

ED62A-COM2A

ESTRUTURAS DE DADOS

Aula 06 - Árvores Binárias

Prof. Rafael G. Mantovani

Roteiro

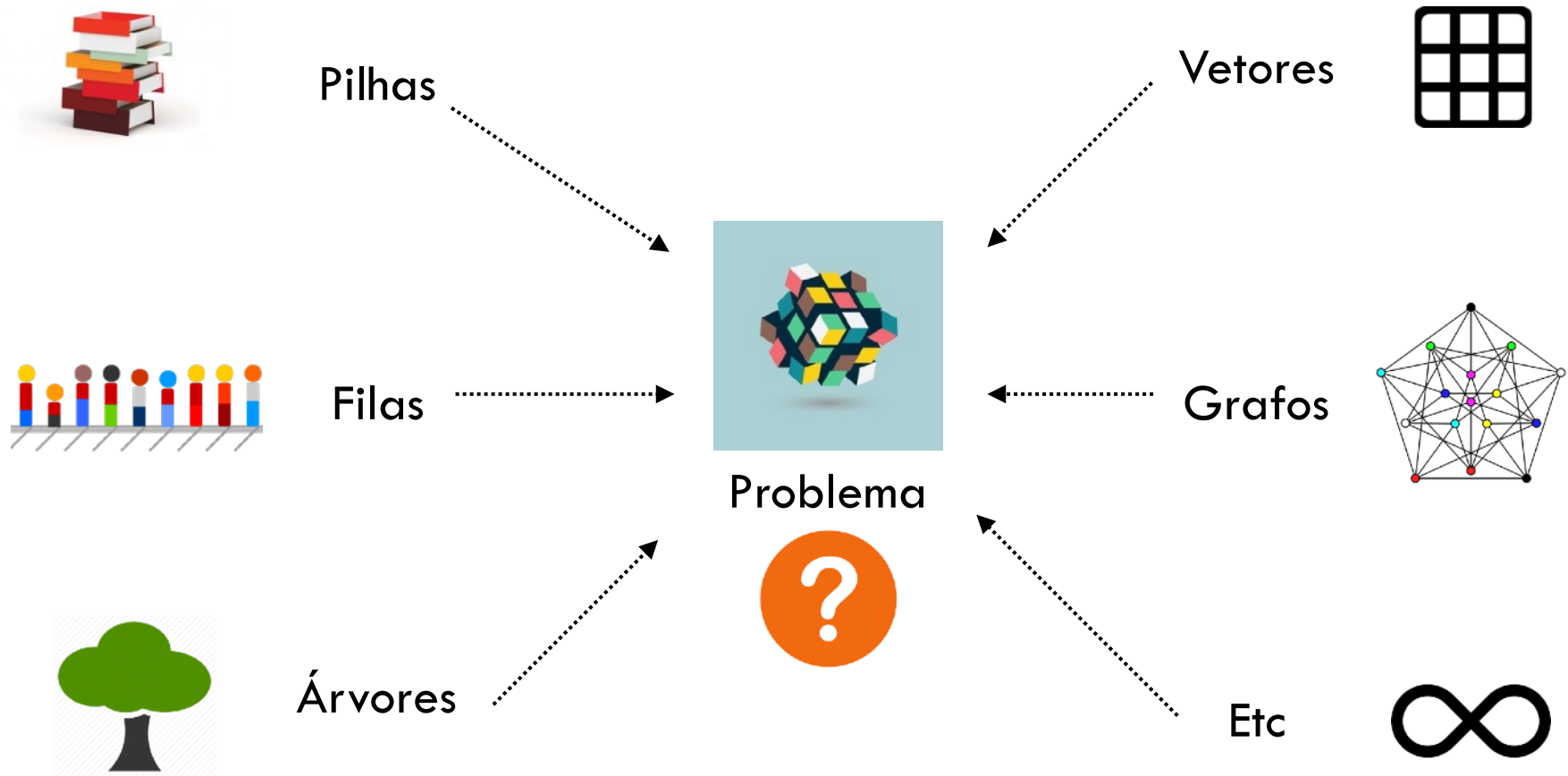


- 1** Introdução
- 2** Árvores Binárias
- 3** Propriedades e Definições
- 4** Inserção em Árvores Binárias
- 5** Pesquisa em Árvores Binárias
- 6** Referências

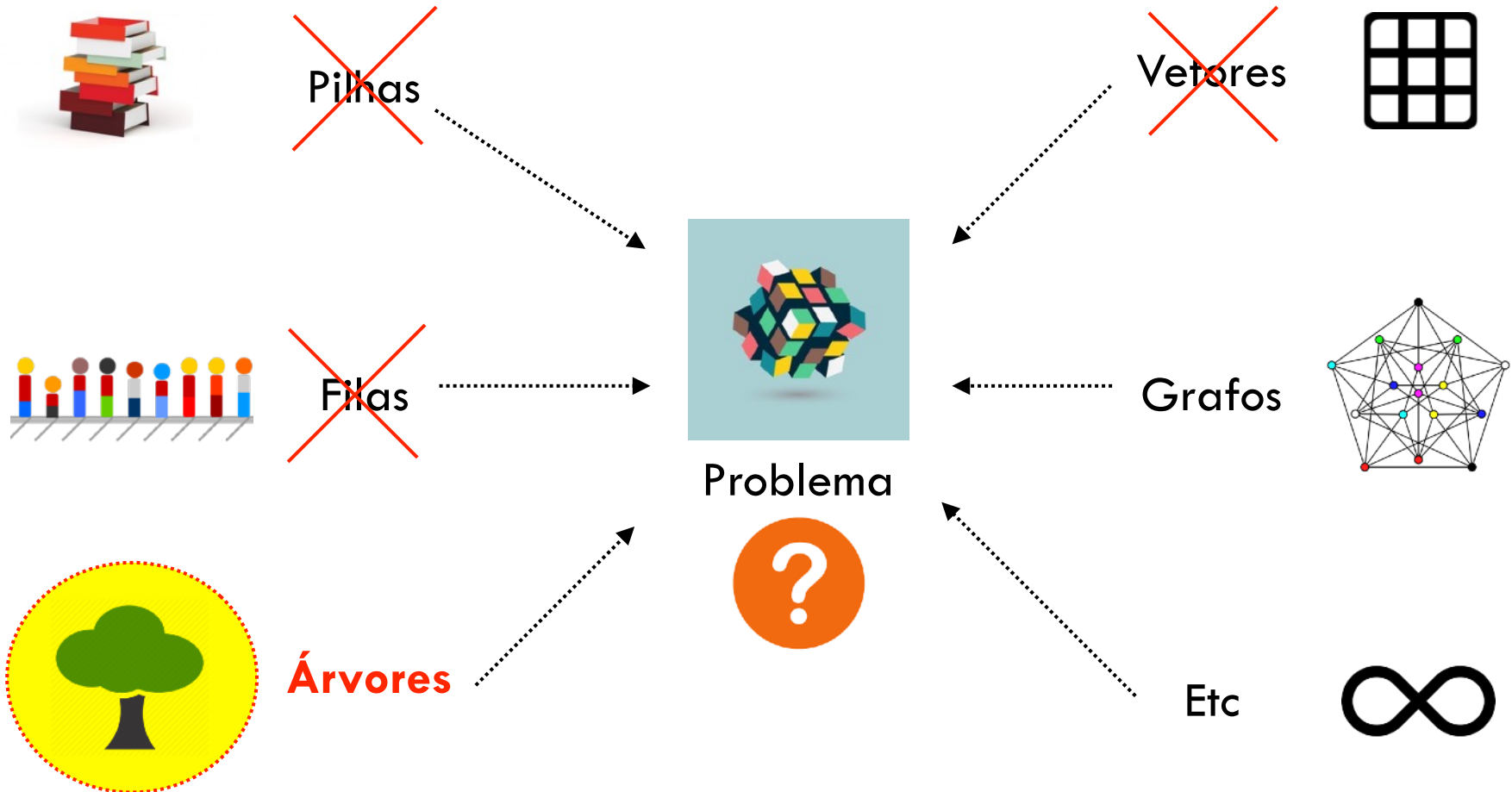
Roteiro

- 1 Introdução**
- 2 Árvores Binárias**
- 3 Propriedades e Definições**
- 4 Inserção em Árvores Binárias**
- 5 Pesquisa em Árvores Binárias**
- 6 Referências**

Introdução



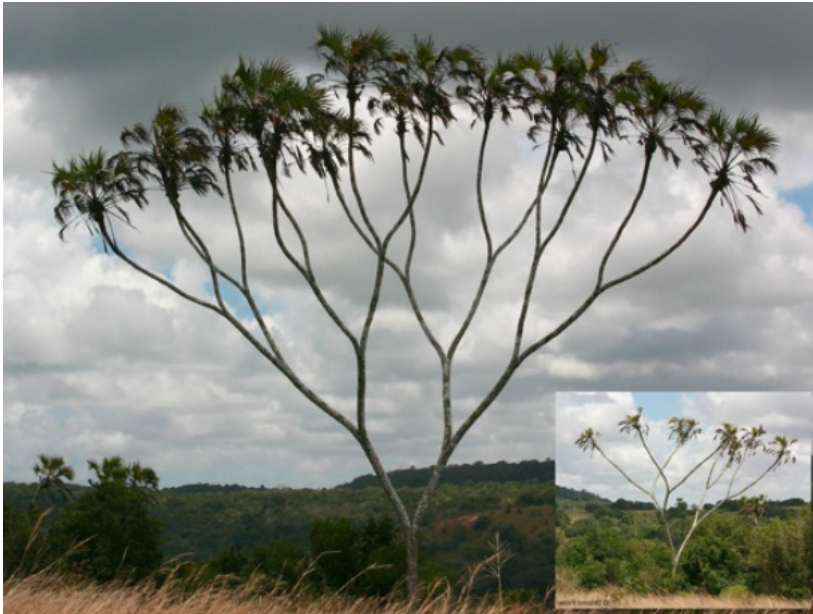
Introdução



Introdução



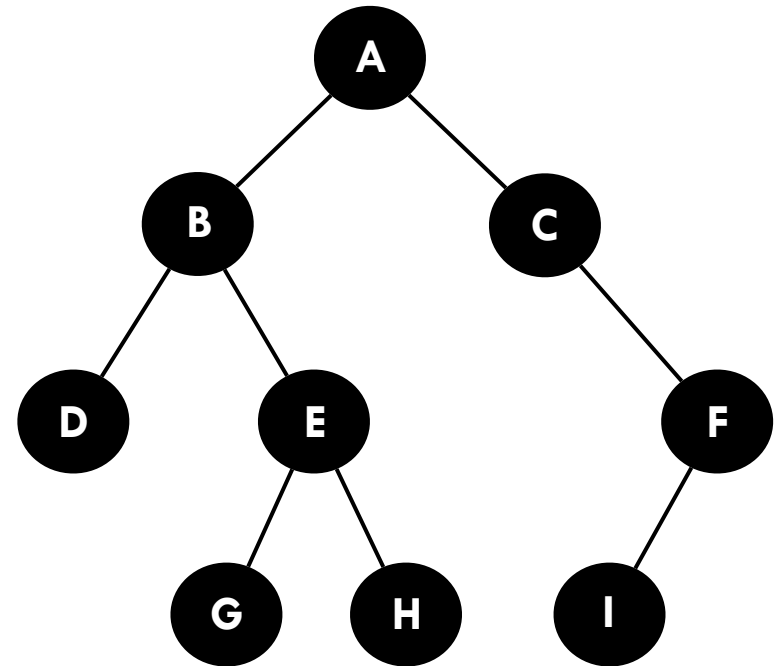
Introdução



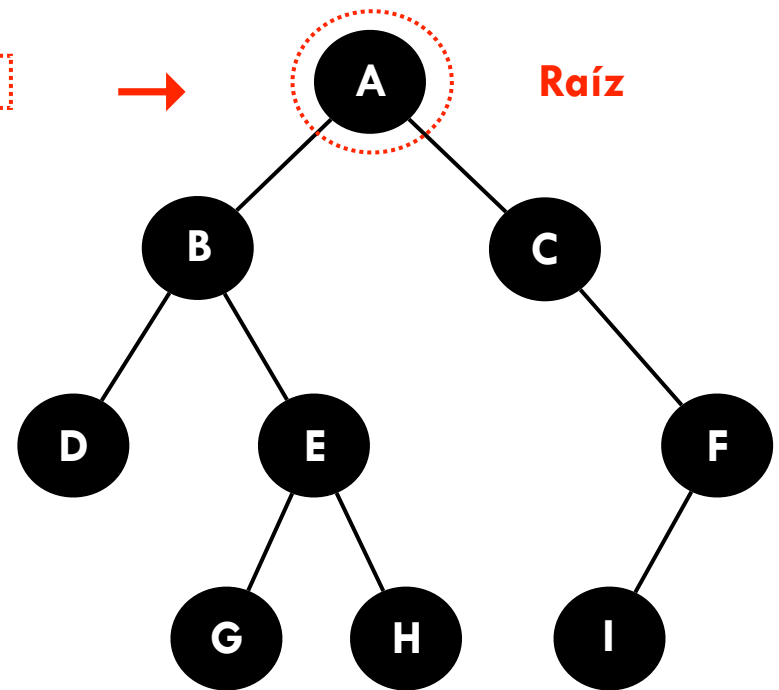
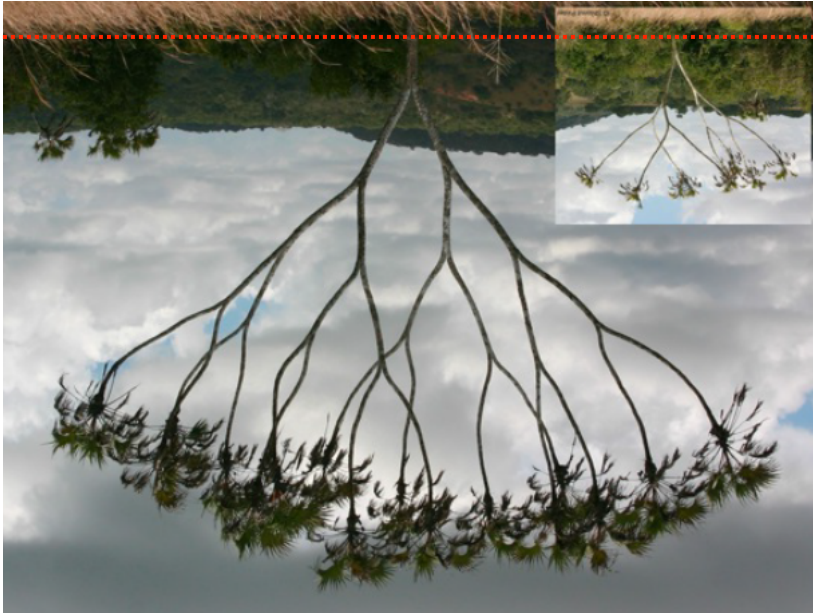
Introdução



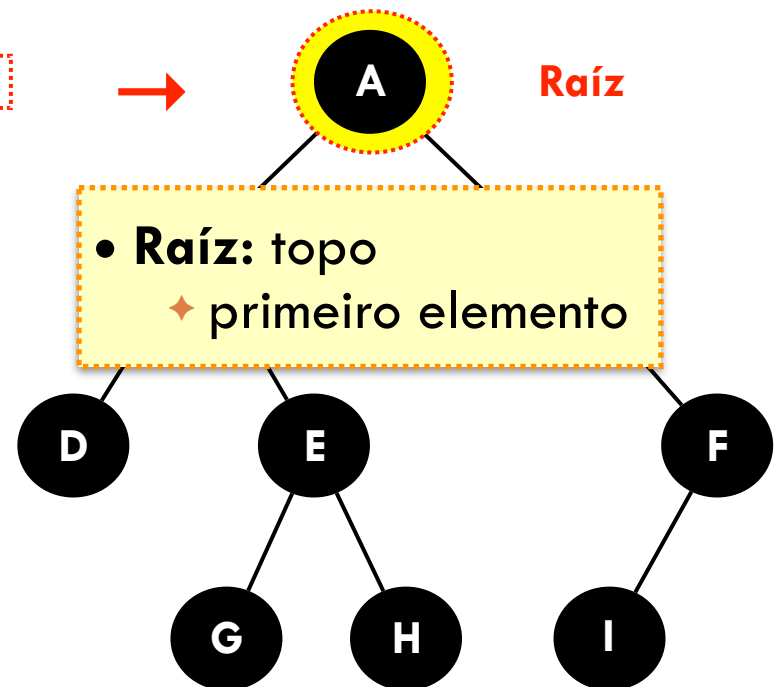
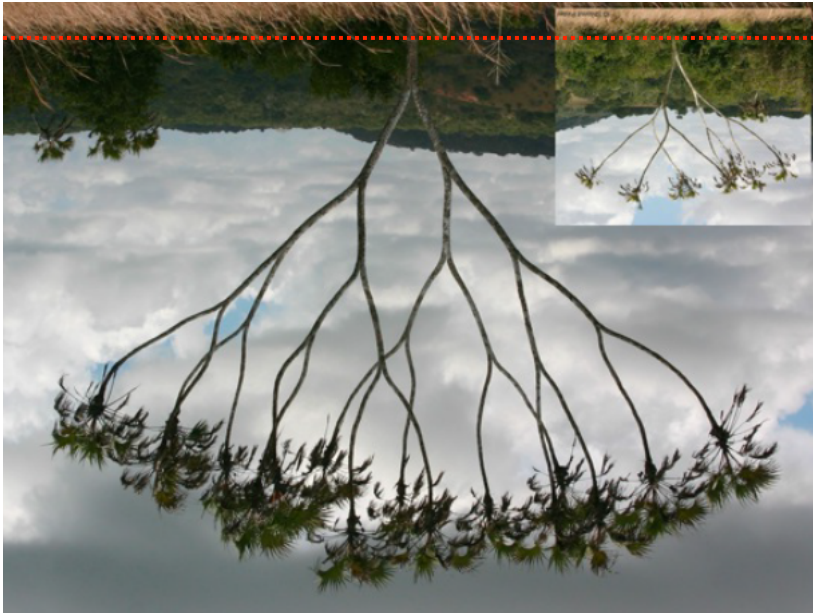
Introdução



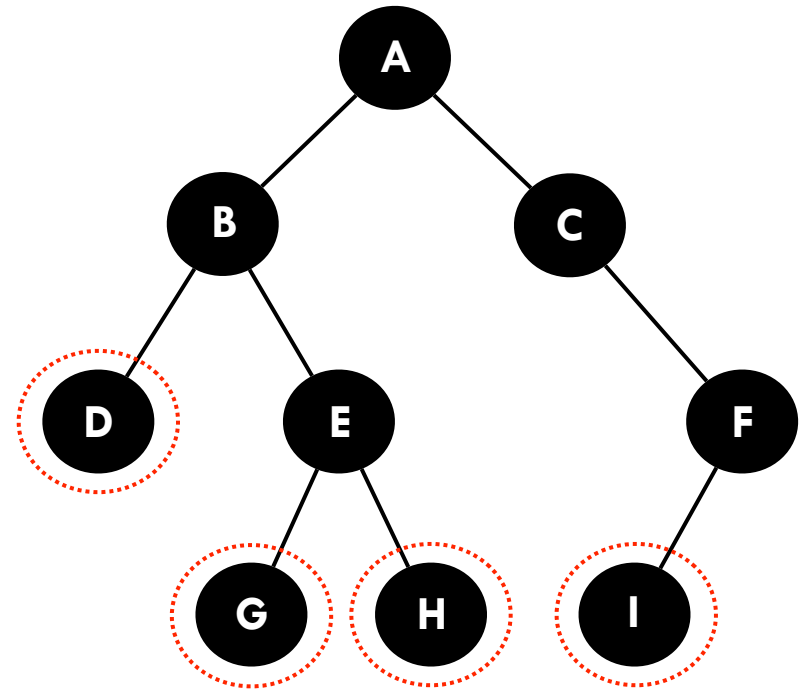
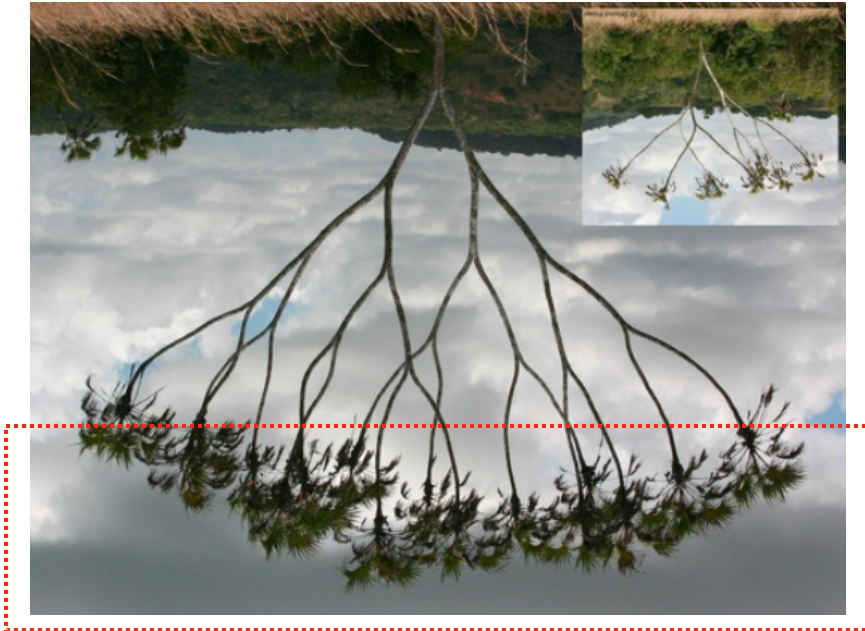
Introdução



