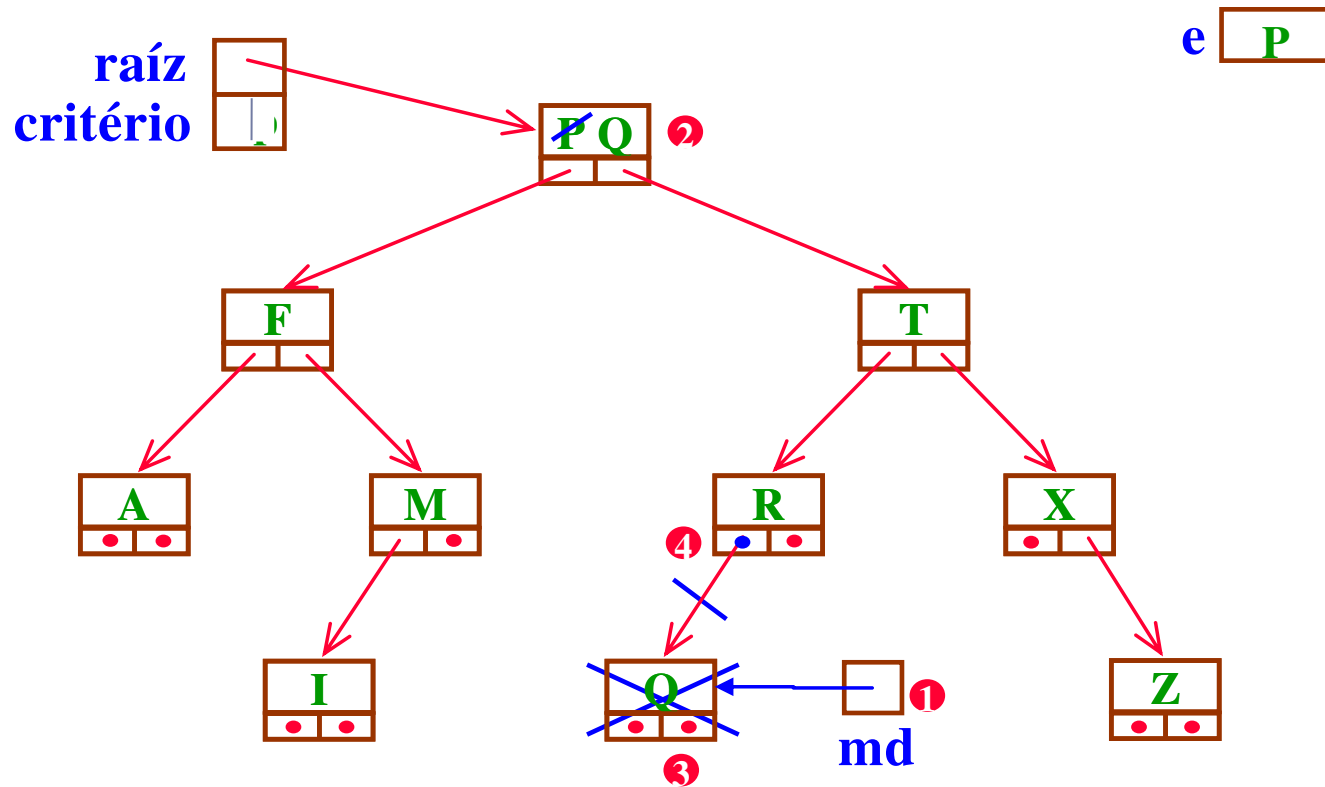
# Árvores Binárias de

## ► Remoção de nó com dois filhos



# Árvores Binárias de

## ► Remoção de nó com dois filhos



# Percurso em árvores binárias

---

- Acessar todos os elementos de uma estrutura de dados é uma importante operação
  - Eg. Calcular a média do campo idade de todos os nós, acrescentar um dígito ao valor do campo “celular” de todos os nós. etc
- Árvores binárias (de pesquisa ou não) permitem variadas formas de acessar todos seus elementos
- Tais operações são chamadas de percurso, travessia, varredura.
- Para cada tipo, obtém-se todos os nós numa sequência diferente
  - Útil para ajudar a verificar propriedades da árvore

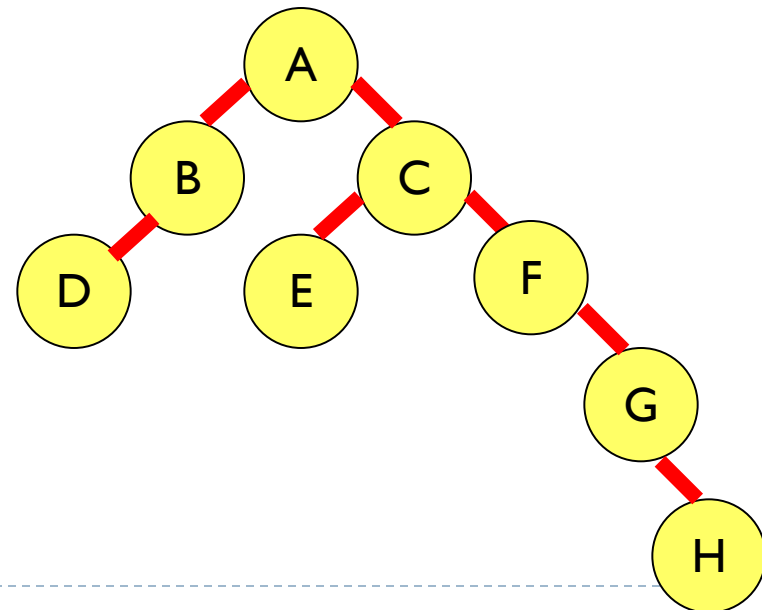


Exercício: Qual o resultado da operação abaixo?

---

- **Percurso Pré-ordem**

- 1- **Visita (i.e. Acessa/imprime)** a raiz da árvore corrente;
- 2- Percorre a subárvore esquerda em pré-ordem;
- 3- Percorre a subárvore direita em pré-ordem;



---

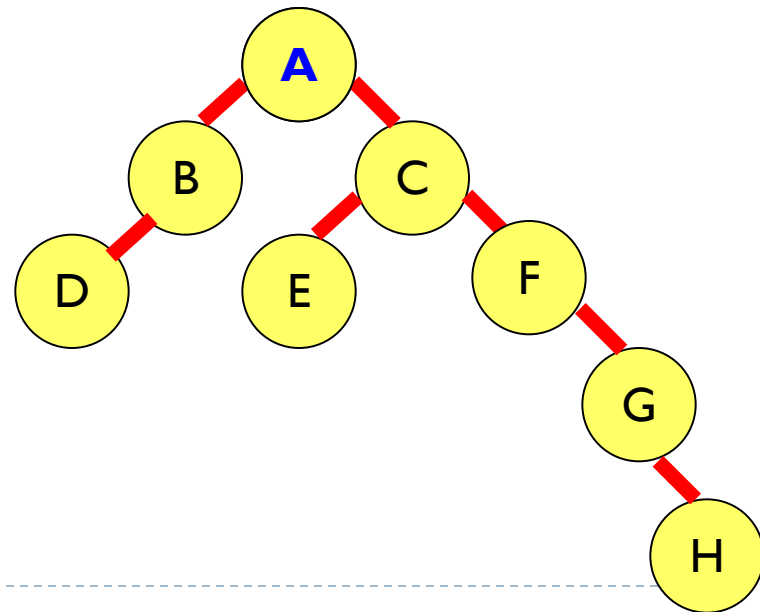
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A**





---

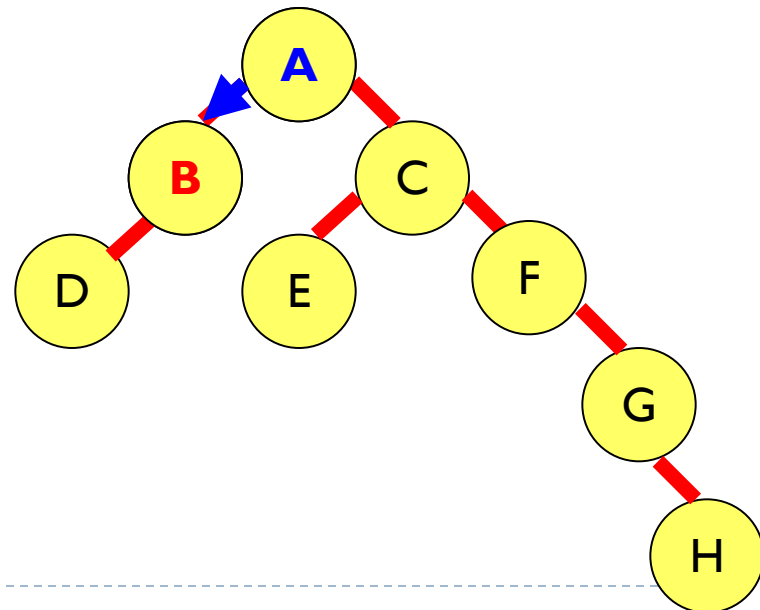
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A**



---

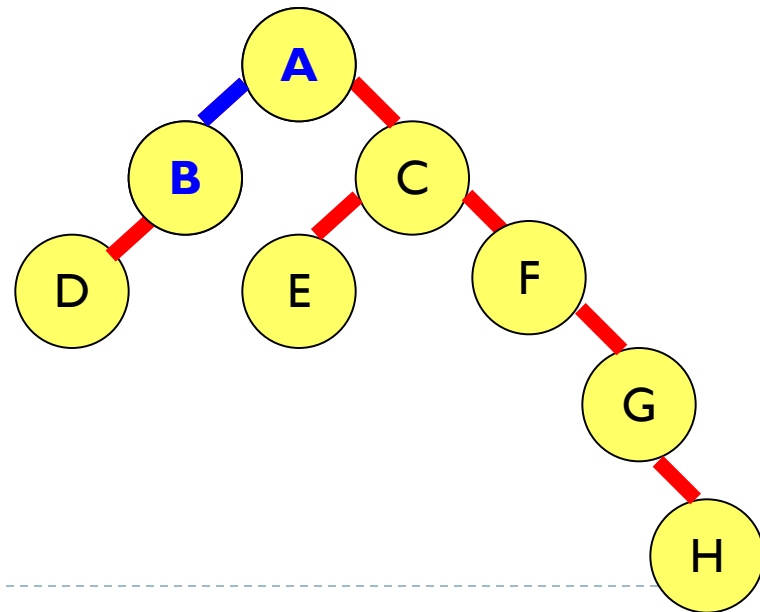
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B**



---

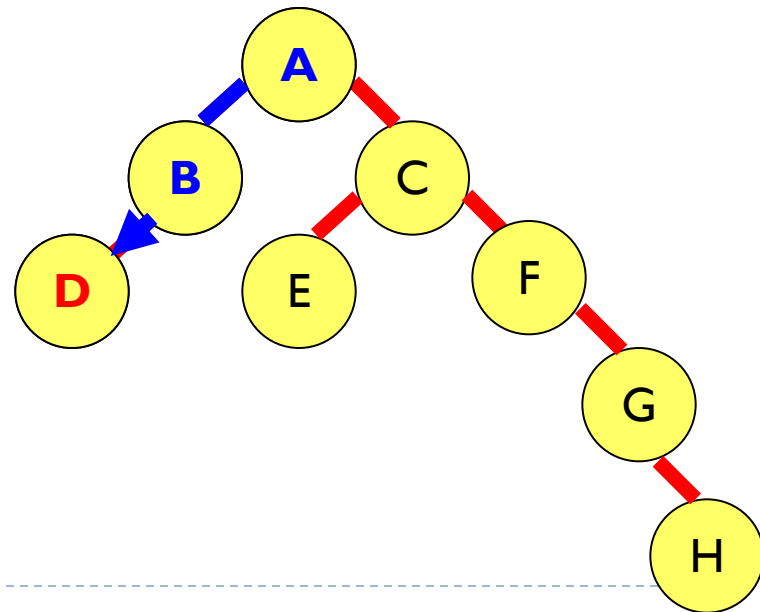
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B**