Introdução

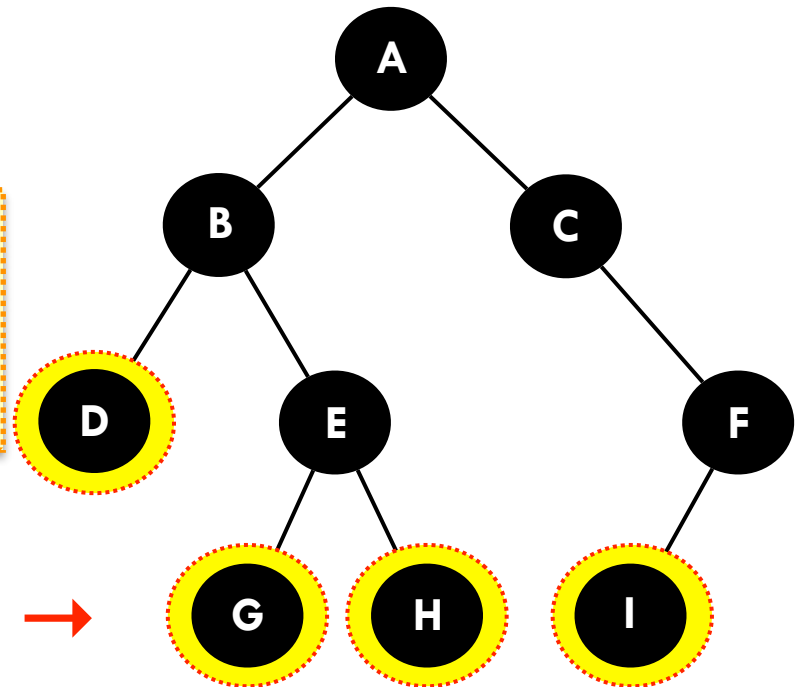


Introdução



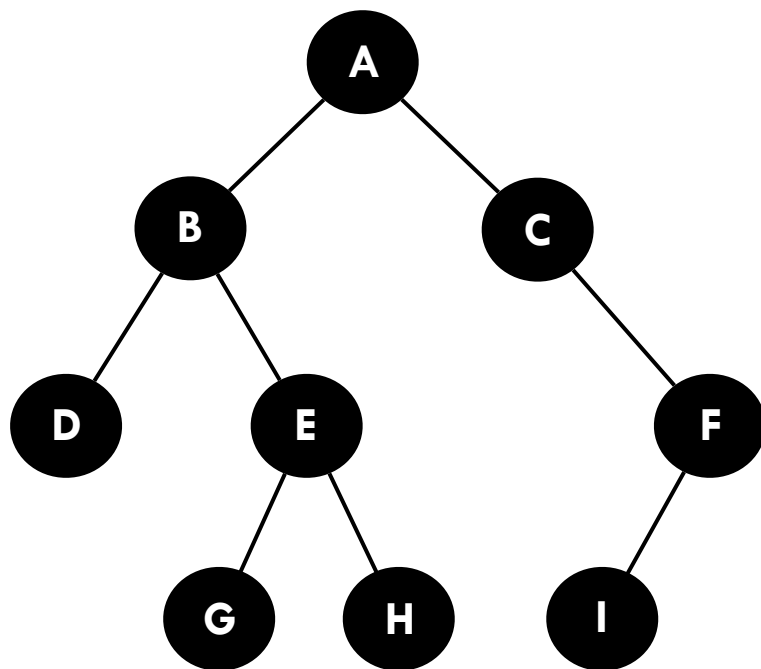
Folhas

Introdução

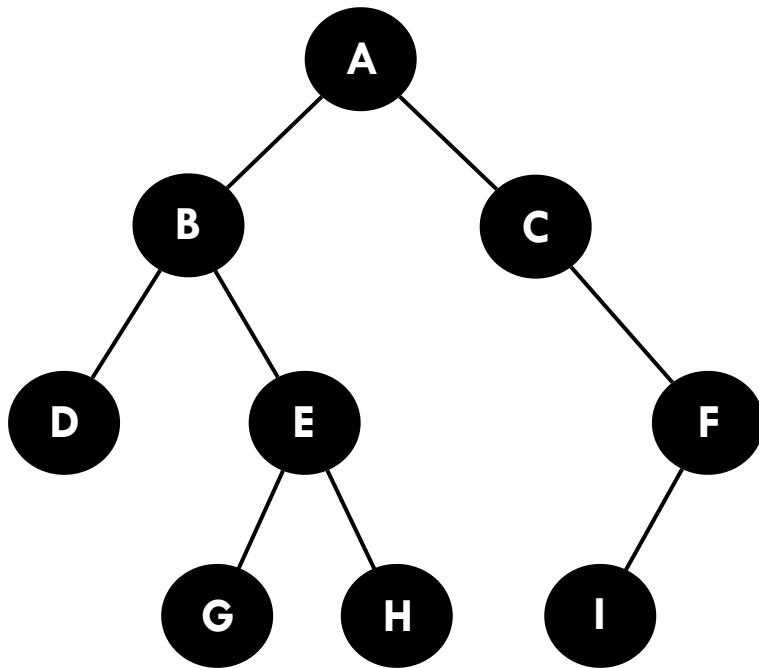


Folhas

Árvores

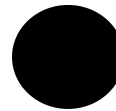


Árvores

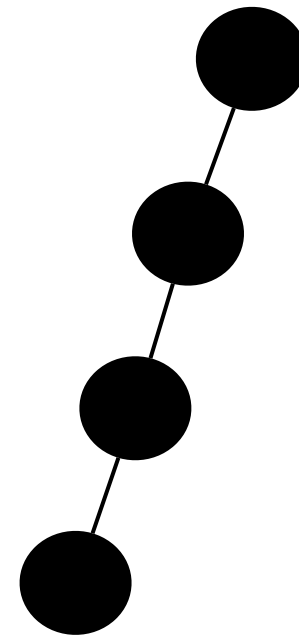


- **Árvores:** alto volume de buscas
 - ✦ Dicionários
 - ✦ Filas de prioridades (Heaps)
 - ✦ Operações custam tempo proporcional à forma da árvore
 - ✦ Árvore completa (n nós)
 - ✦ $O(\log n)$

Exemplos de árvores



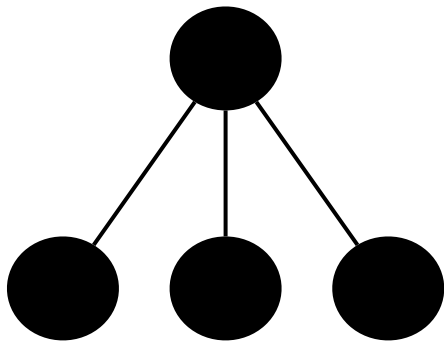
(b)



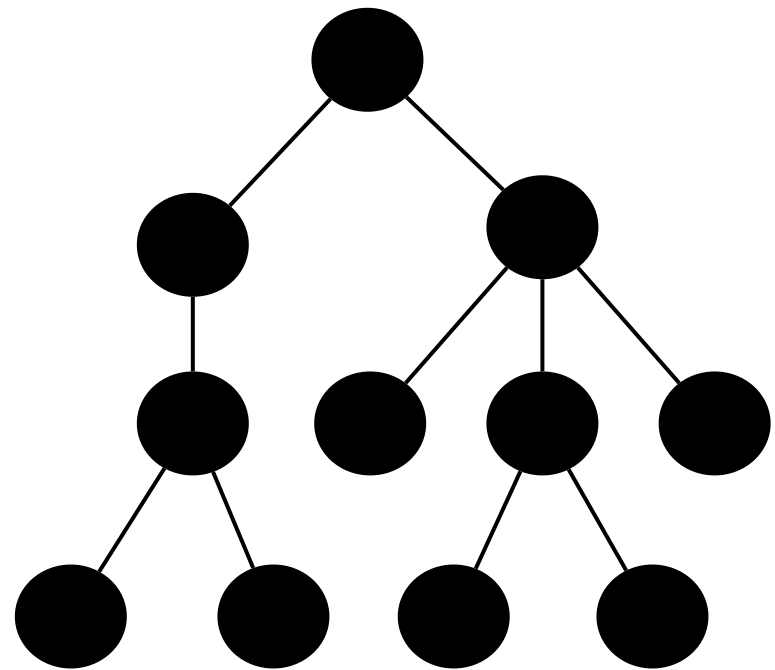
(c)

(a)
árvore vazia

Exemplos de árvores



(d)

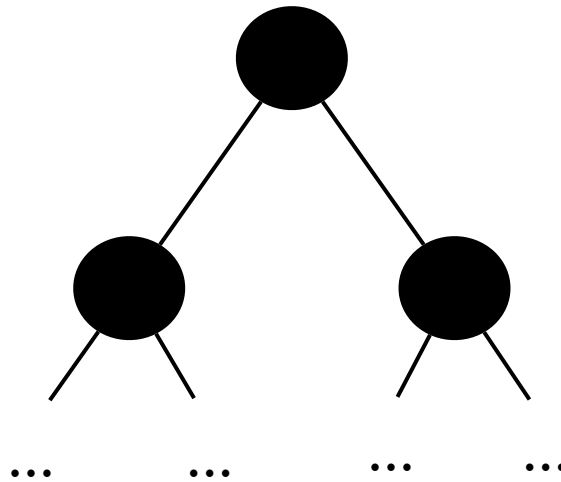


(e)

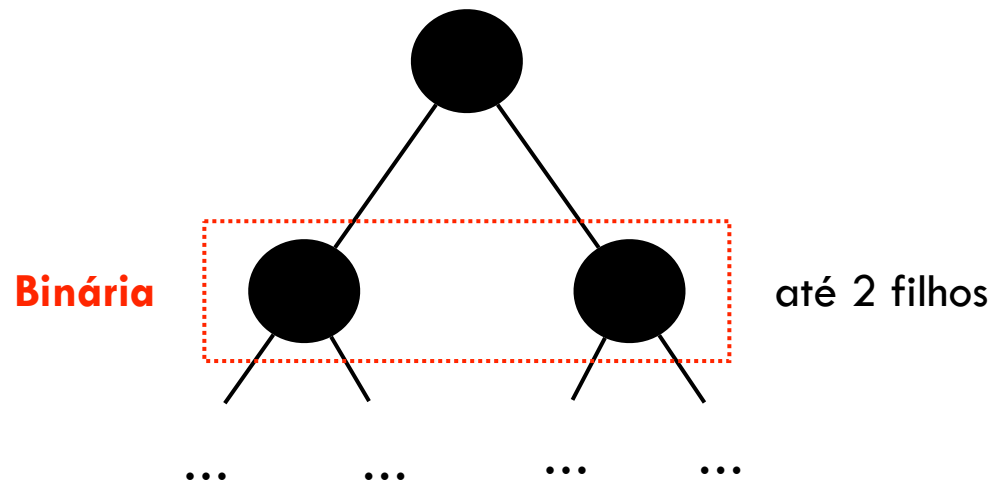
Roteiro

- 1 Introdução
- 2 Árvores Binárias
- 3 Propriedades e Definições
- 4 Inserção em Árvores Binárias
- 5 Pesquisa em Árvores Binárias
- 6 Referências

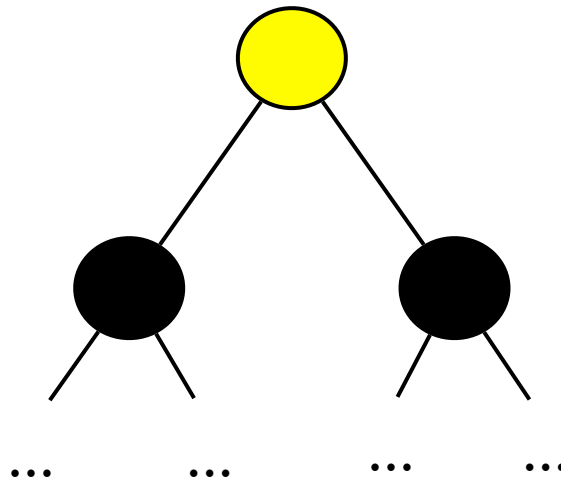
Árvore Binária



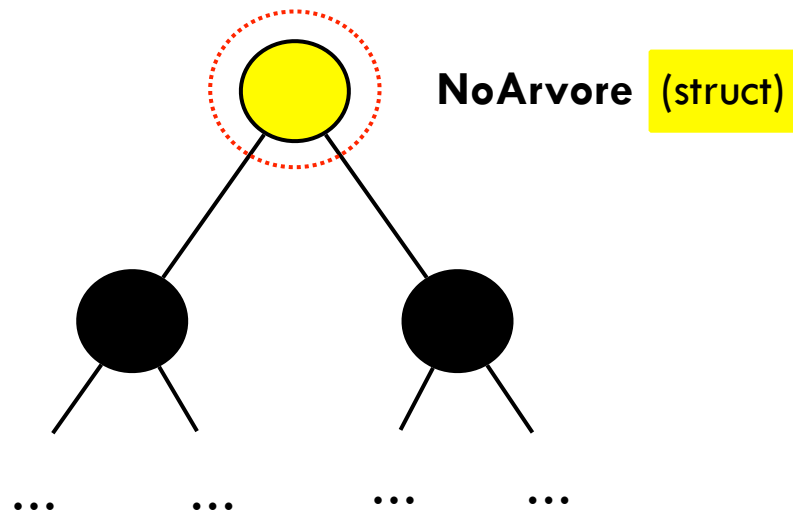
Árvore Binária



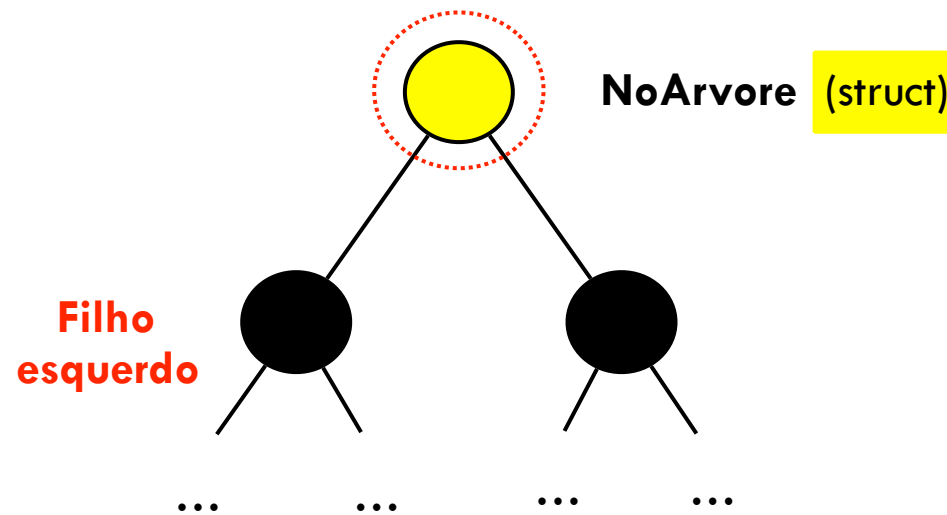
Árvore Binária



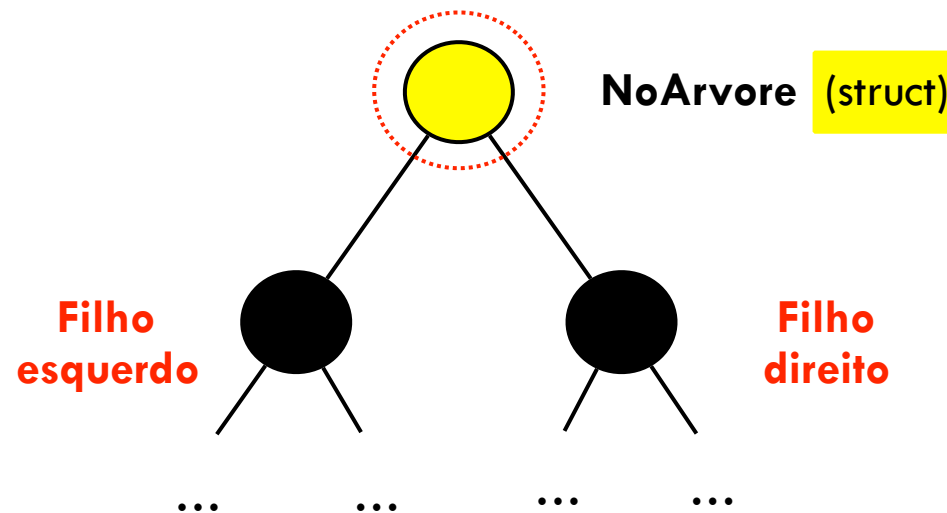
Árvore Binária



Árvore Binária

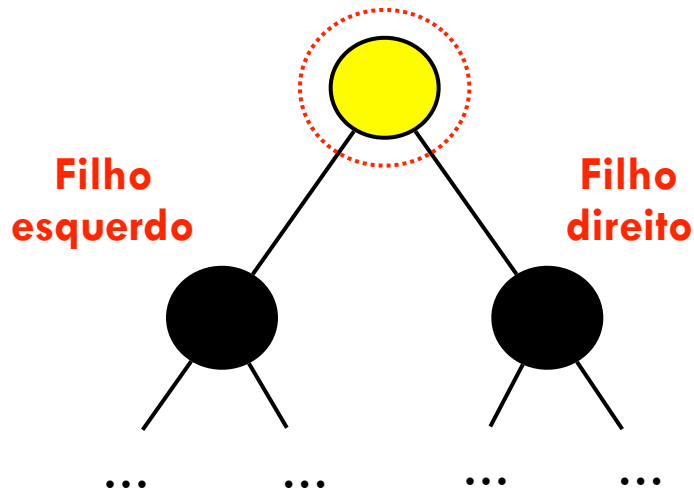


Árvore Binária



Árvore Binária

NoArvore

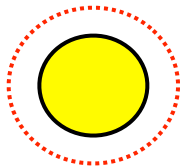


NoArvore

1. inteiro chave
2. /* ... */
3. NoArvore* filho à direita
4. NoArvore* filho à esquerda
5. NoArvore* pai [opcional]