---

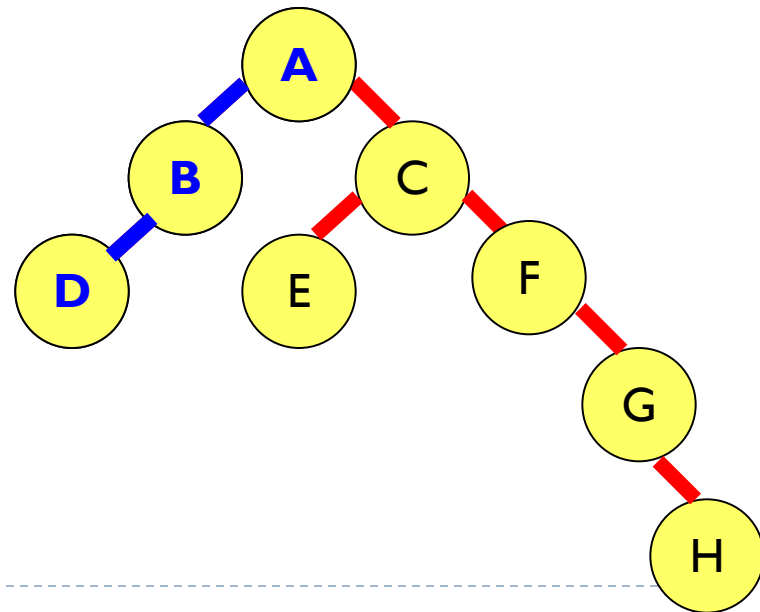
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**



---

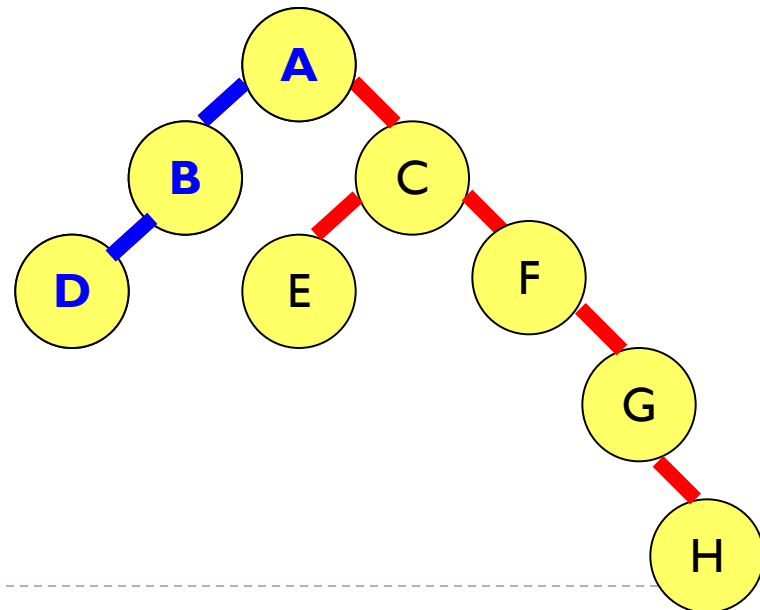
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**



---

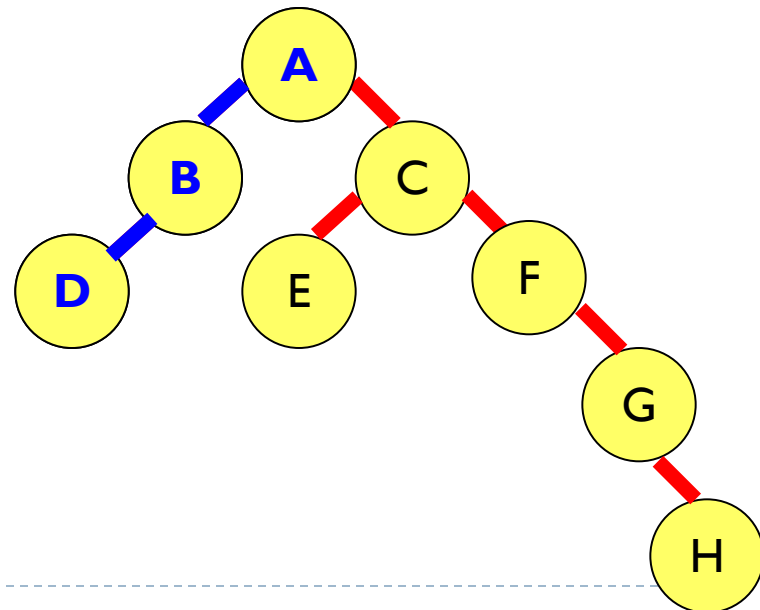
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**



---

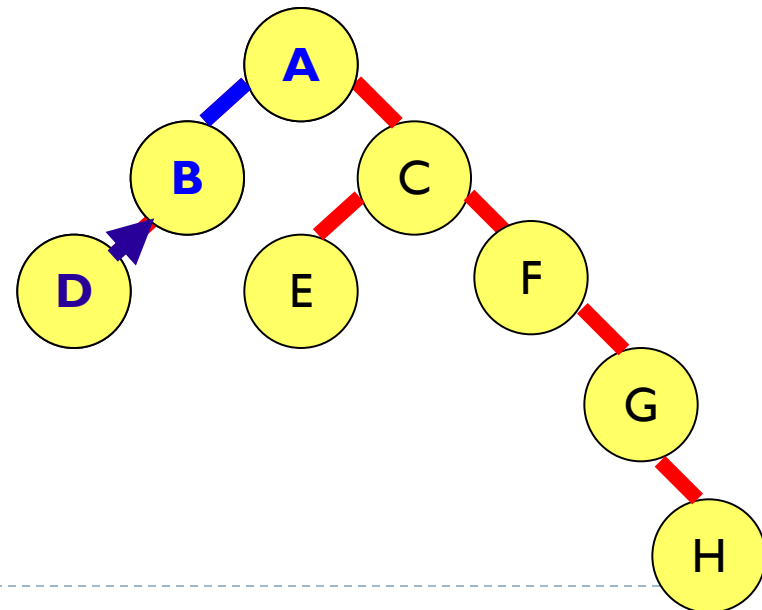
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**



---

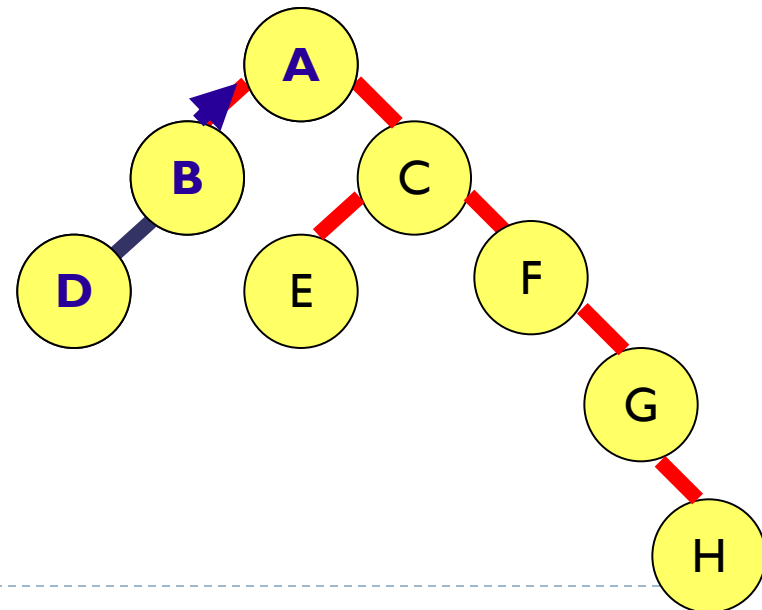
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**





---

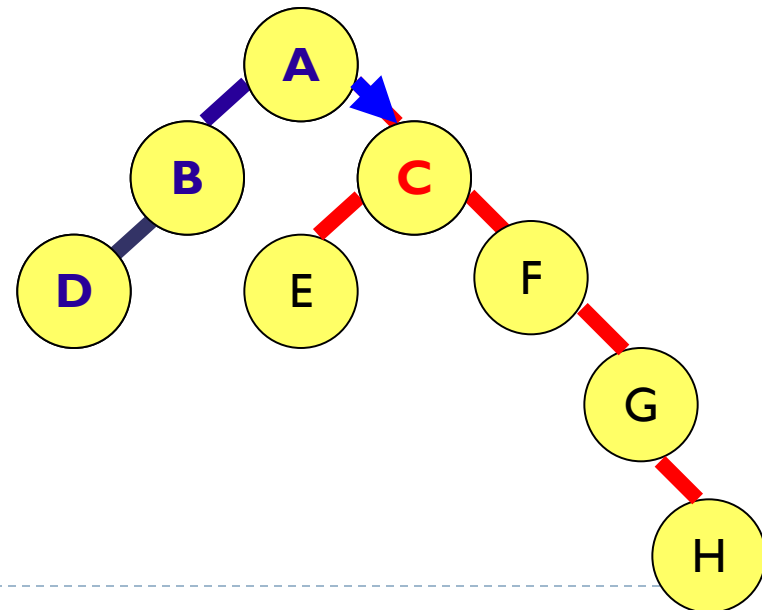
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D**



---

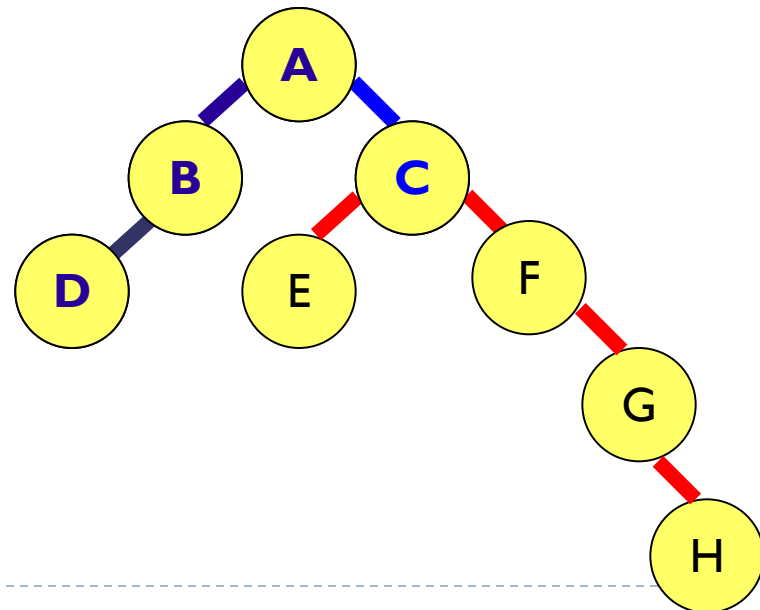
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C**



---

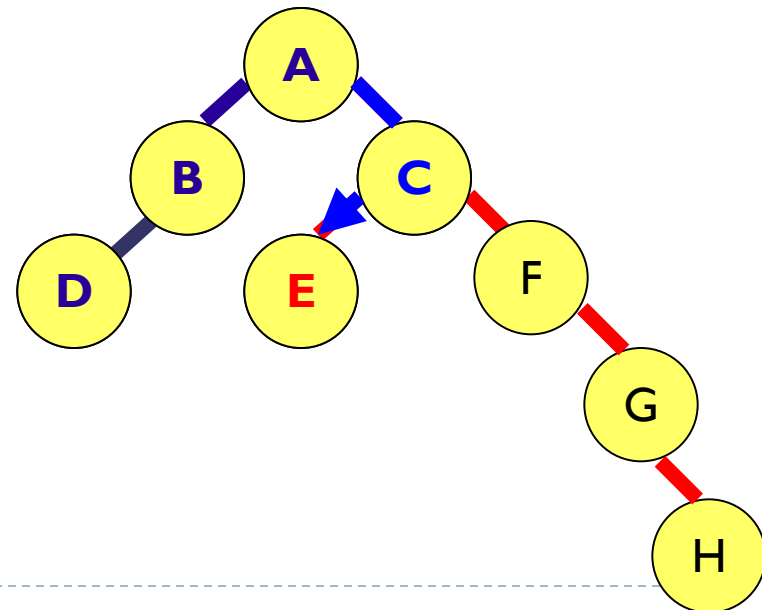
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C**



---

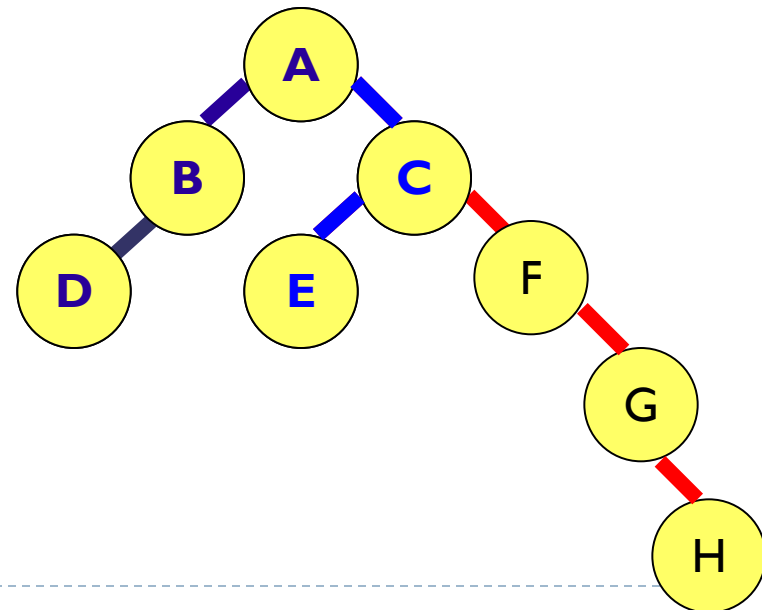
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E**