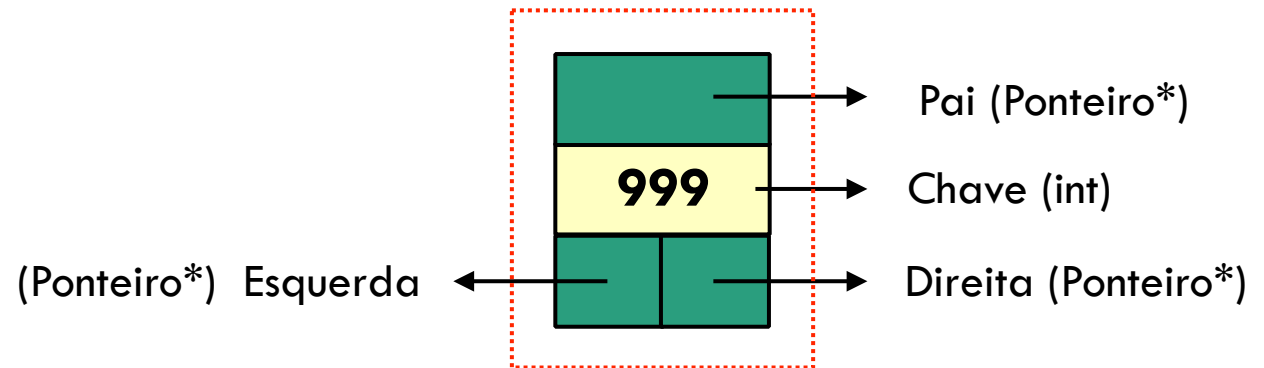
Árvore Binária

NoArvore



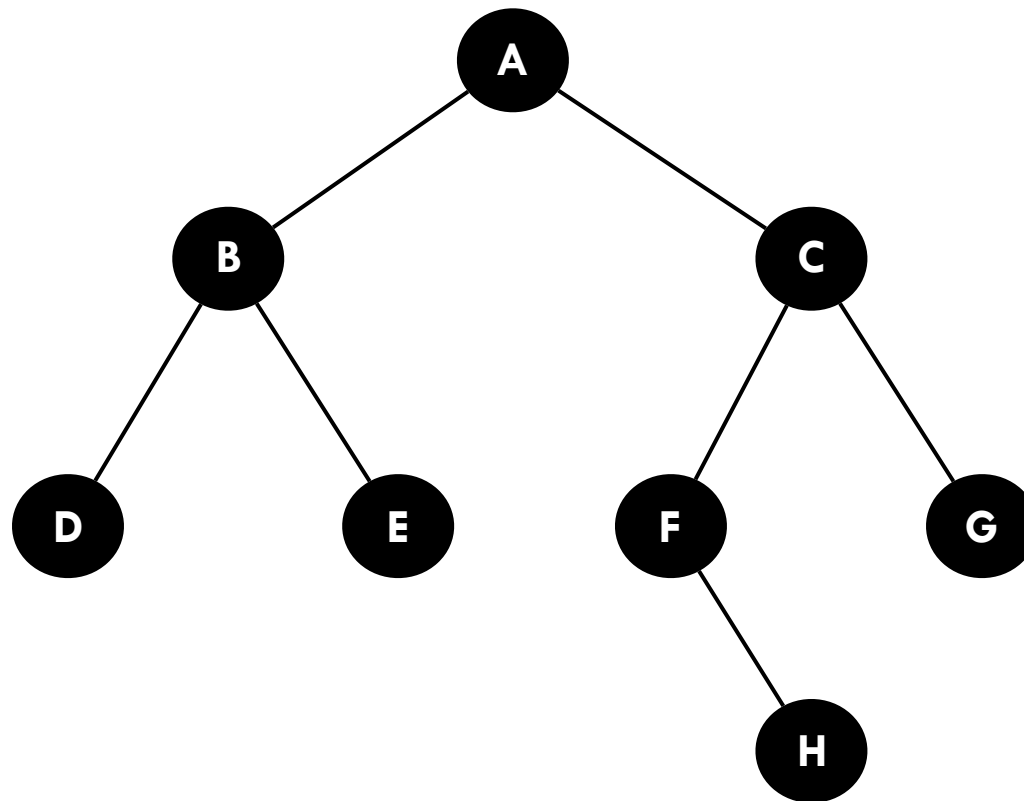
Abstração

Struct

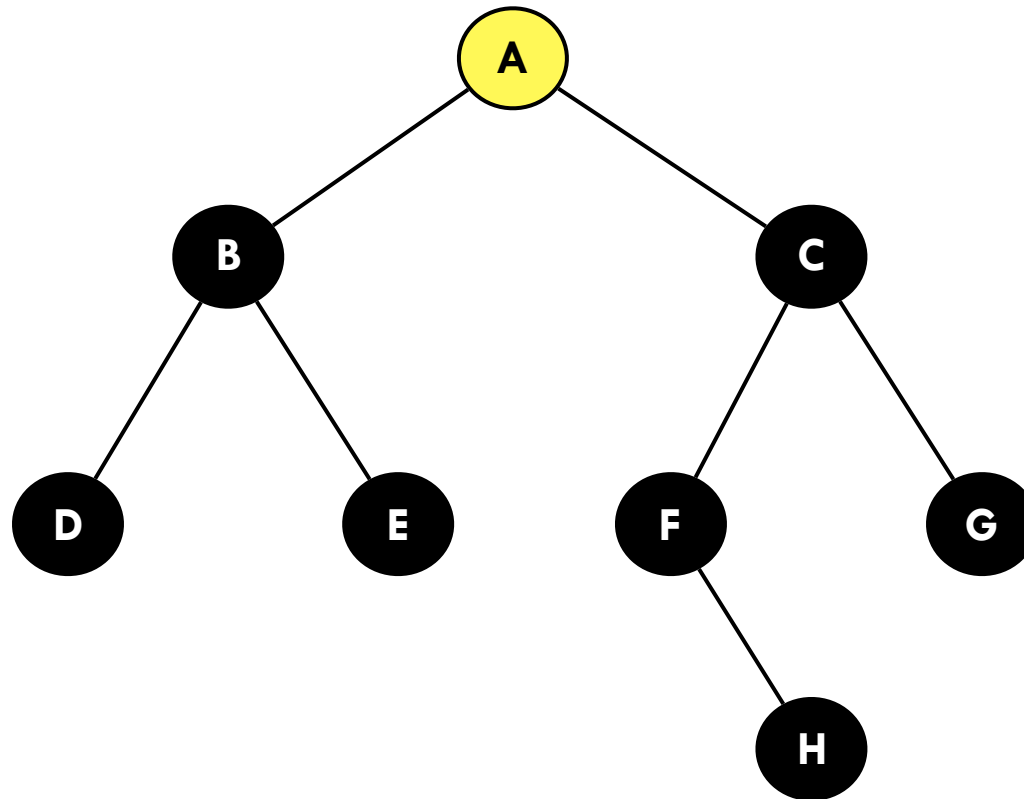


Tipo Abstrato
de Dados

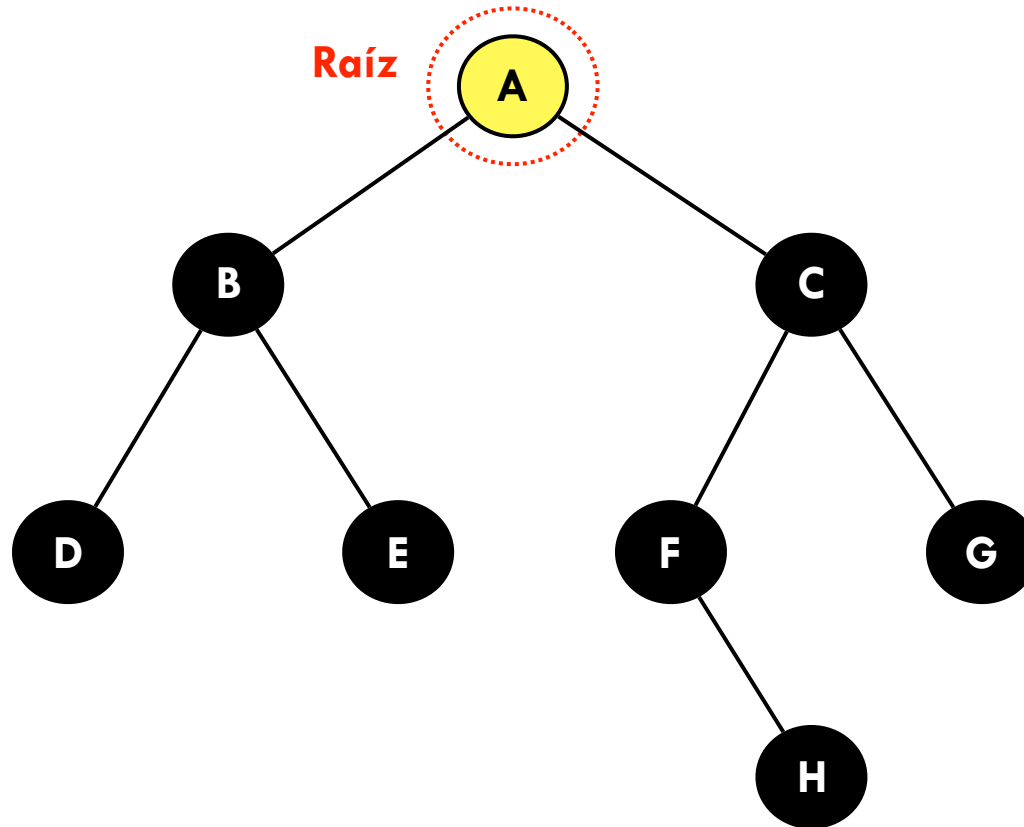
Árvore Binária



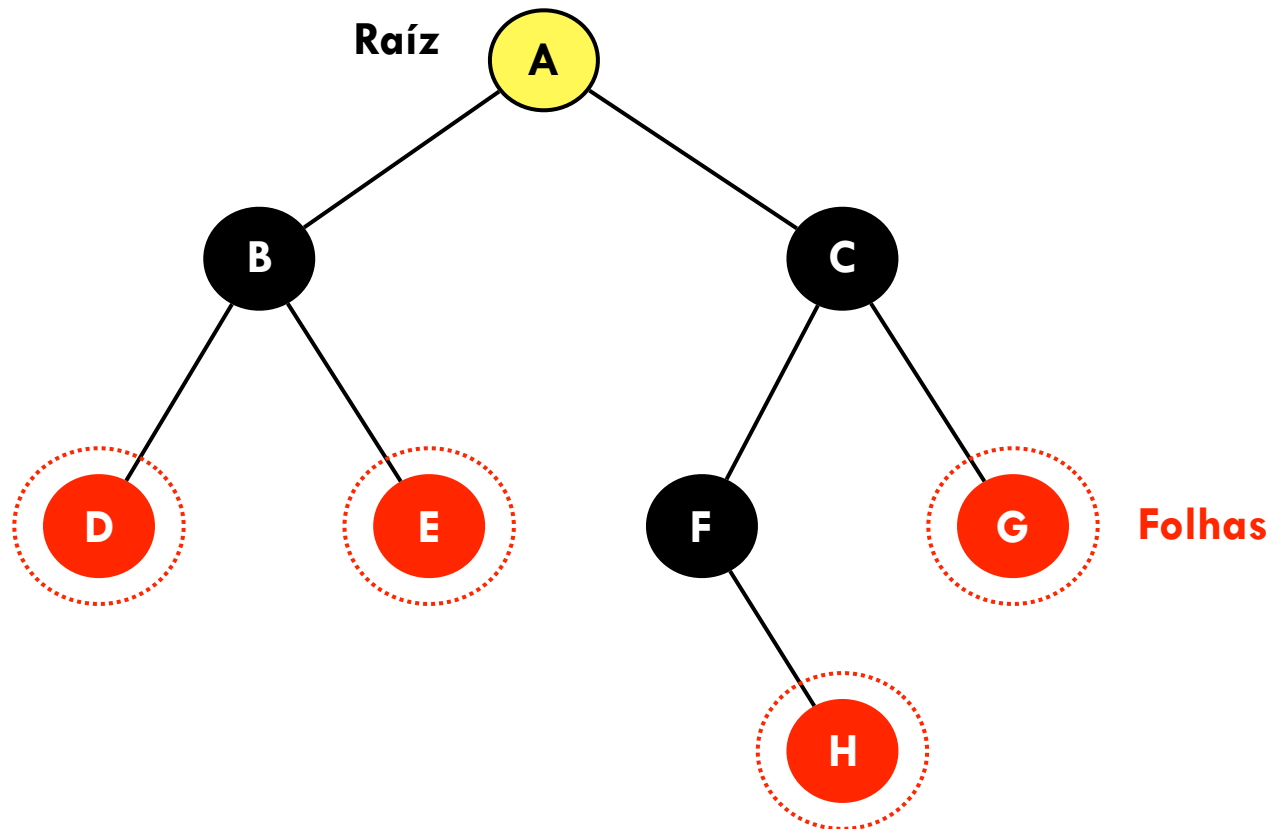
Árvore Binária



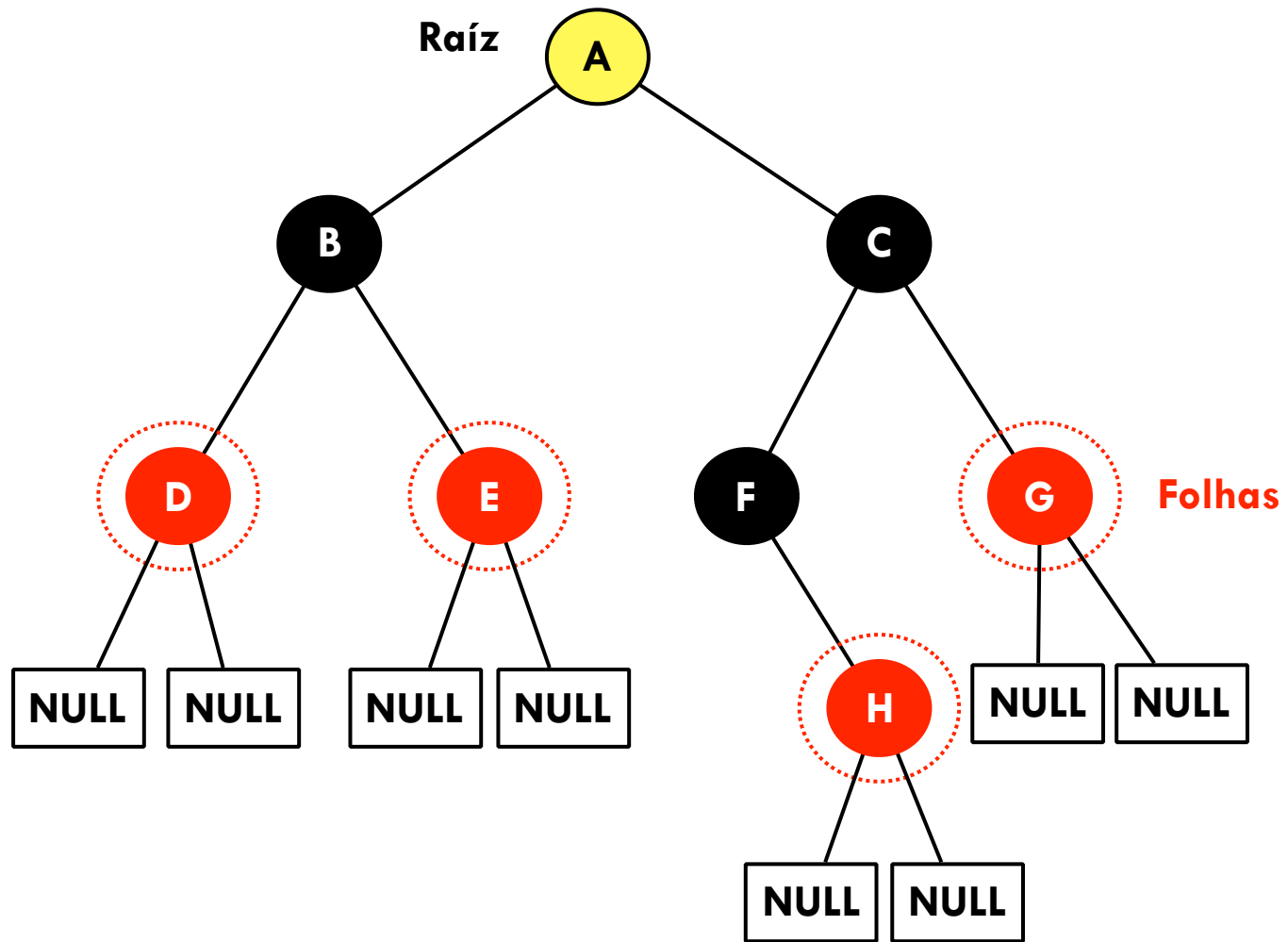
Árvore Binária



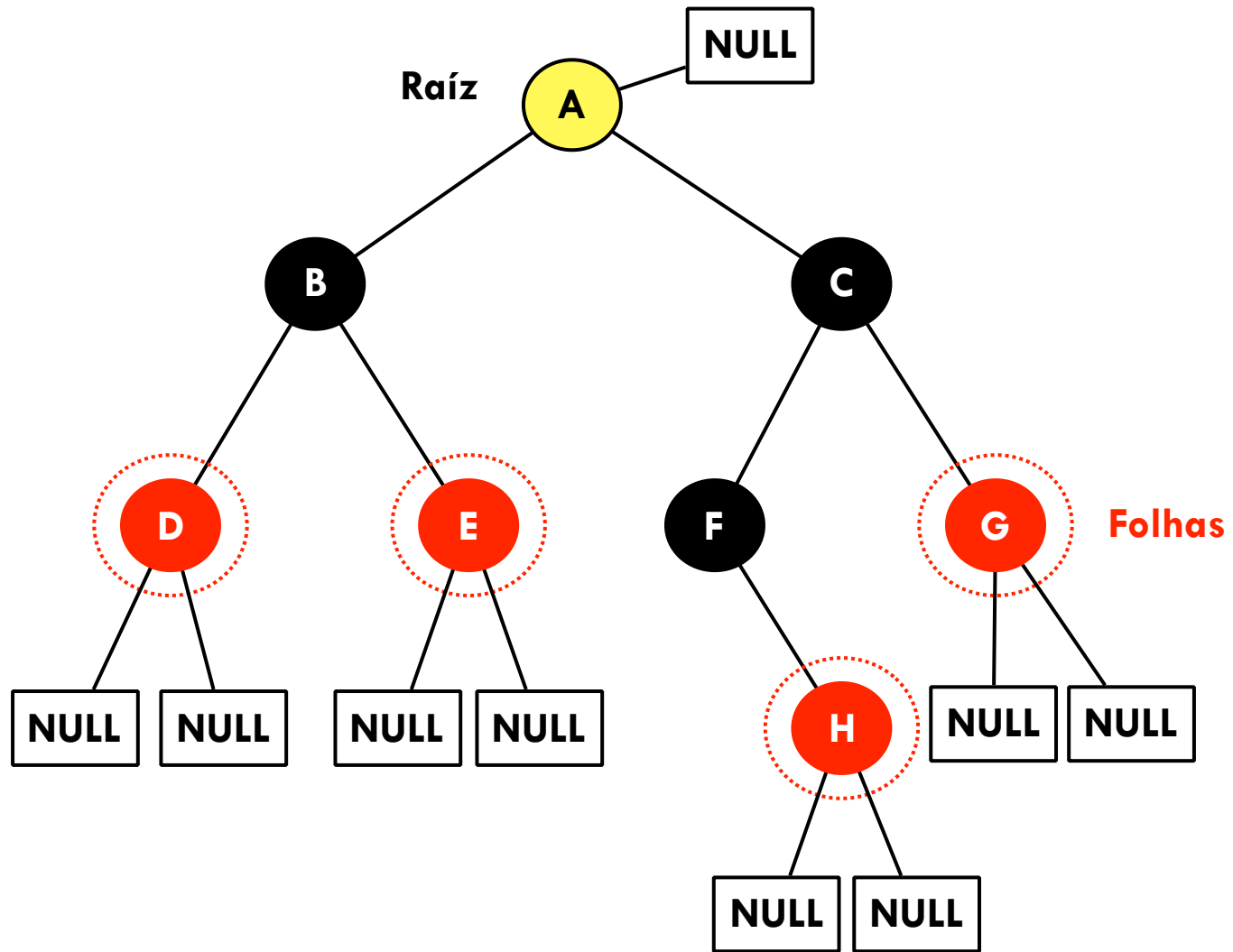
Árvore Binária



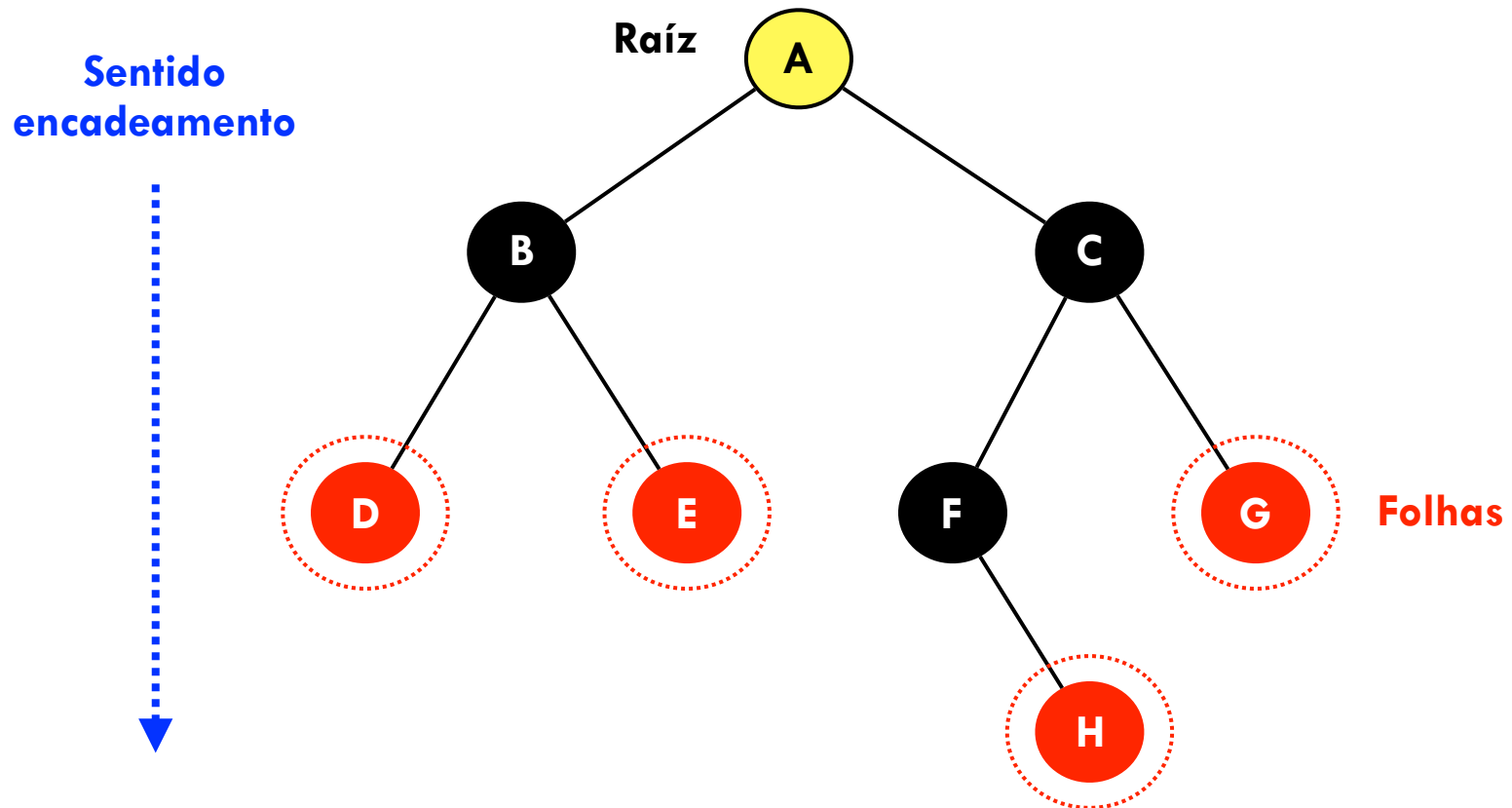
Árvore Binária



Árvore Binária



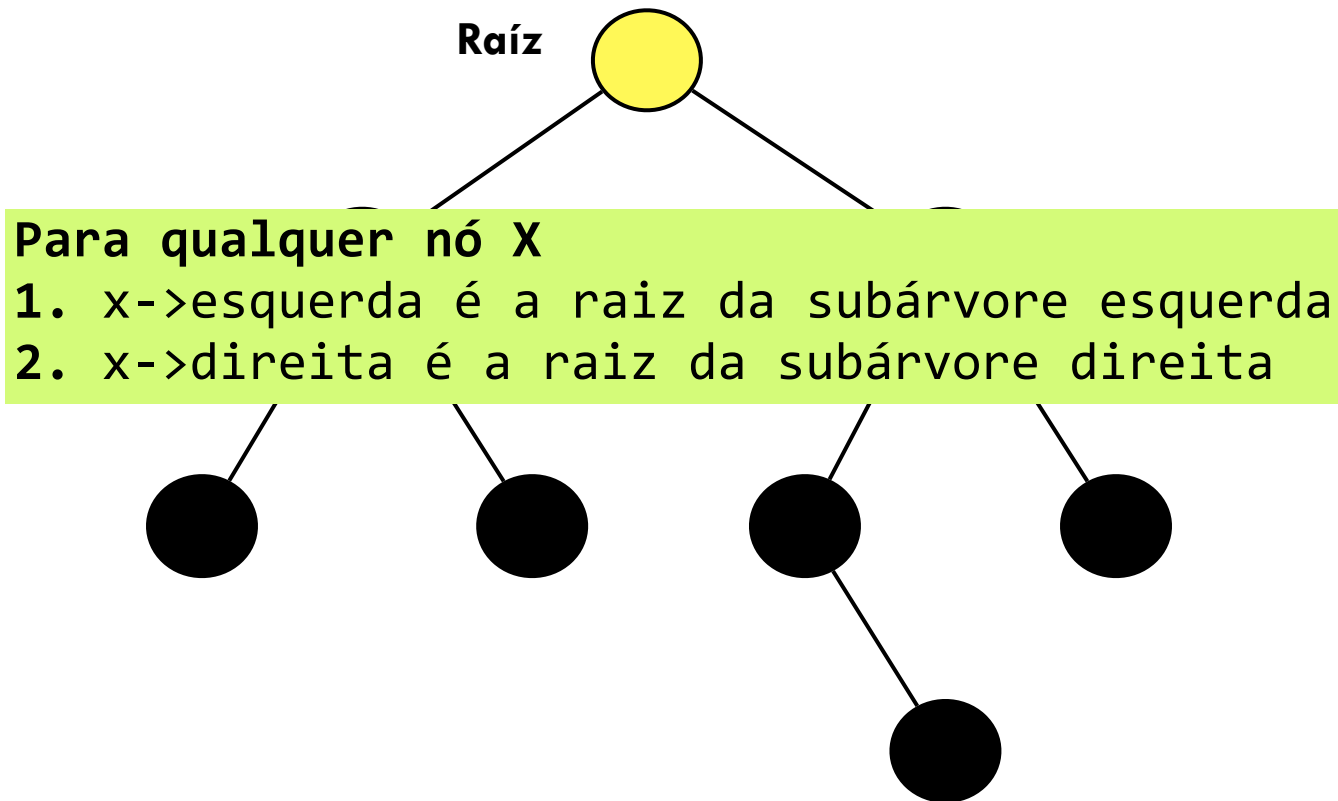
Árvore Binária



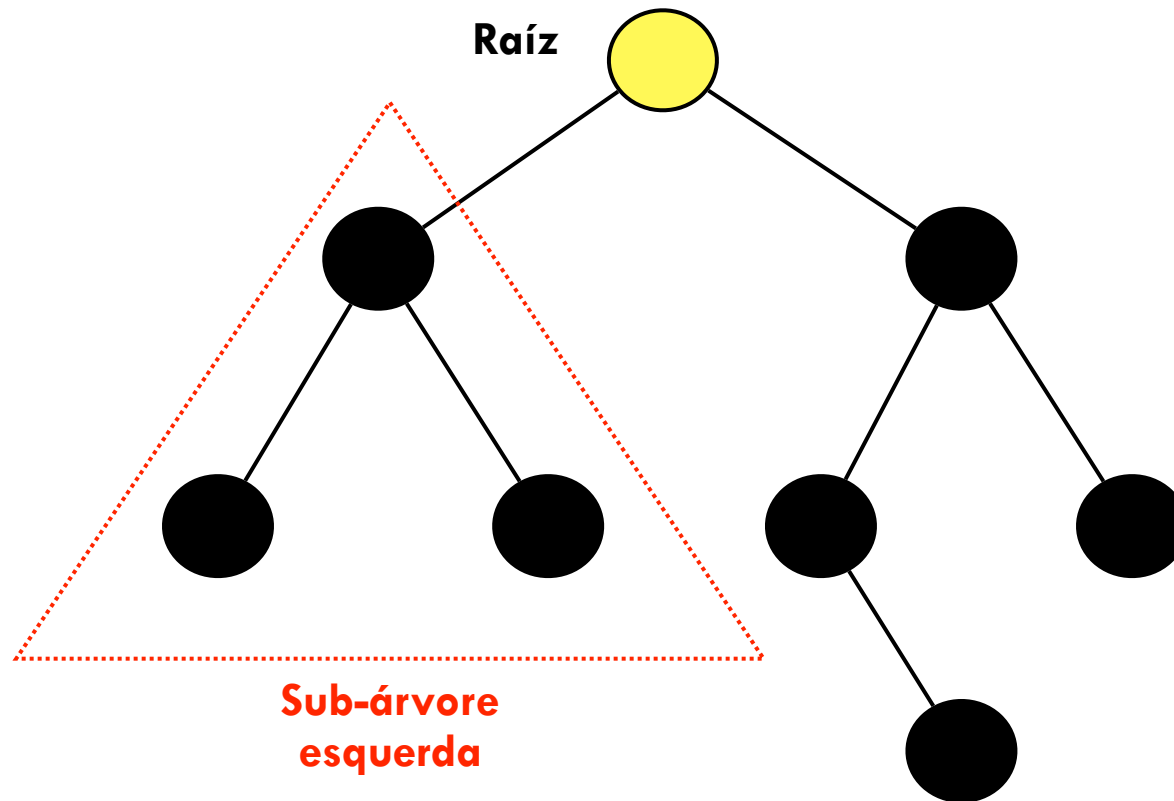
Roteiro

- 1 Introdução
- 2 Árvores Binárias
- 3 Propriedades e Definições
- 4 Inserção em Árvores Binárias
- 5 Pesquisa em Árvores Binárias
- 6 Referências