---

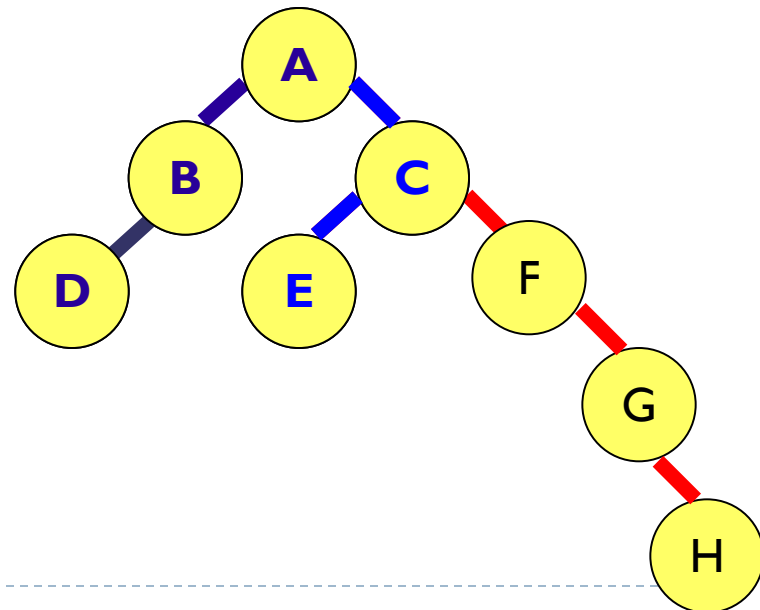
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E**



---

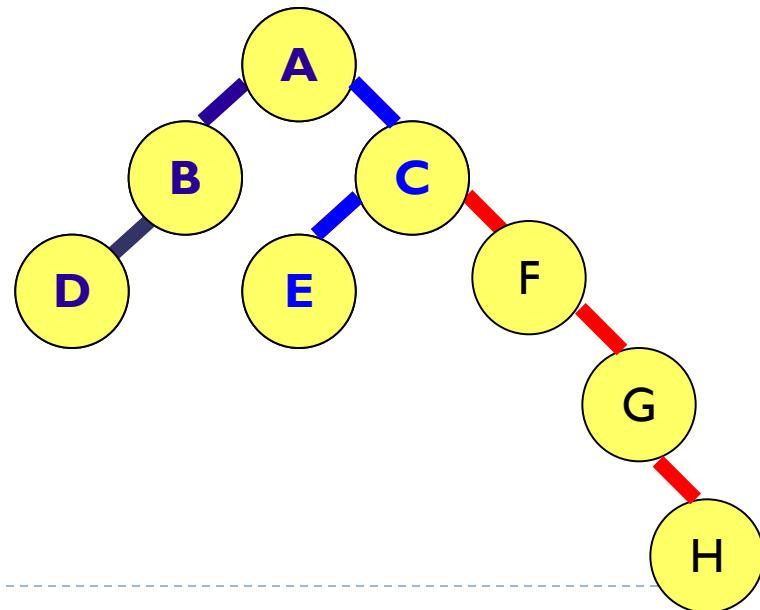
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E**



---

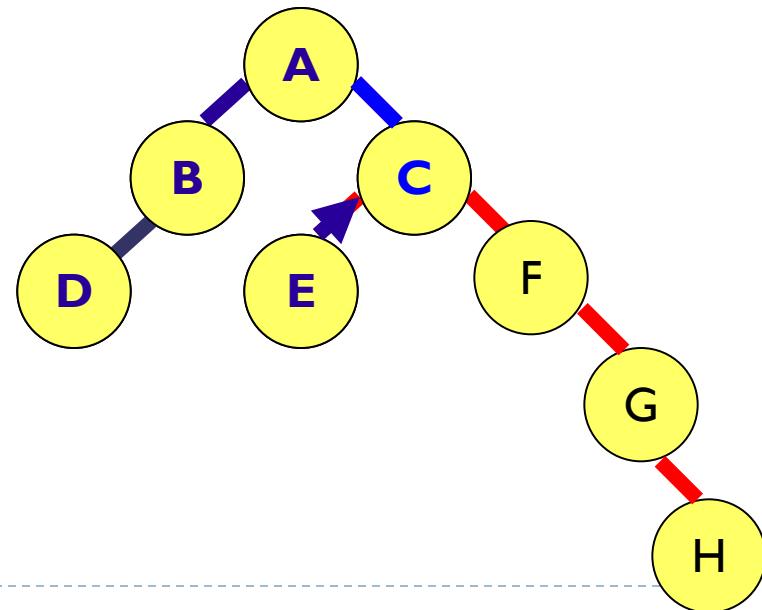
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E**



---

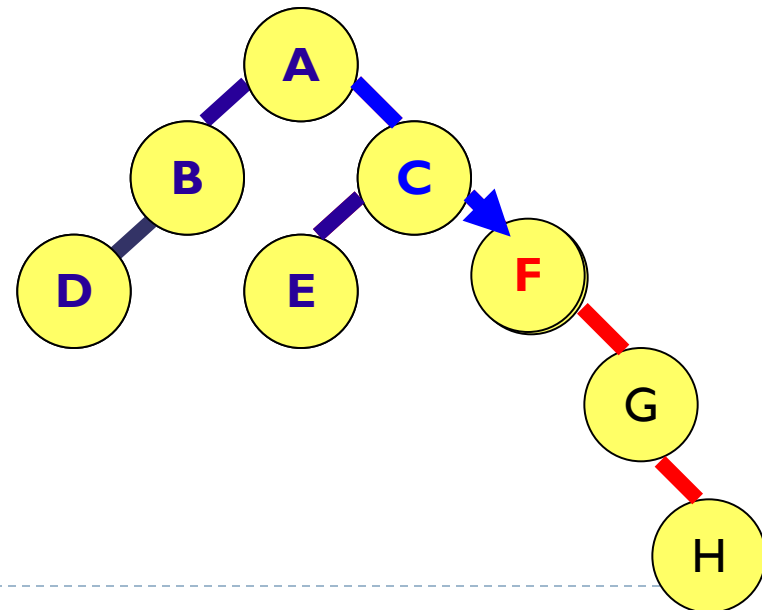
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E**