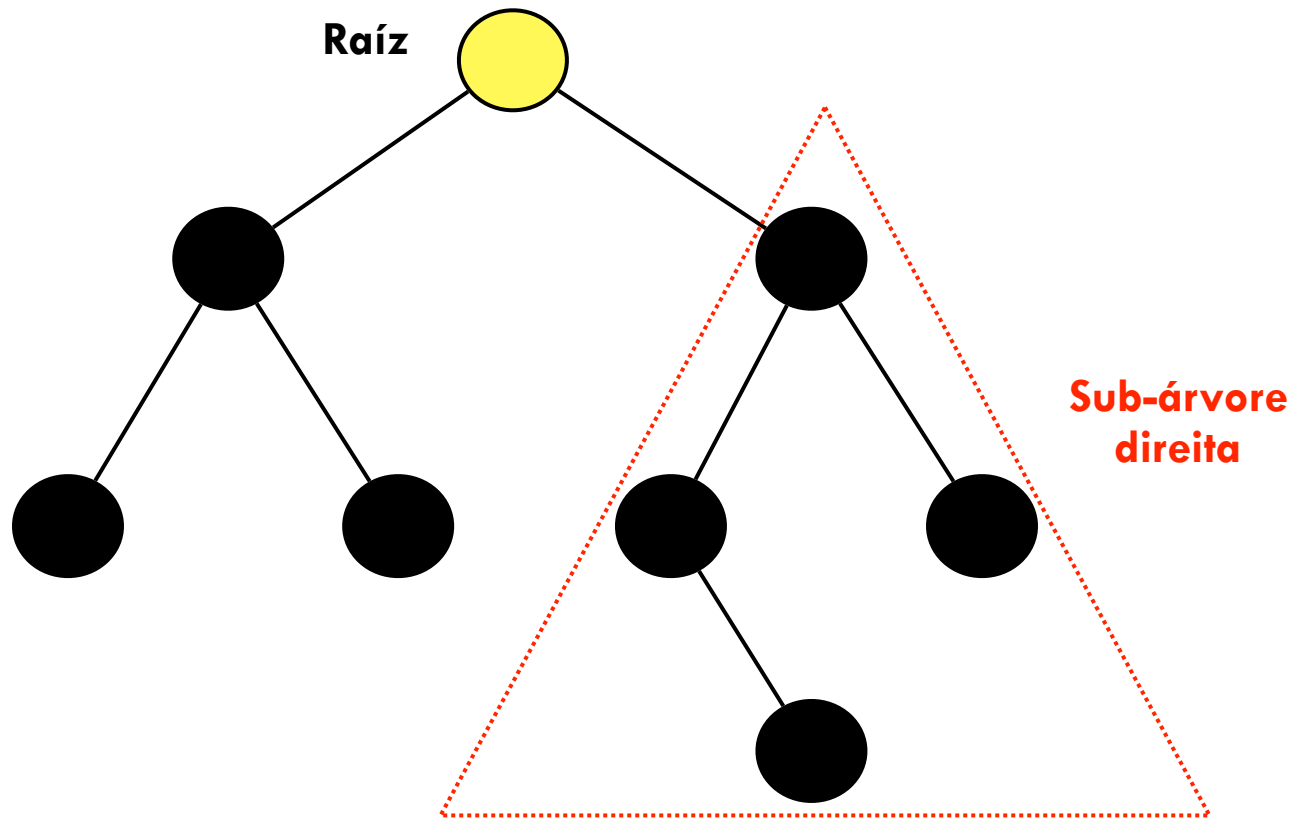
Sub-árvores



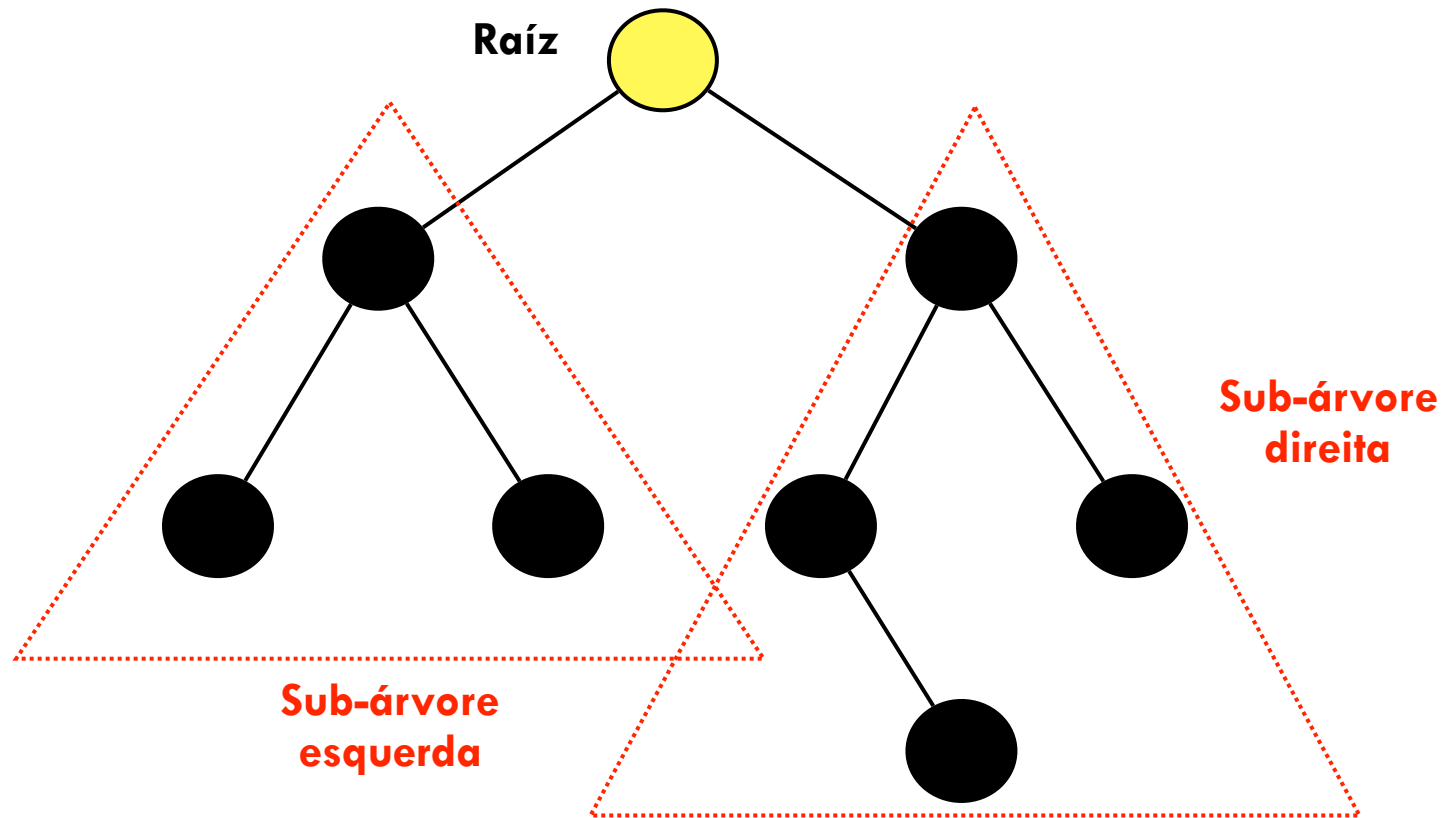
Sub-árvores



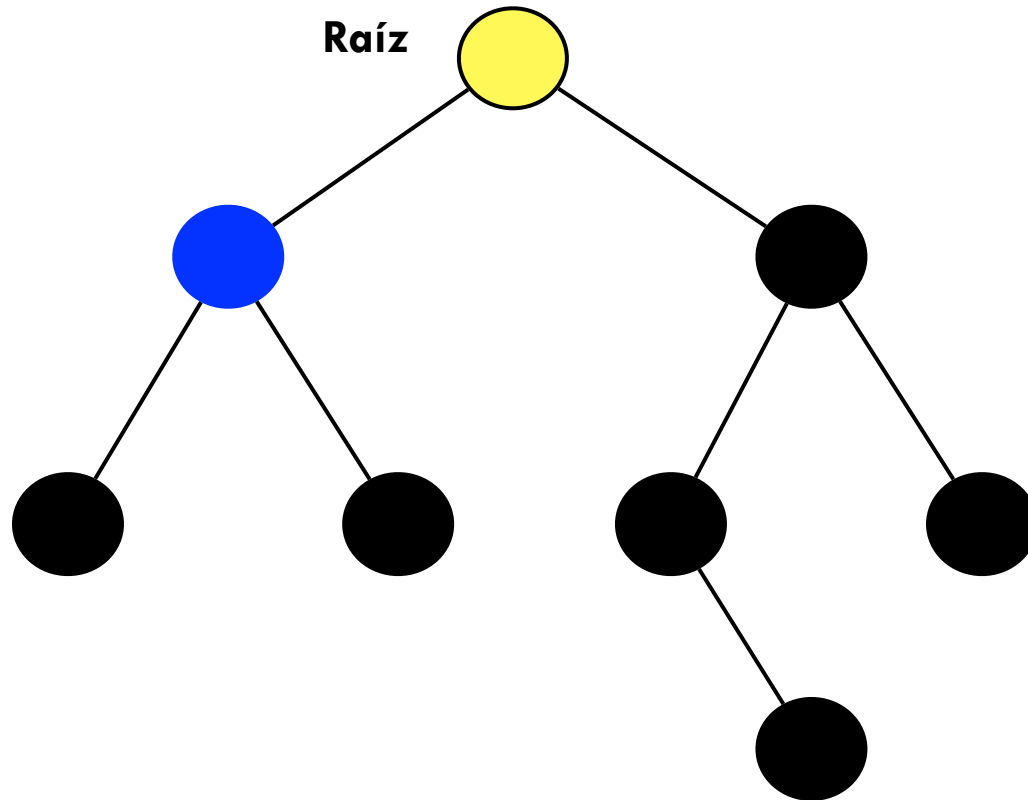
Sub-árvores



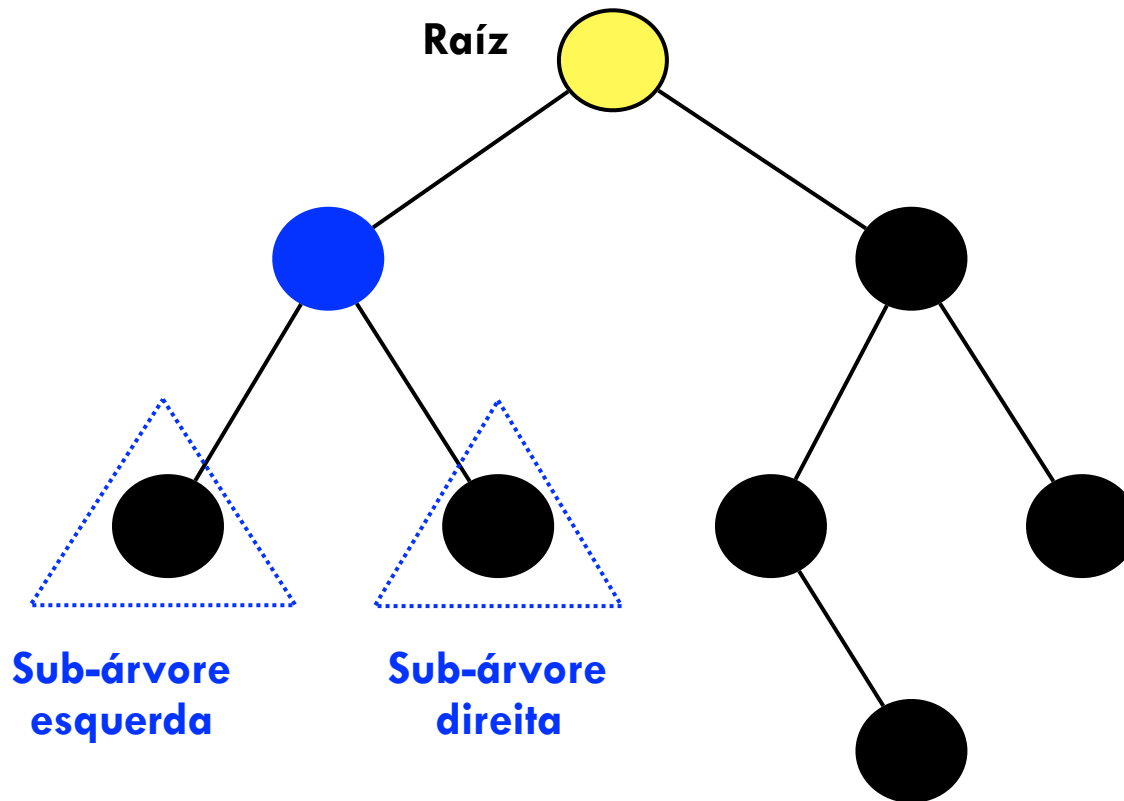
Sub-árvores



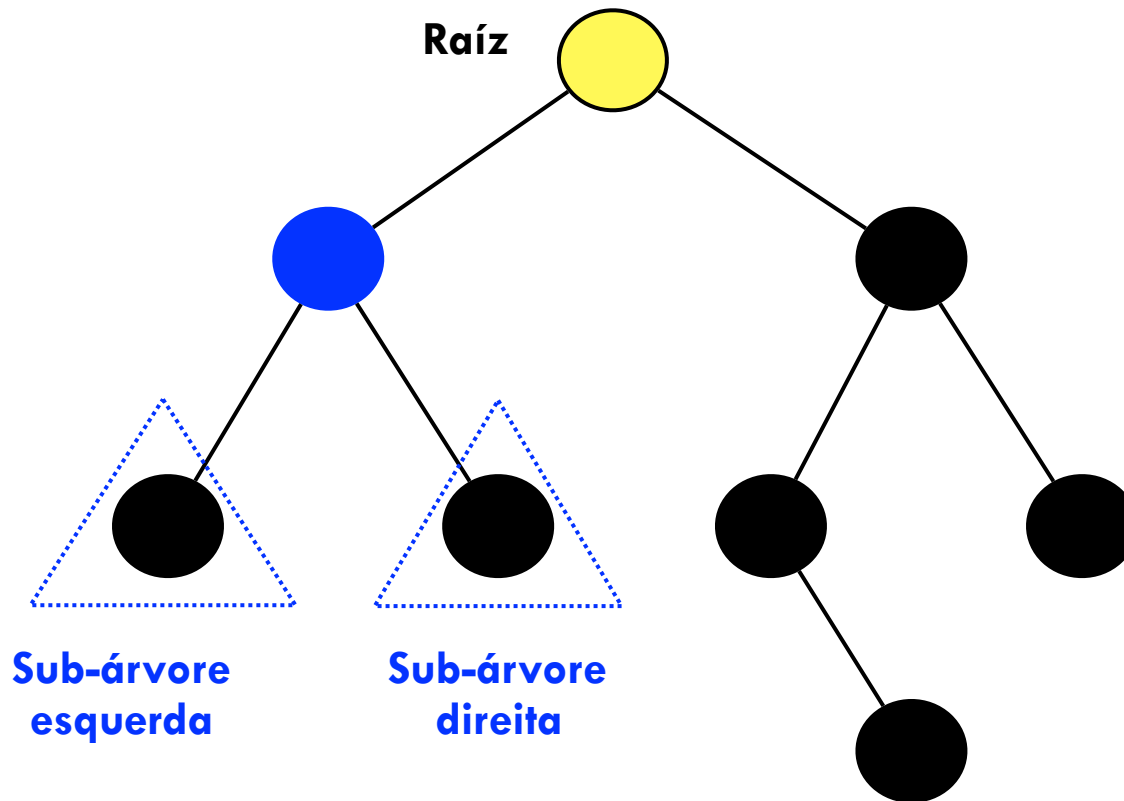
Sub-árvores



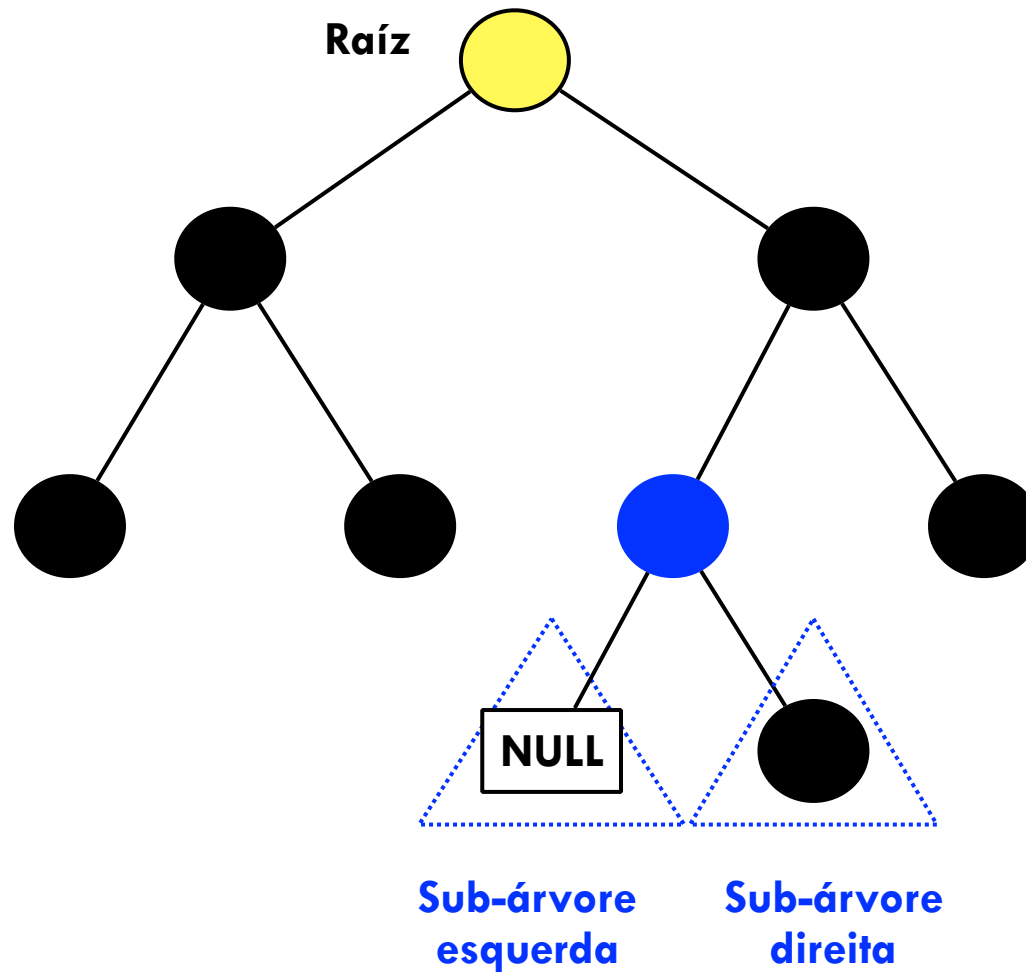
Sub-árvores



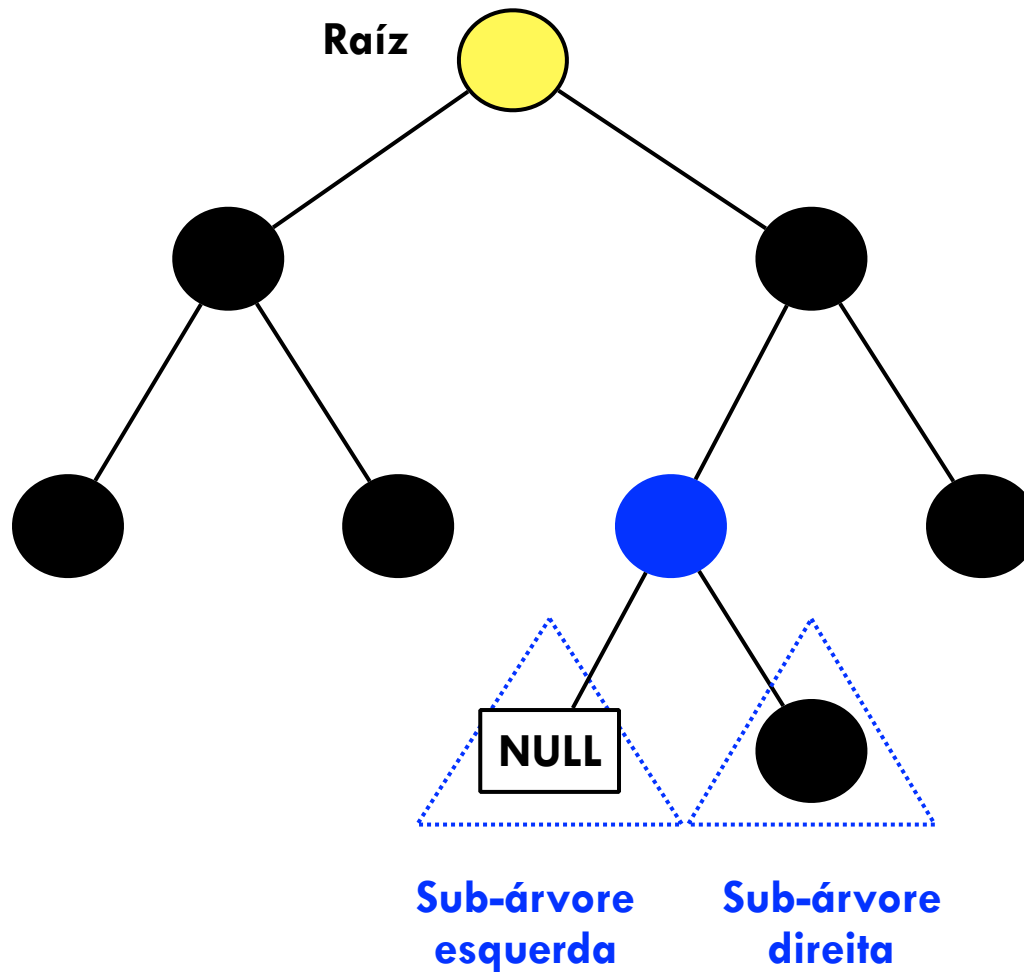
Sub-árvores



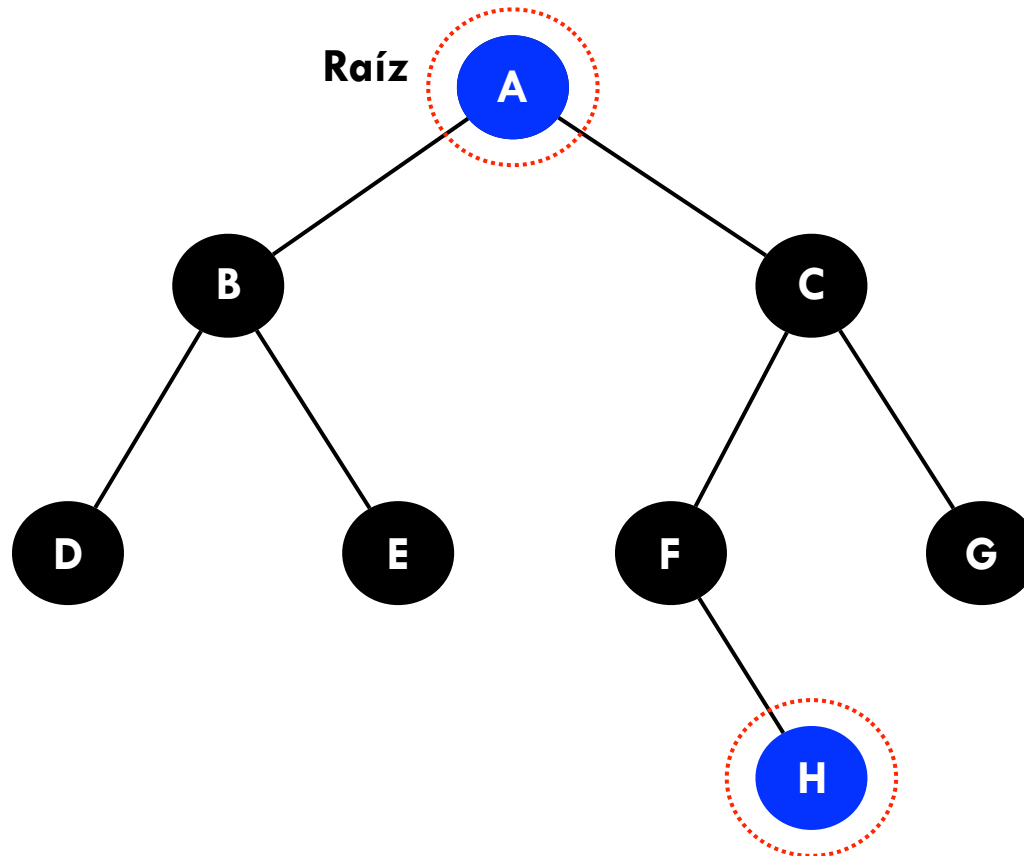
Sub-árvores



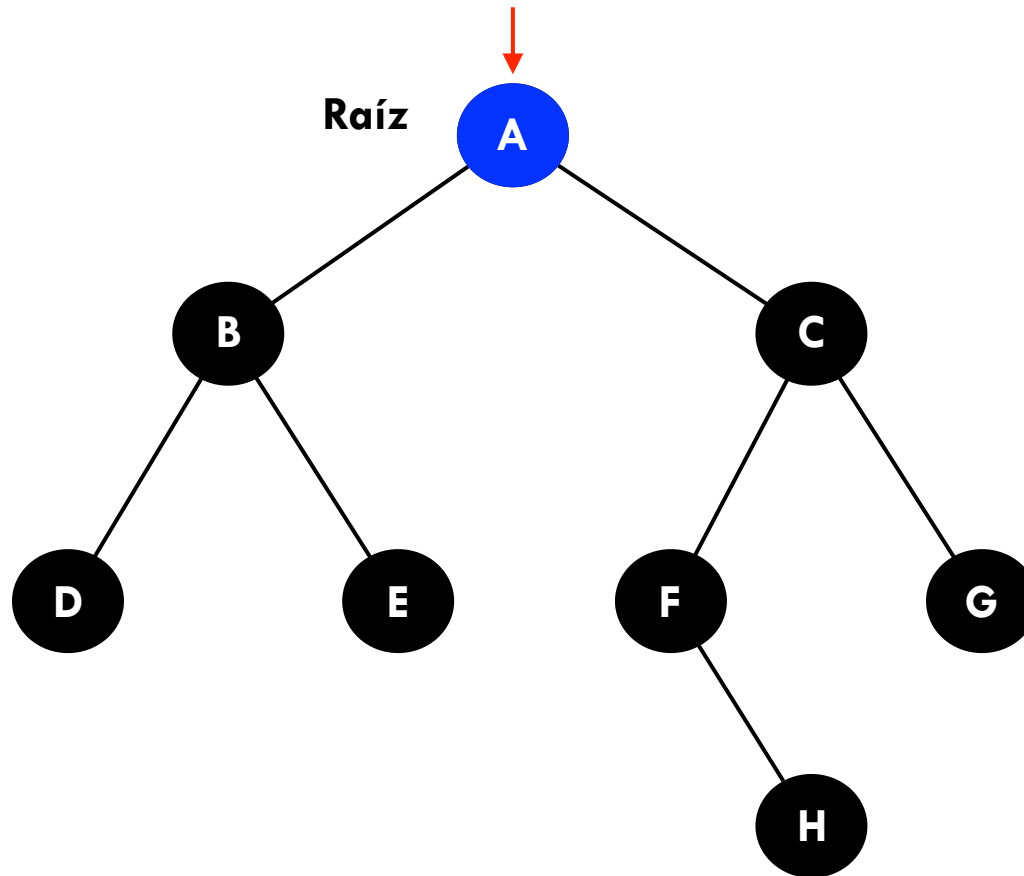
Sub-árvores



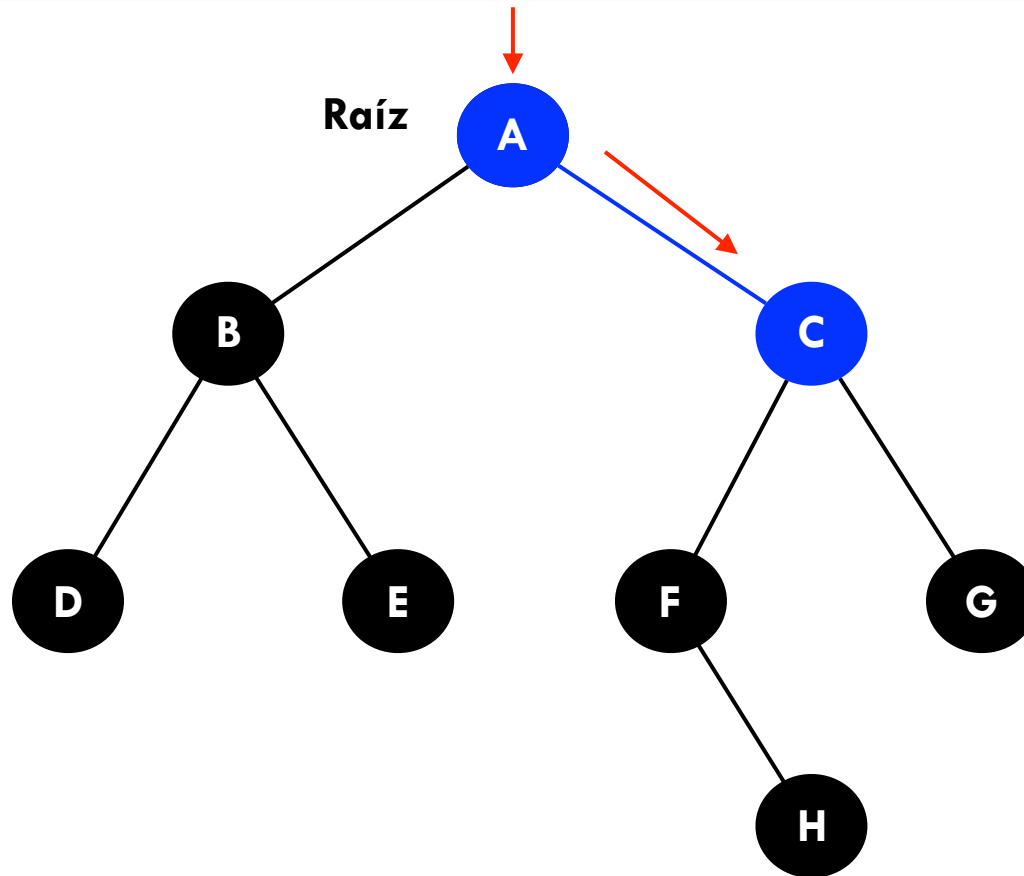
Caminho



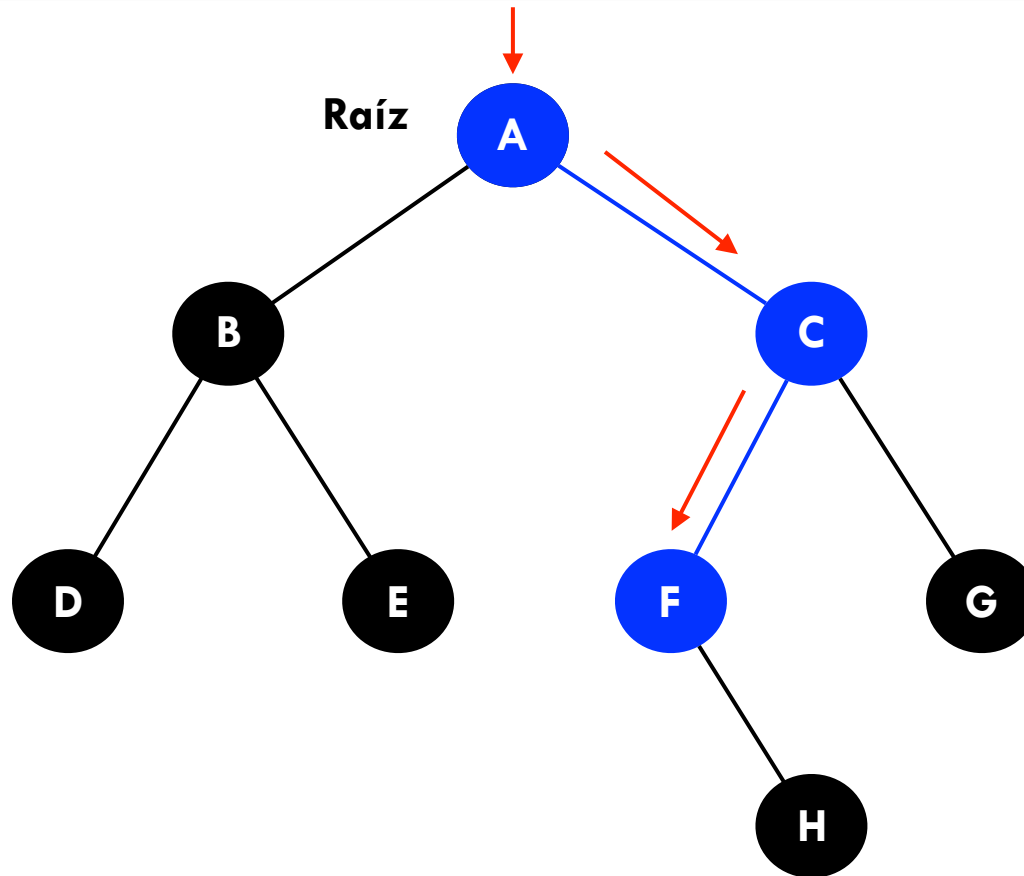
Caminho



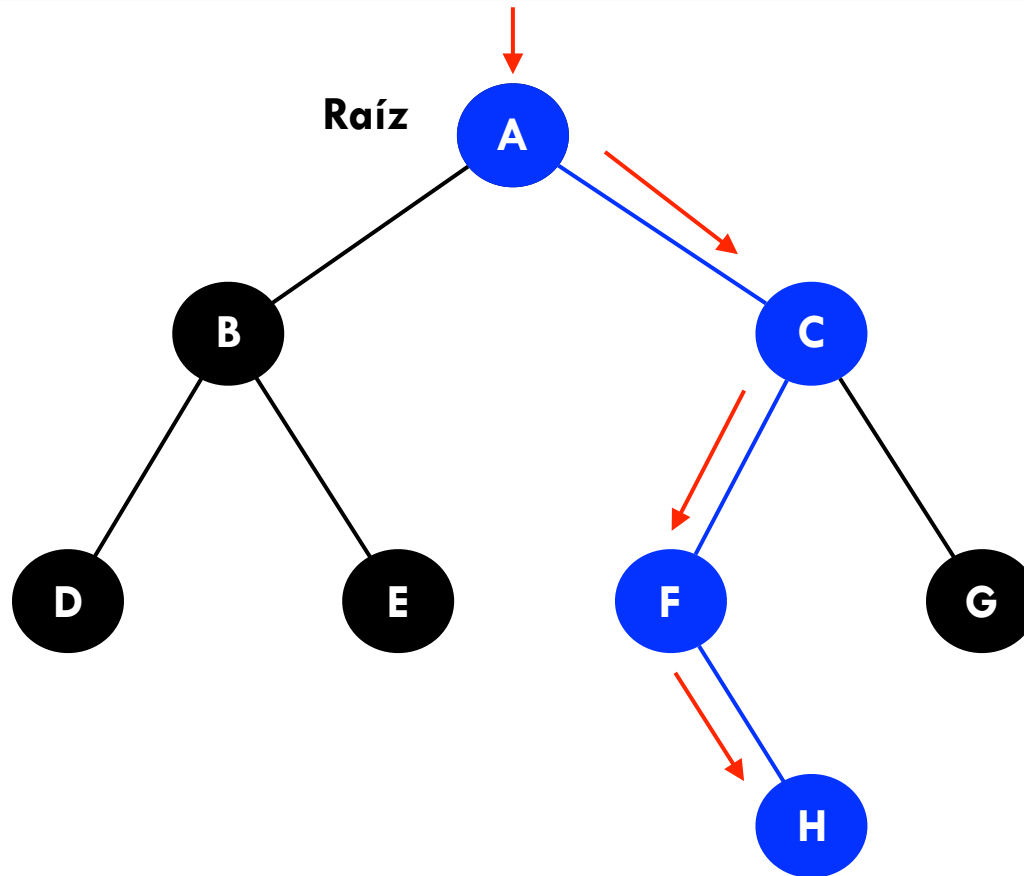
Caminho



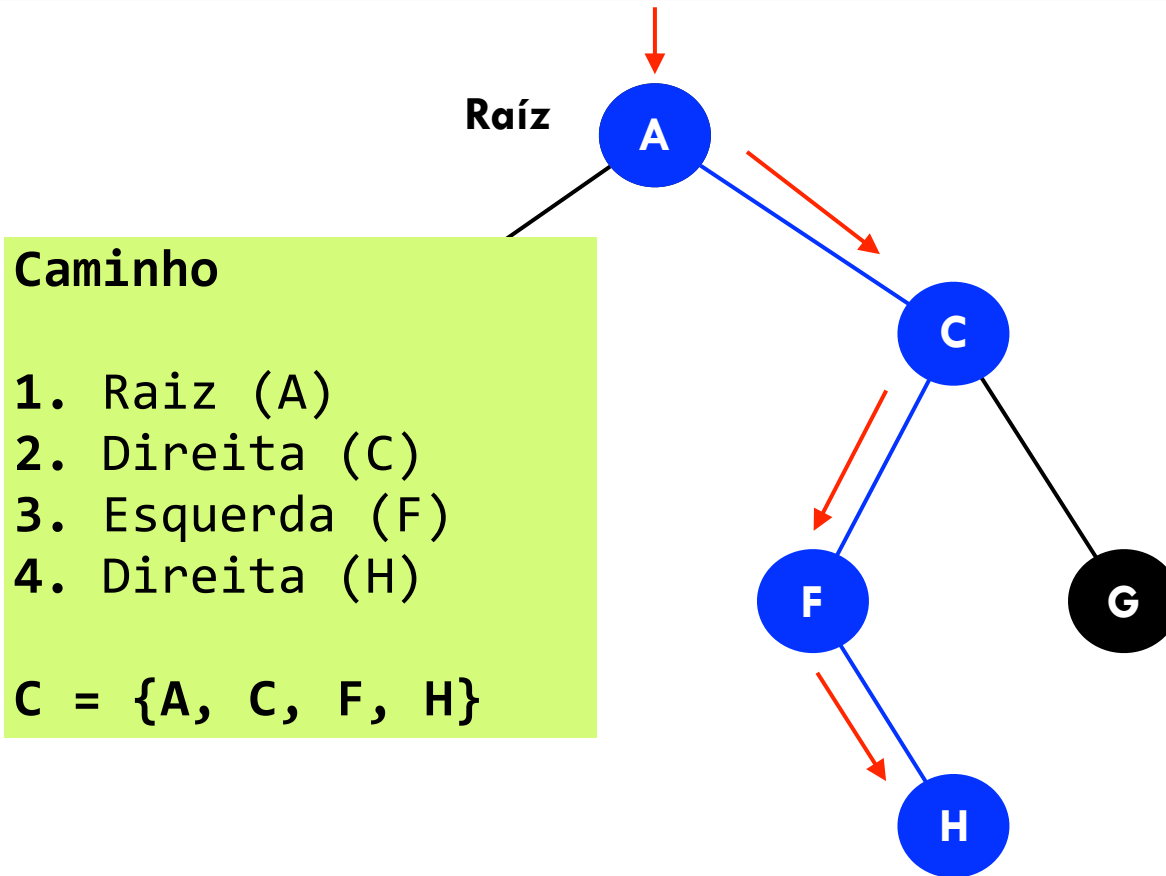
Caminho



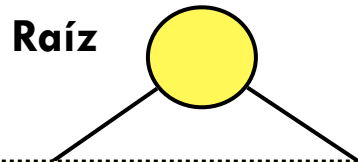
Caminho



Caminho



Altura



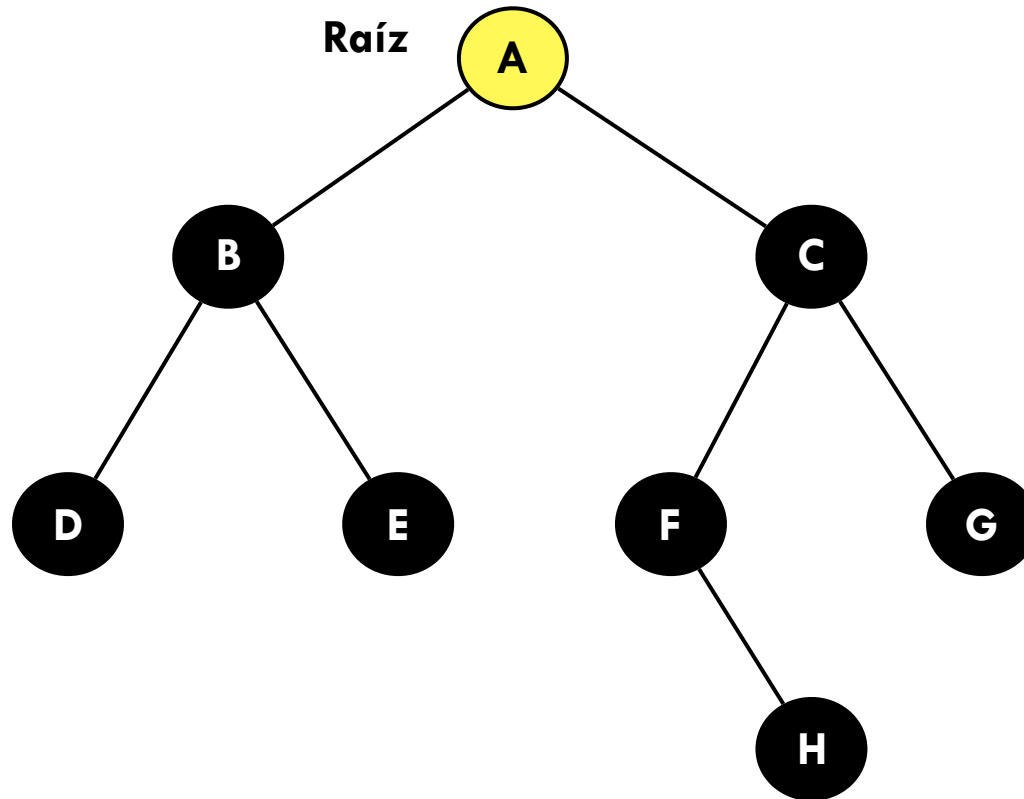
1. Altura de um nó X

* é a distância do caminho entre x e o seu descendente mais afastado

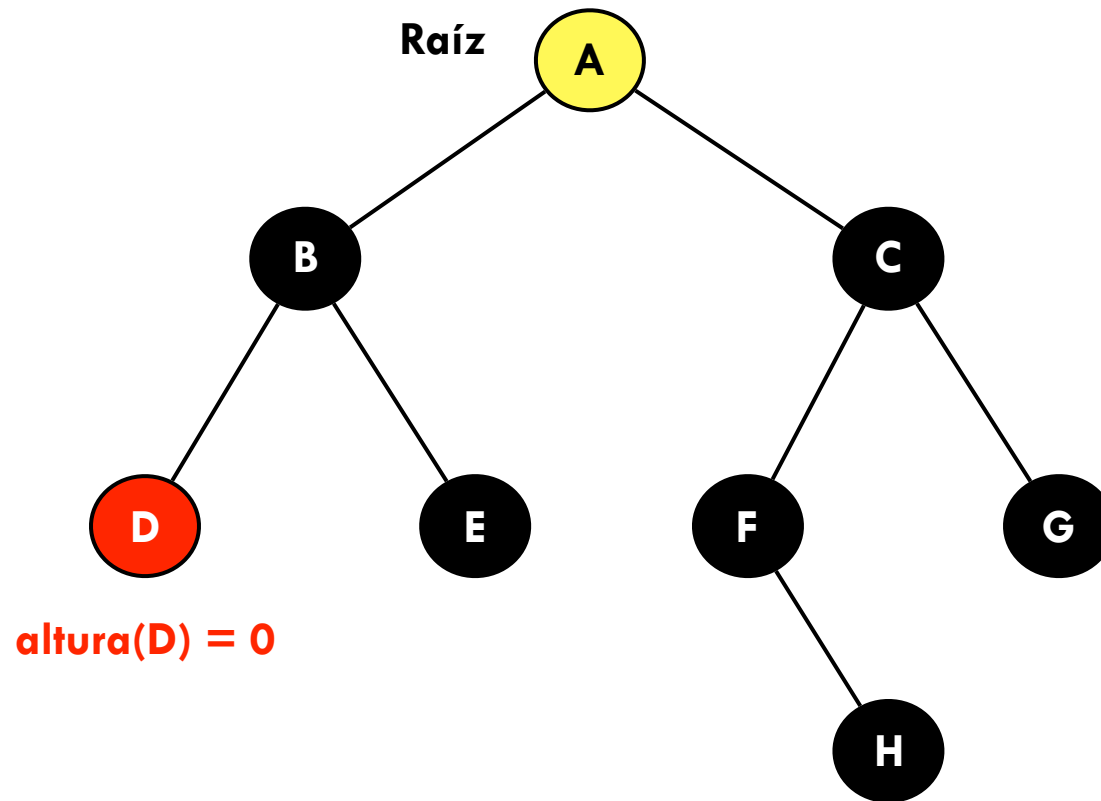
2. Altura de uma árvore

* é a altura da raíz da árvore

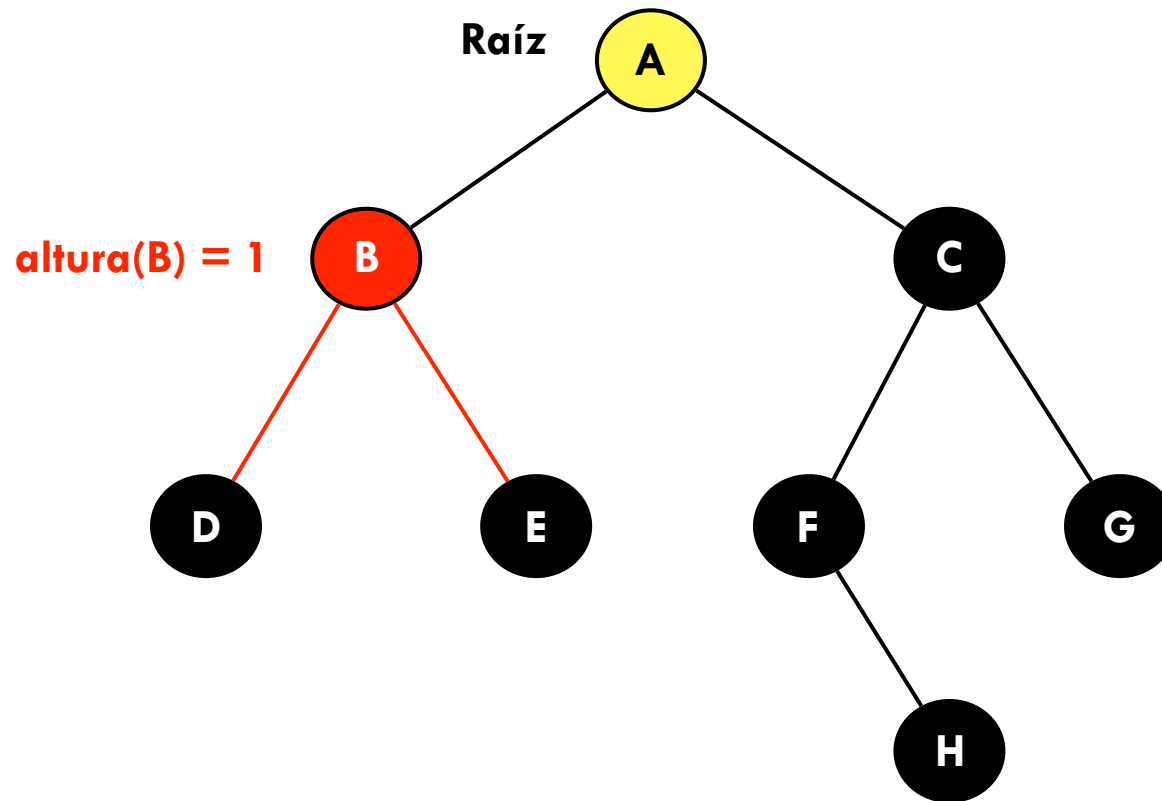
Altura