---

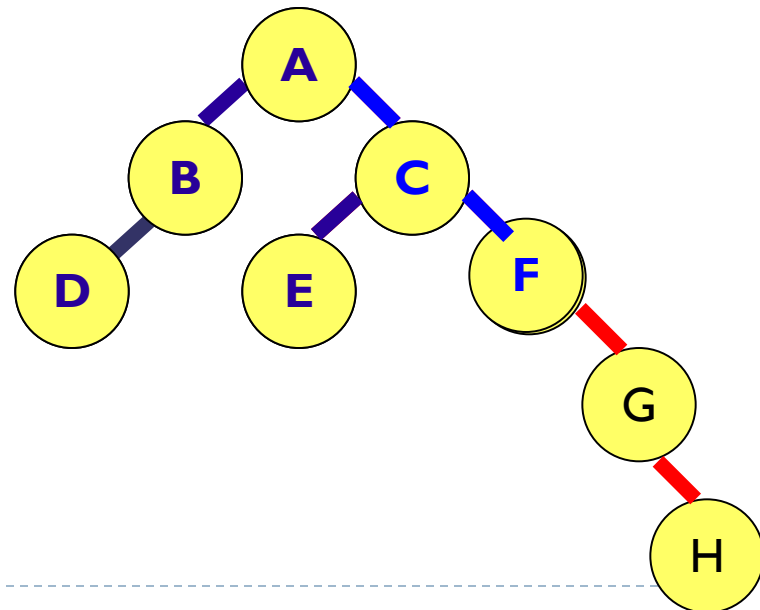
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F**



---

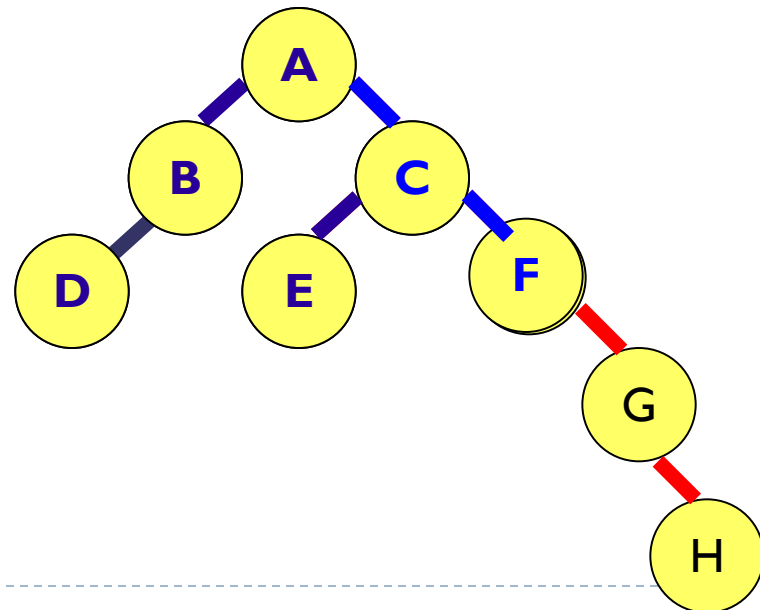
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F**



---

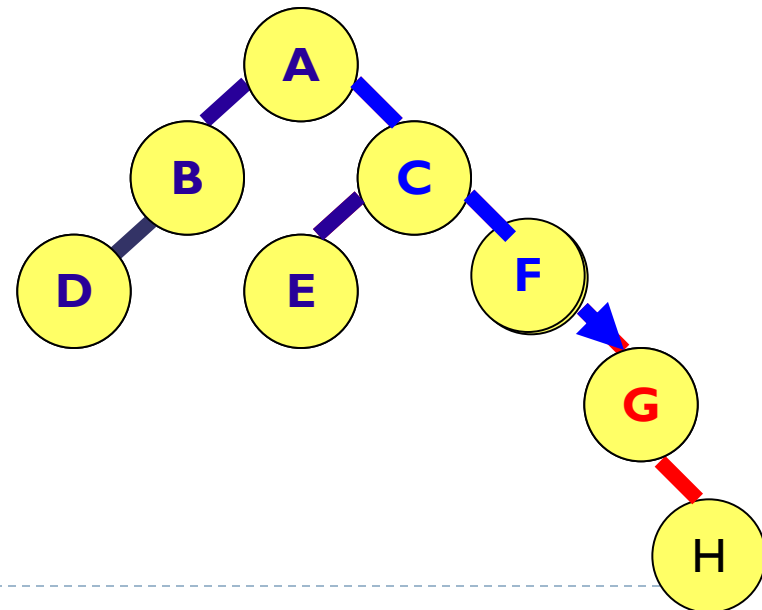
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F**



---

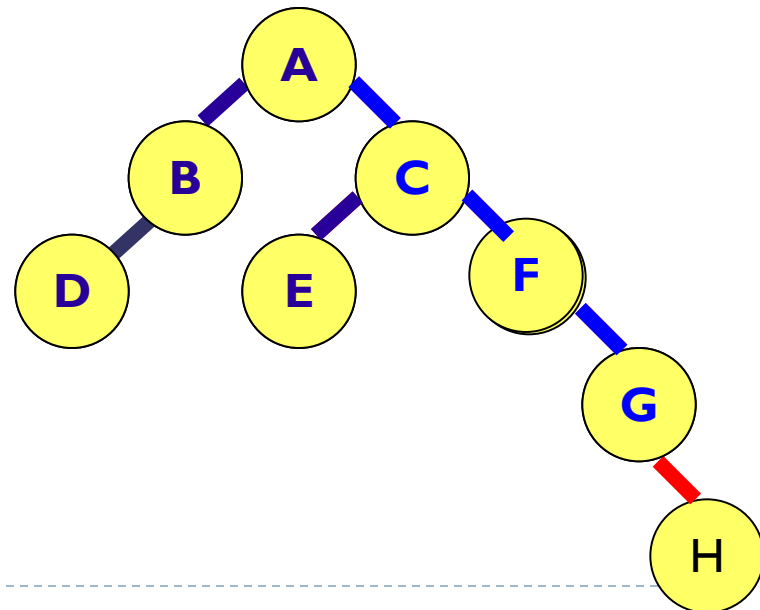
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G**



---

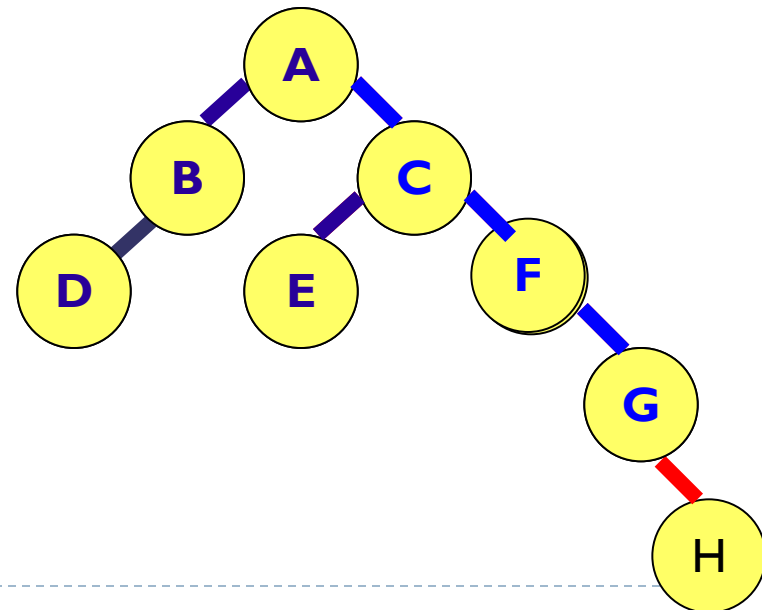
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G**



---

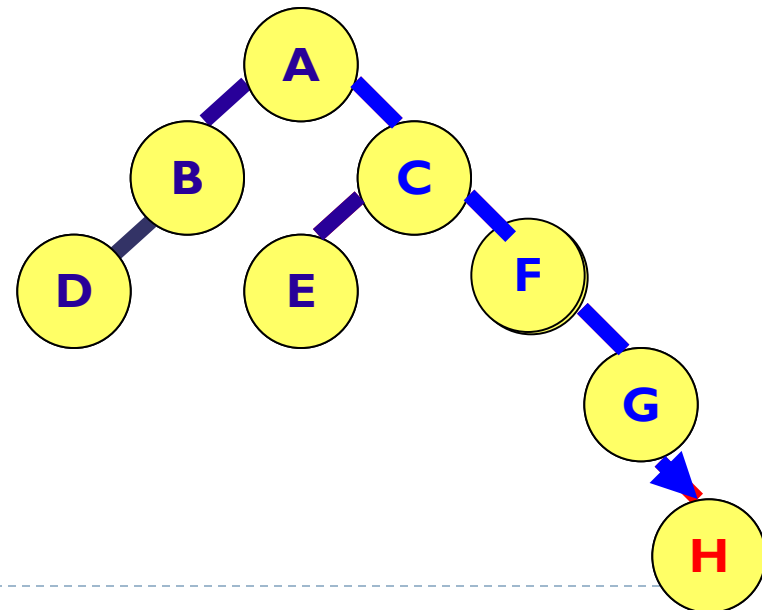
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G**



---

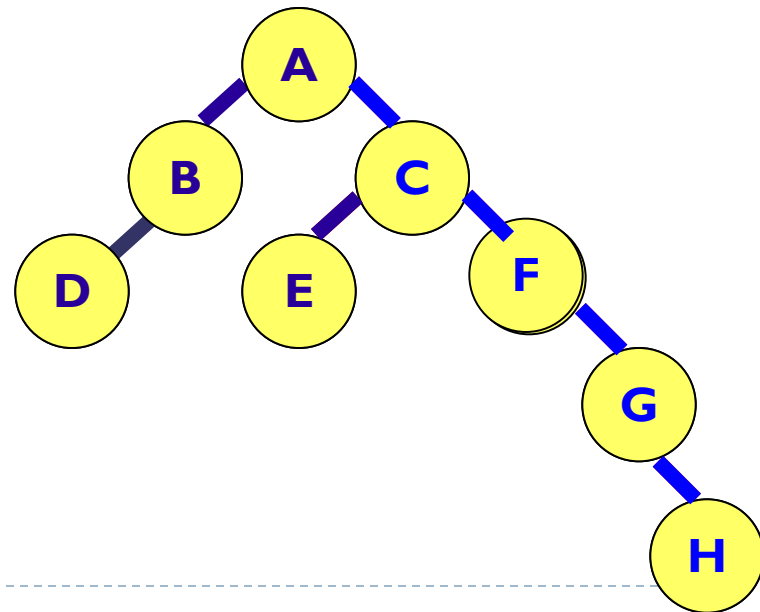
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G H**



---

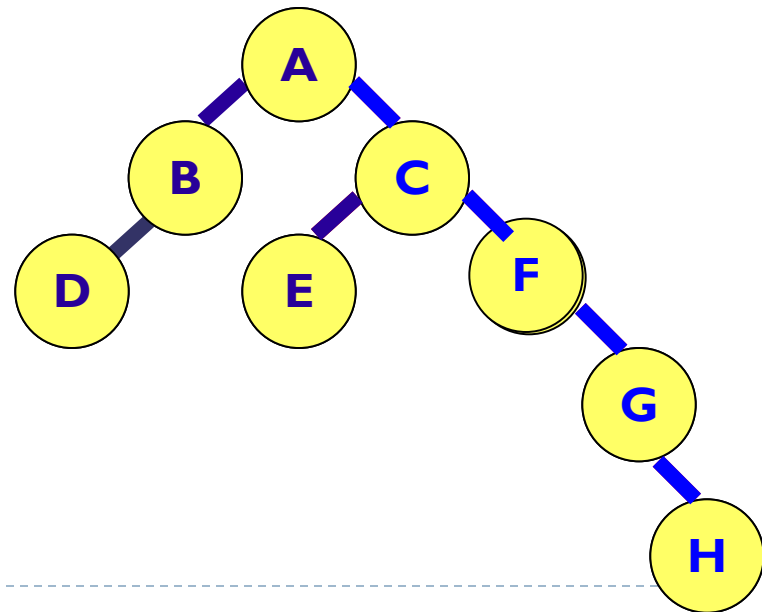
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G H**