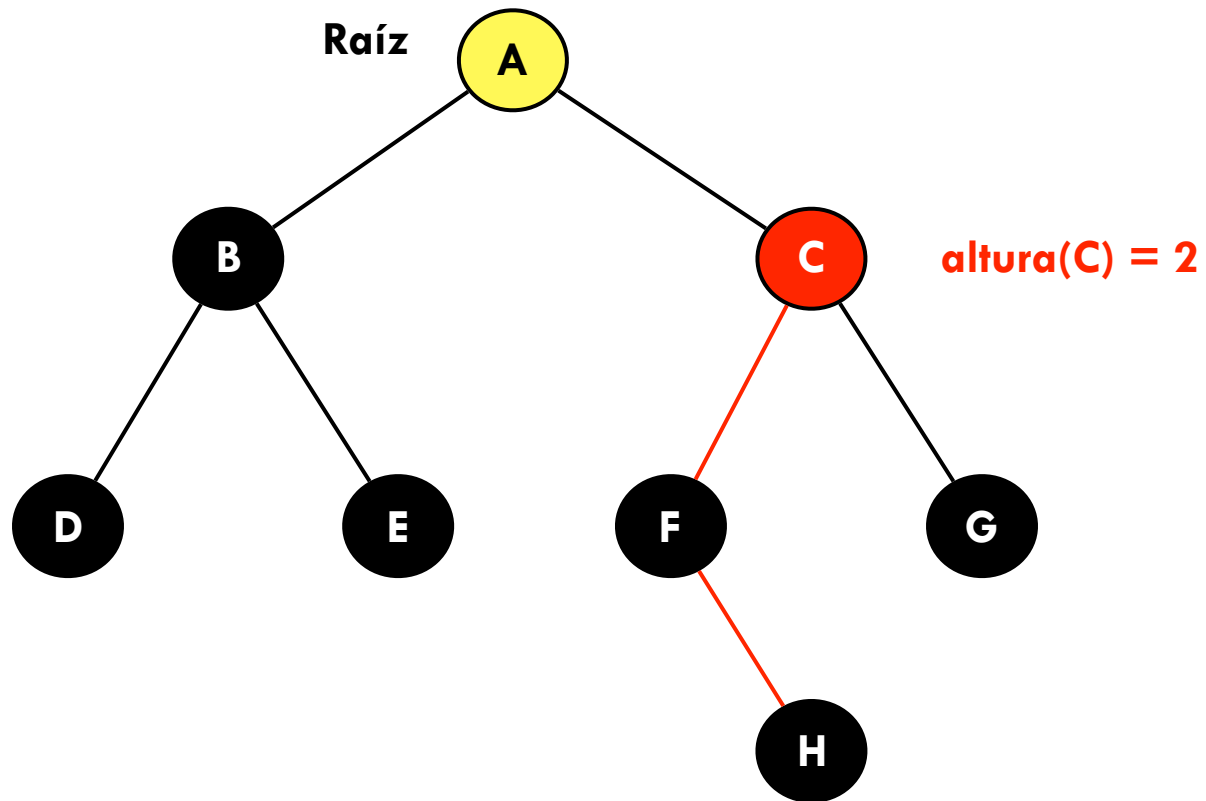
Altura



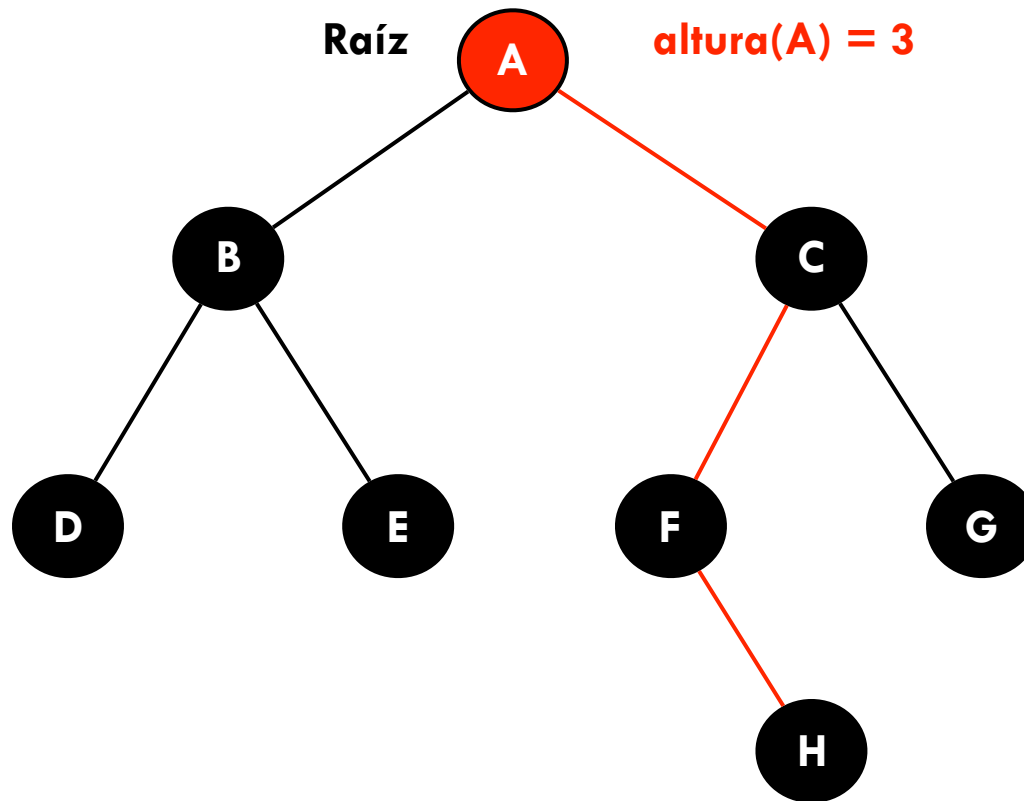
Altura



Altura



Altura



Exercício 01

- Desenhe uma árvore binária quem com 17 nós tenha a menor altura possível.
- Repita com a maior altura possível.

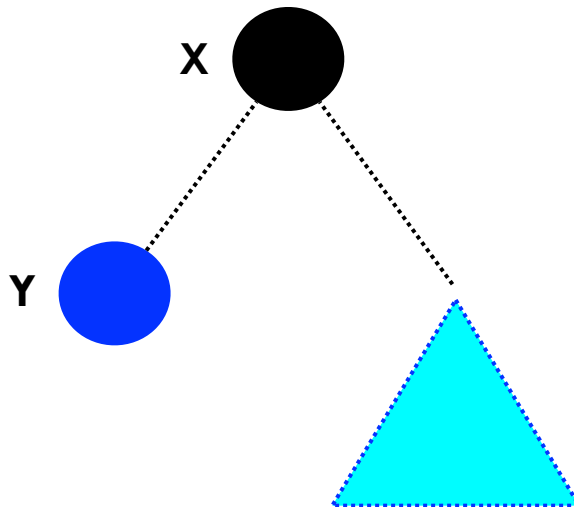
Árvores binárias de busca

□ Propriedade

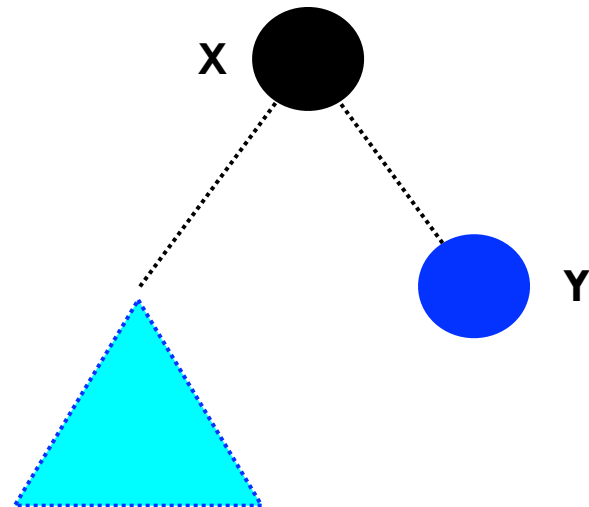
- Dados: nós de árvore X e Y
 - Se Y é um nó da subárvore esquerda de X ,
 - então $Y.chave \leq X.chave$.
 - Se Y é um nó da subárvore direita de X ,
 - então $Y.chave > X.chave$