---

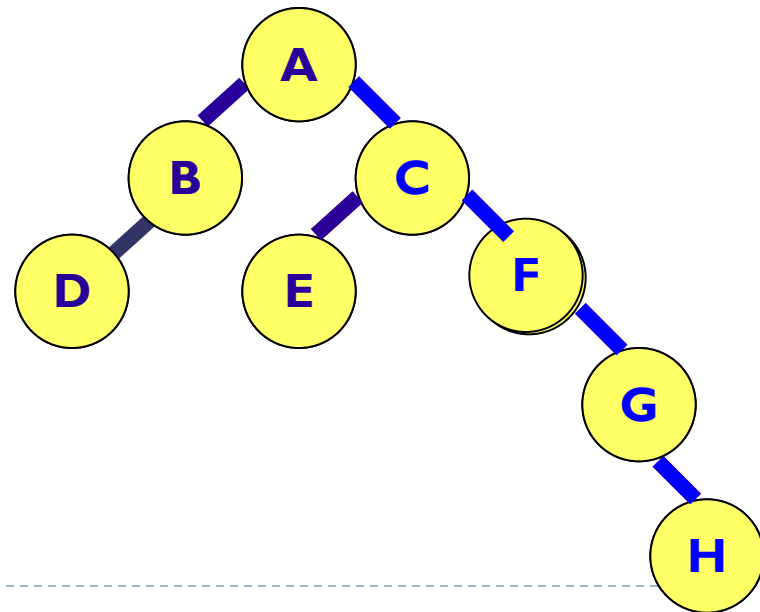
- **Percurso Pré-ordem**

1- Visita a raiz;

2- Percorre a subárvore esquerda em pré-ordem;

3- Percorre a subárvore direita em pré-ordem;

Resultado: **A B D C E F G H**



---

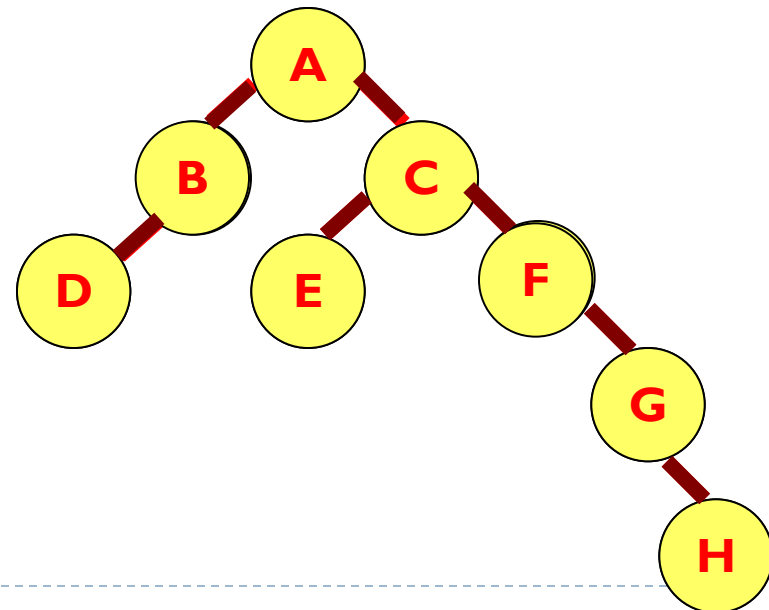
- **Percurso In-ordem**

1- Percorre a subárvore esquerda em in-ordem;

2- Visita a raiz;

3- Percorre a subárvore direita em in-ordem;

Resultado: **D B A E C F G H**



---

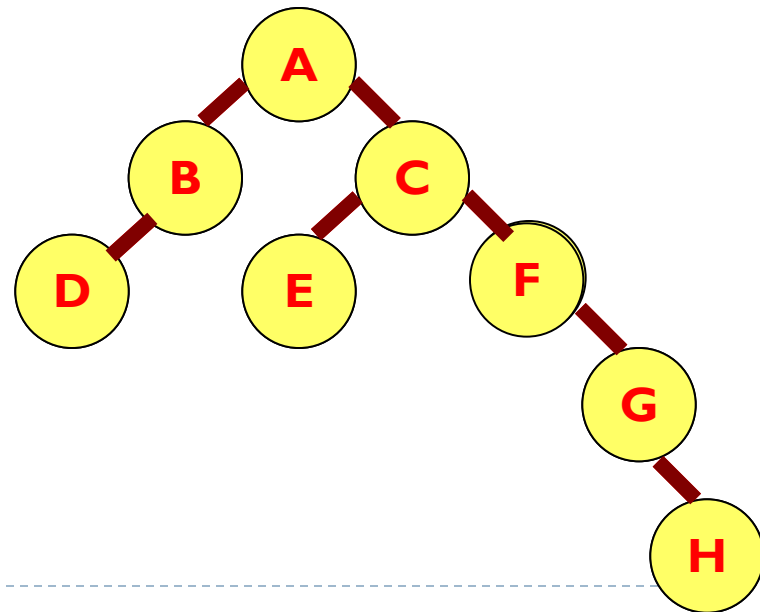
- **Percurso Pós-ordem**

1- Percorre a subárvore esquerda em pós-ordem;

2- Percorre a subárvore direita em pós-ordem;

3- Visita a raiz;

Resultado: **D B E H G F C A**



# Pre-ordem

---

// Recebe a raiz r de uma árvore binária.  
// Caminha raiz-esq-dir.

```
void preOrdem( TipoBinaria *r)
{
    if (r != NULL)
    {
        printf( "%d\n", r->chave);
        preOrdem( r->esq);
        preOrdem( r->dir);
    }
}
```



# Em-ordem

---

// Recebe a raiz r de uma árvore binária.  
// Caminha esq-raiz-dir.

```
void emOrdem(TipoBinaria *r)
{
    if (r != NULL)
    {
        emOrdem( r->esq);
        printf( "%d\n", r->chave);
        emOrdem( r->dir);
    }
}
```



# Pós-ordem

---

```
// Recebe a raiz r de uma árvore binária.  
// Caminha esq-dir-raiz.
```

```
void posOrdem( TipoBinaria *r)  
{  
    if (r != NULL)  
    {  
        posOrdem( r->esq);  
        posOrdem( r->dir);  
        printf( "%d\n", r->chave);  
    }  
}
```