Árvores binárias de busca

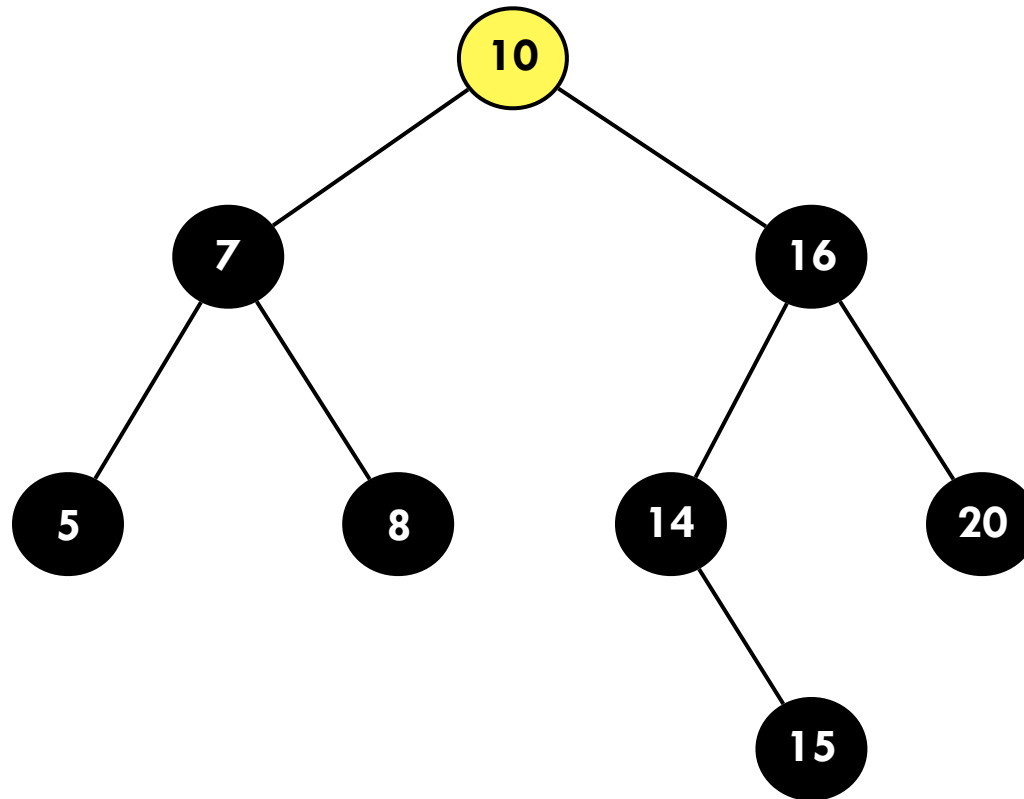
Se $y.chave \leq x.chave$



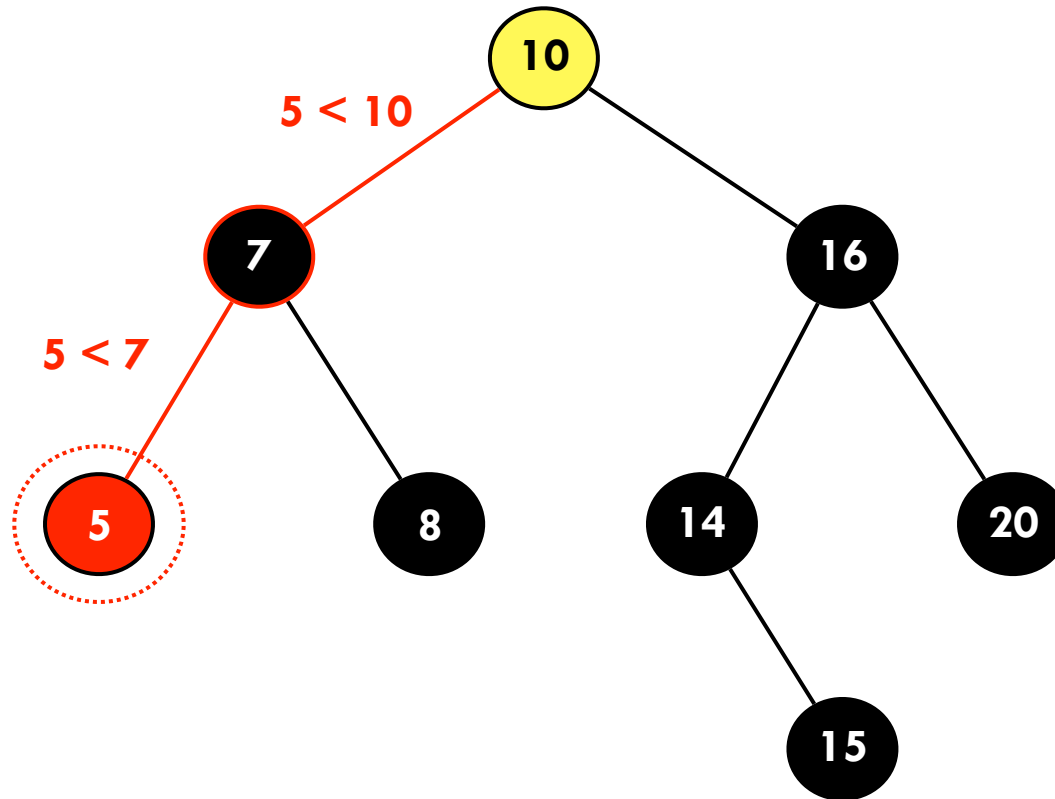
Se $y.chave > x.chave$



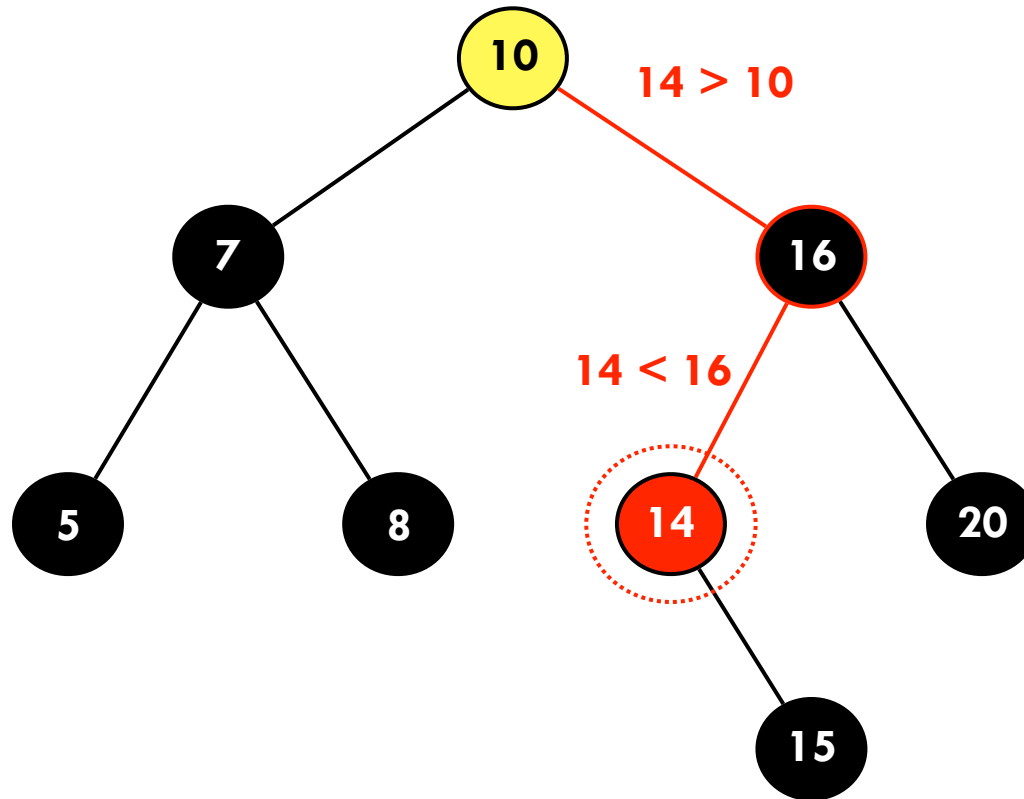
Exemplo



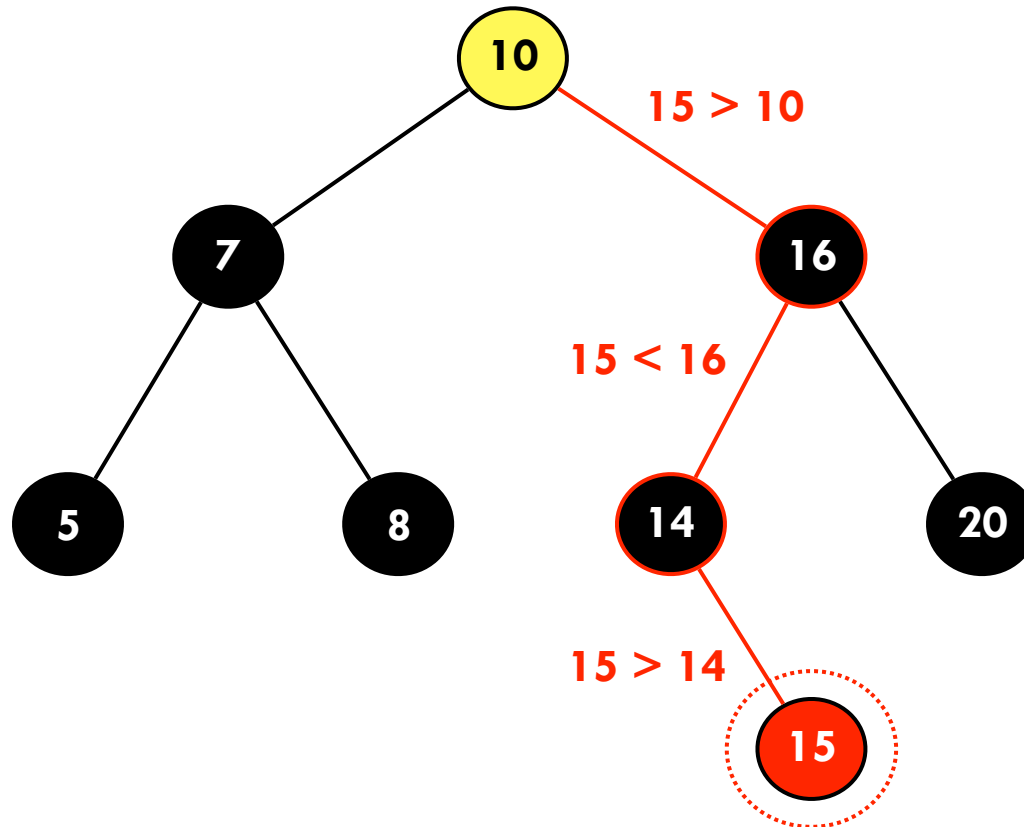
Exemplo



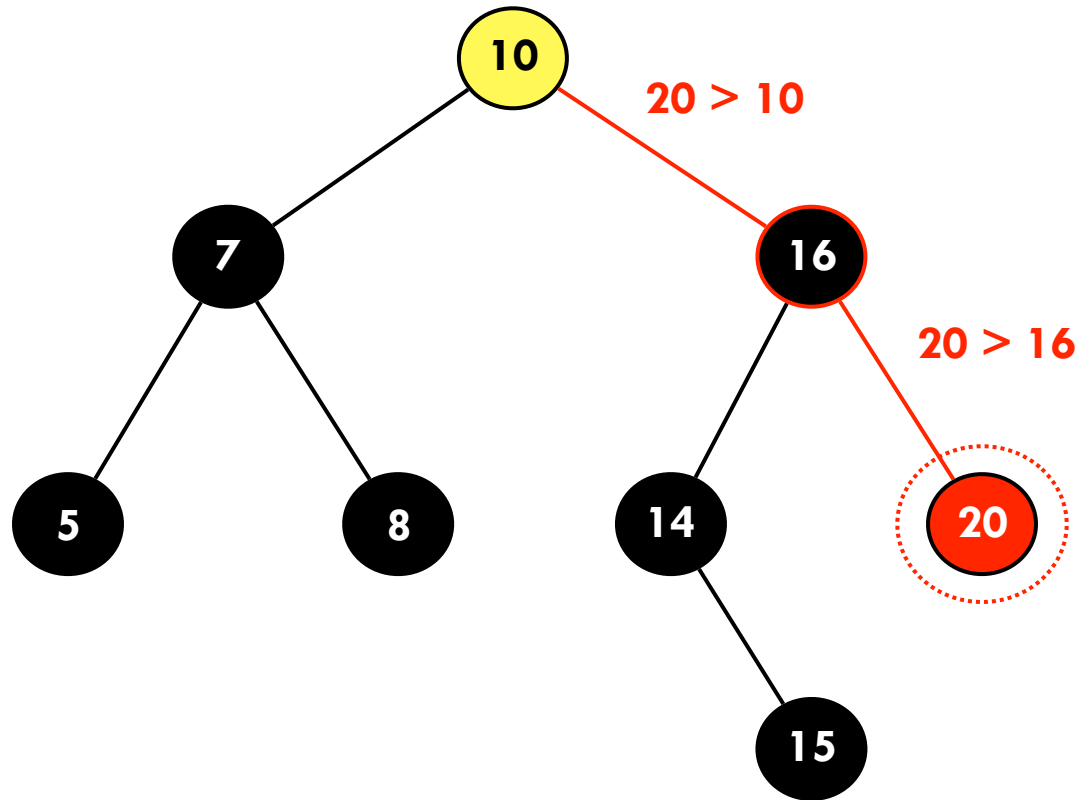
Exemplo



Exemplo



Exemplo



Exercício 02

- Gerar sequências aleatórias de números e montar as respectivas árvores.

Exercício 03

- Trace árvores binárias de busca de alturas 2, 3, 4, 5 e 6 para o conjunto de dados
 - $C = \{1, 4, 5, 10, 16, 17, 21\}$

Roteiro

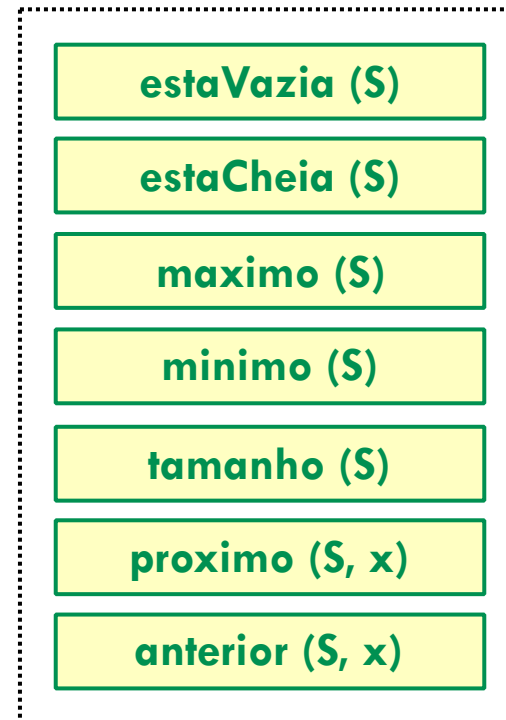
- 1 Introdução
- 2 Árvores Binárias
- 3 Propriedades e Definições
- 4 Inserção em Árvores Binárias
- 5 Pesquisa em Árvores Binárias
- 6 Referências

Operações em Árvores

Dada uma estrutura S , chave k , elemento x :



**Operações de
modificação**



**Operações adicionais
de consulta**

Operações em Árvores

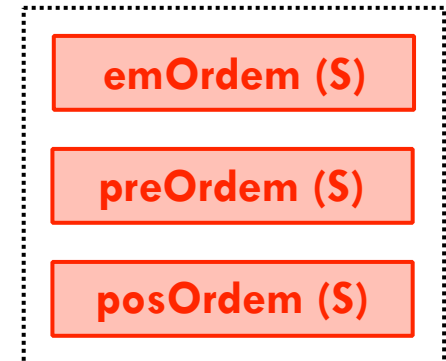
Dada uma estrutura **S**, chave **k**, elemento **x**:



**Operações de
modificação**

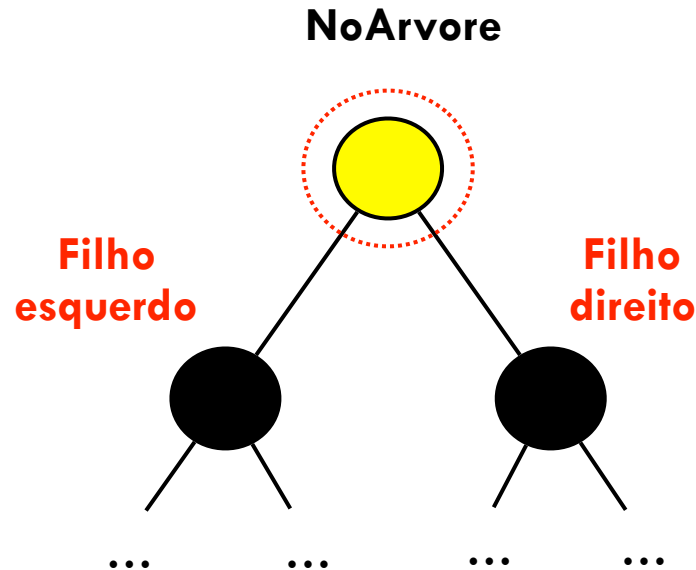


**Operações adicionais
de consulta**



**Percursos em
árvores**

Tipos de Árvore Binária



tipo **NoArvore**

Raiz

***NoArvore**

Arvore Binária



Elemento armazenado

***NoArvore (direita)**

***NoArvore (esquerda)**

Tipos de Árvore Binária

```
typedef struct {
    int chave;
} Objeto;

typedef struct NoArvore *PtrArvore;

typedef struct NoArvore{
    Objeto elemento;
    PtrArvore direita;
    PtrArvore esquerda;
} NoArvore;

/* Definir arvore binária na main */

PtrArvore raiz; /* como definir e
                  usar árvore */
```

Raiz

***NoArvore**

Arvore Binária

tipo **NoArvore**

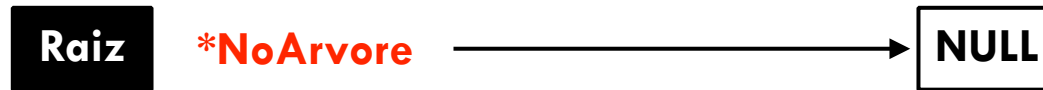


Elemento armazenado

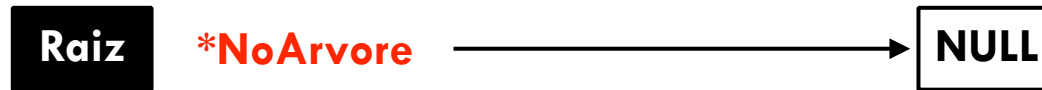
***NoArvore (direita)**

***NoArvore (esquerda)**

Inicialização



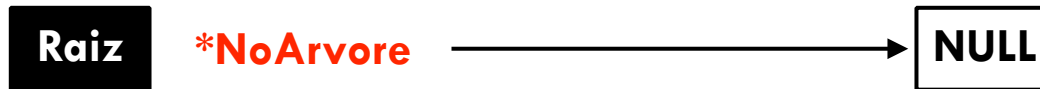
Inicialização



```
IniciaArvore (*arvore)  
1. *arvore = NULL;
```

```
EstaVazia (*arvore)  
1. return(*arvore == NULL);
```


Inicialização



```
IniciaArvore (*arvore)
1. *arvore = NULL;
```

```
EstaVazia (*arvore)
1. return(*arvore == NULL);
```

```
void iniciaArvore(PtrArvore *arvore){
    *arvore = NULL;
}
```

```
bool estaVazia(PtrArvore *arvore) {
    return(*arvore == NULL);
}
```

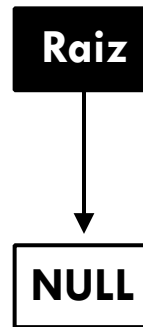
Inserção

- As operações de inserção e remoção provocam mudanças no conjunto dinâmico representado por uma árvore de busca binária
- A estrutura deve ser mudada, mas a propriedade de busca da árvore deve ser mantida
 - inserção → fácil
 - remoção → mais complicado

Inserção

Versão:

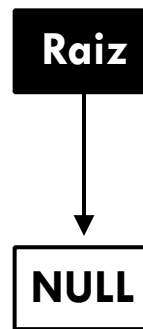
- Iterativa
- Recursiva



Inserção

Versão:

- Iterativa
- Recursiva

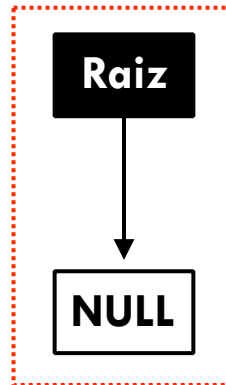


Inserir $x = 5$

Inserção

Versão:

- Iterativa
- Recursiva



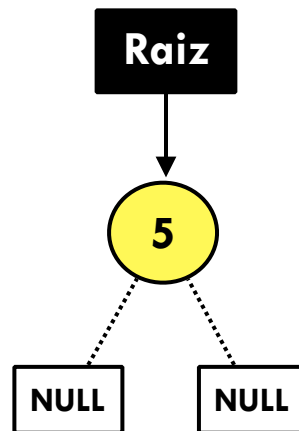
**Árvore
vazia**

Inserir $x = 5$

Inserção

Versão:

- Iterativa
- Recursiva

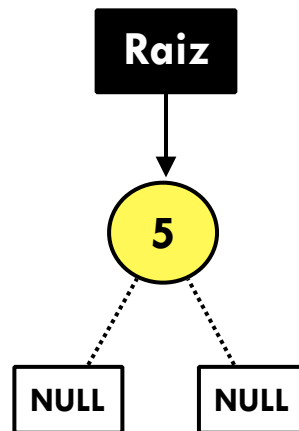


Inserir $x = 5$

Inserção

Versão:

- Iterativa
- Recursiva



Inserir $x = 5$

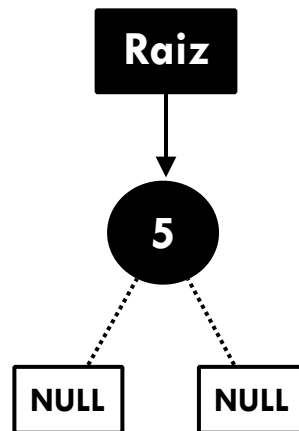
Se árvore está vazia:

1. alocar memória para um novo nó
2. atribuir à raiz o novo nó criado/alocado
3. `raiz->direita = raiz->esquerda = NULL;`
4. `raiz->elemento = x`
5. `return(true)`

Inserção

Versão:

- Iterativa
- Recursiva

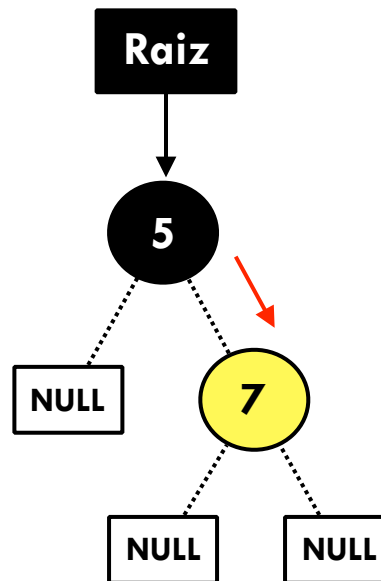


Inserir $x = 7$

Inserção

Versão:

- Iterativa
- Recursiva



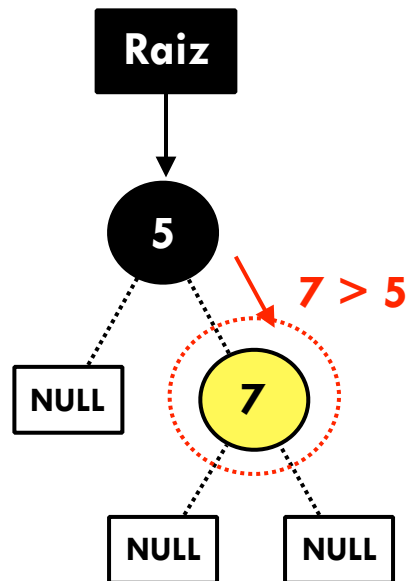
Inserir $x = 7$

Inserção

Versão:

- Iterativa
- Recursiva

Inserir $x = 7$



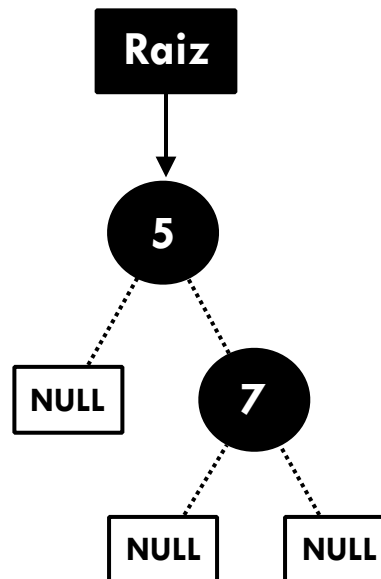
$7 >$ elemento na raiz (5)

- * deve ser inserido na sub-arvore **direita**
- * assim mantém-se a propriedade de busca

Inserção

Versão:

- Iterativa
- Recursiva



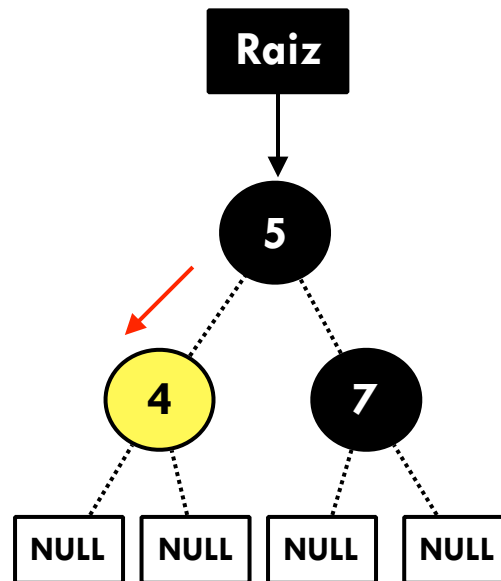
Inserir $x = 4$

Inserção

Versão:

- Iterativa
- Recursiva

Inserir $x = 4$

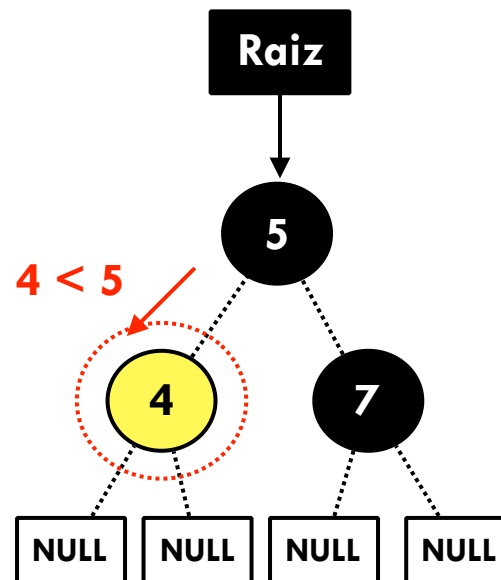


Inserção

Versão:

- Iterativa
- Recursiva

Inserir $x = 4$



$4 < \text{elemento na raiz (5)}$

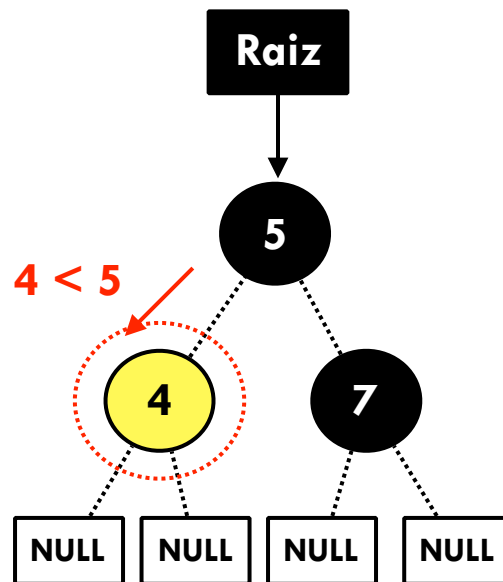
- * deve ser inserido na sub-arvore esquerda
- * assim mantém-se a propriedade de busca

Inserção

Versão:

- Iterativa
- Recursiva

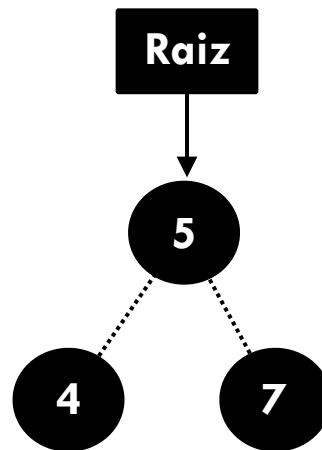
Inserir $x = 4$



Inserção

Versão:

- Iterativa
- Recursiva



Inserir $x = 3$

Inserir $x = 2$

Inserir $x = 6$

Inserir $x = 8$

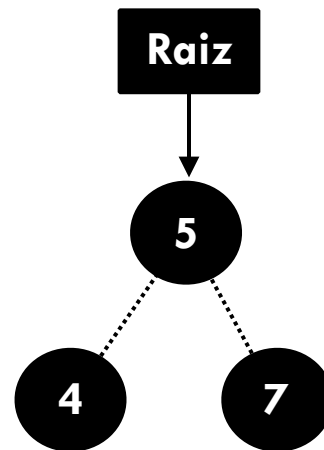
Inserir $x = 9$

Como fica a árvore?

Inserção

Versão:

- Iterativa
- Recursiva



Inserir $x = 3$

Inserir $x = 2$

Inserir $x = 6$

Inserir $x = 8$

Se $x.chave > chave$ do nó corrente
 acessar o filho da direita (subárvore direita)
Senão // $x.chave < chave$
 acessar o filho da esquerda (subárvore esquerda)
Quando encontrar um ponteiro == NULL
 inserir novo elemento

Inserção

Inserção (**PtrArvore** *arvore, **Objeto** x) // Versão recursiva

1. se *arvore == NULL // condição de parada da recursão
 1. (*arvore) = malloc(sizeof(NoArvore));
 2. (*arvore)->direita = (*arvore)->esquerda = NULL;
 3. (*arvore)->elemento = x;
 4. return(**true**);
2. se (*arvore)->elemento.chave == x.chave
 1. // não insere chave duplicada
 2. return (**false**);
3. se (*arvore)->elemento.chave > x.chave
 1. return(Inserção(&(*arvore)->esquerda, x));
4. senão
 1. return(Inserção(&(*arvore)->direita, x))

Inserção

Inserção (**PtrArvore** *arvore, **Objeto** x) // Versão recursiva

1. se *arvore == NULL // condição de parada da recursão
 1. (*arvore) = malloc(sizeof(NoArvore));
 2. (*arvore)->direita = (*arvore)->esquerda = NULL;
 3. (*arvore)->elemento = x;
 4. return(**true**);
2. se (*arvore)->elemento.chave == x.chave
 1. // não insere chave duplicada
 2. return (**false**);
3. se (*arvore)->elemento.chave > x.chave
 1. return(Inserção(&(*arvore)->esquerda, x));
4. senão
 1. return(Inserção(&(*arvore)->direita, x))

**Percorre a árvore
até achar a
posição
de inserção**

Exercício 04

- Implementar Tipos Abstratos de Árvore Binária e a operação de inserção de elementos.
- Se preciso, implementar funções auxiliares para checar se a estrutura está correta.

Exercício 05

- Como escrever uma versão iterativa para a função de inserção em uma árvore binária?

Roteiro

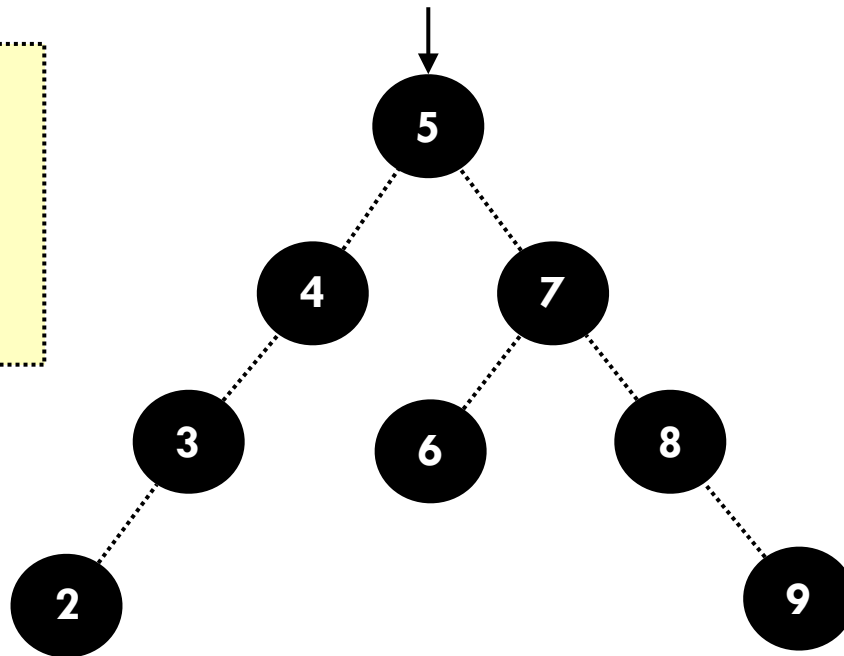
- 1 Introdução
- 2 Árvores Binárias
- 3 Propriedades e Definições
- 4 Inserção em Árvores Binárias
- 5 Pesquisa em Árvores Binárias
- 6 Referências

Percursos

Raiz

Útil para:

- Imprimir
- Consultar
- Remover
- Inserir

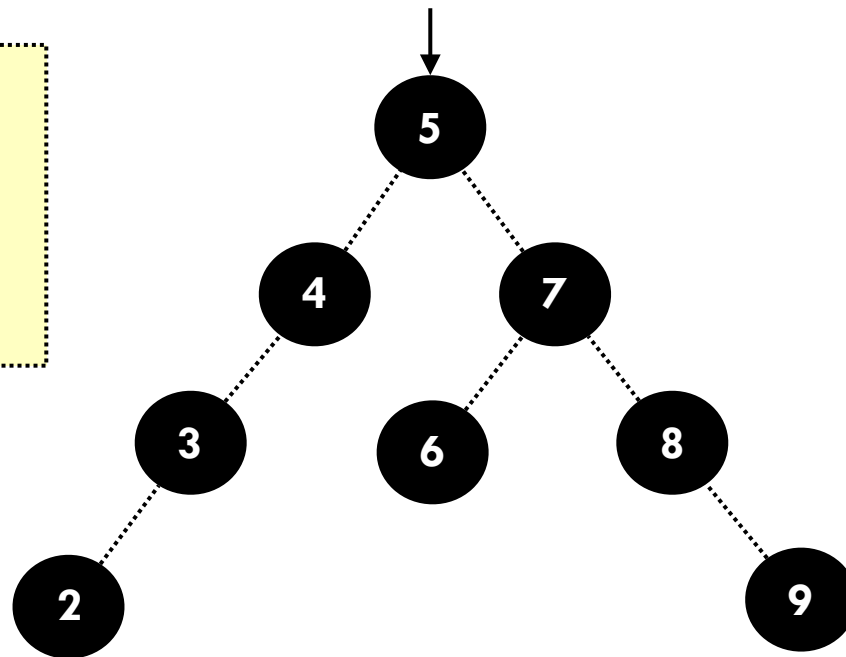


Percursos

Raiz

Útil para:

- Imprimir
- Consultar
- Remover
- Inserir

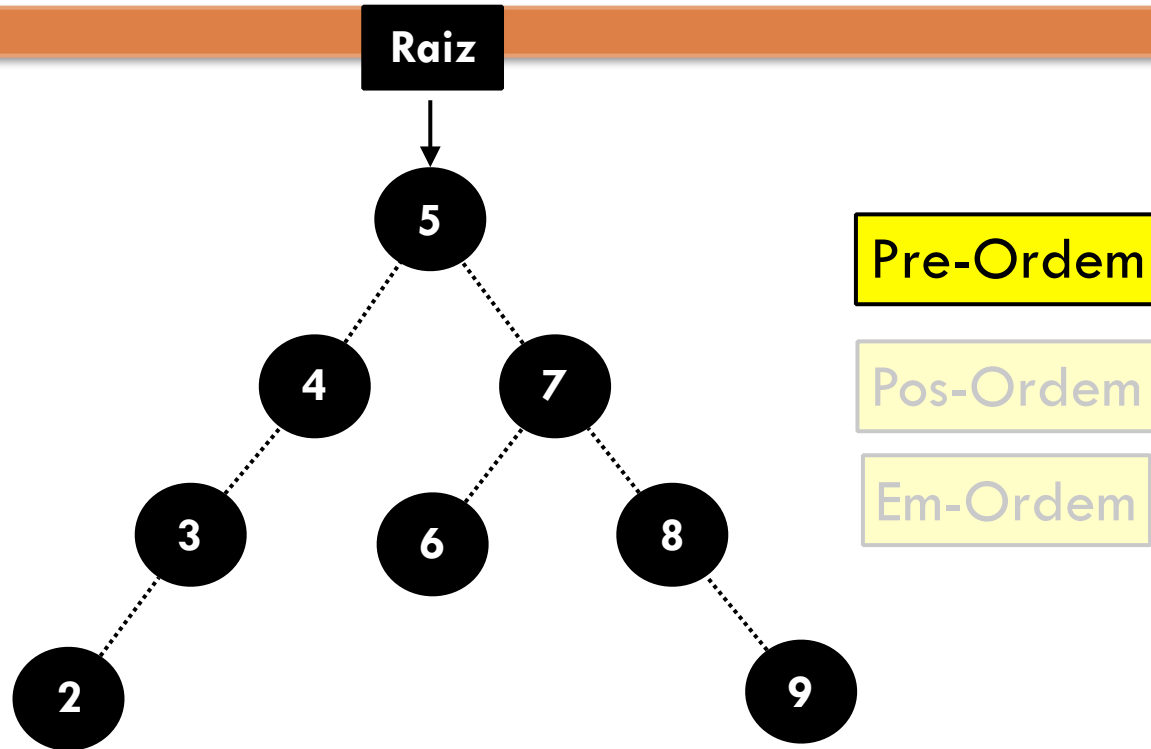


Pre-Ordem

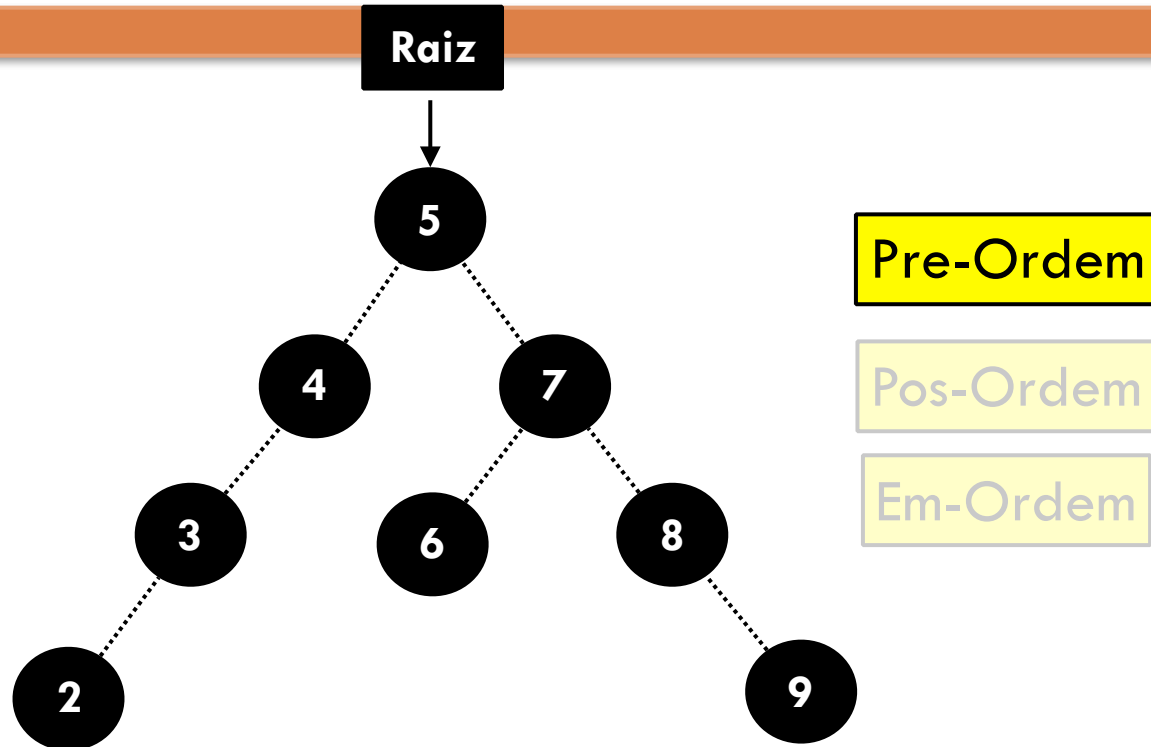
Pos-Ordem

Em-Ordem

Percurso: Pre-ordem



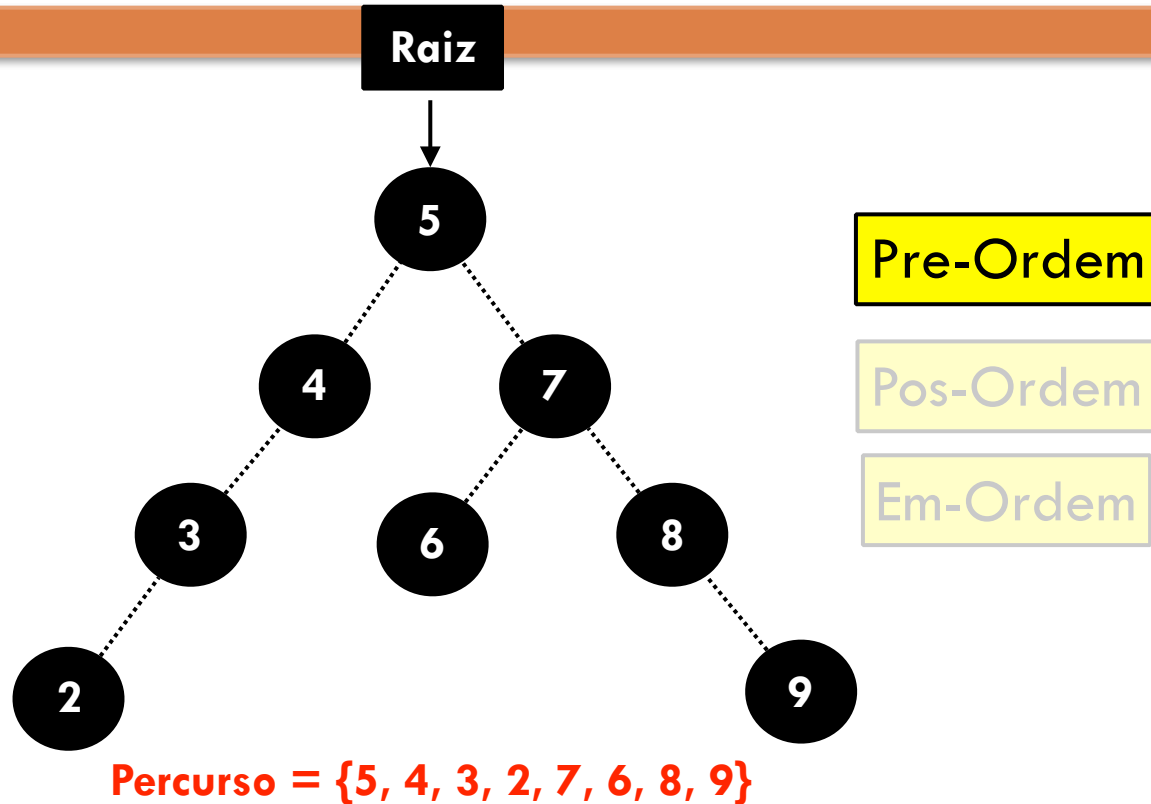
Percurso: Pre-ordem



Pre-Ordem (*arvore)

1. imprime o valor do nó corrente // (*arvore)
2. Pre-Ordem((*arvore)->esquerda)
3. Pre-Ordem((*arvore)->direita)

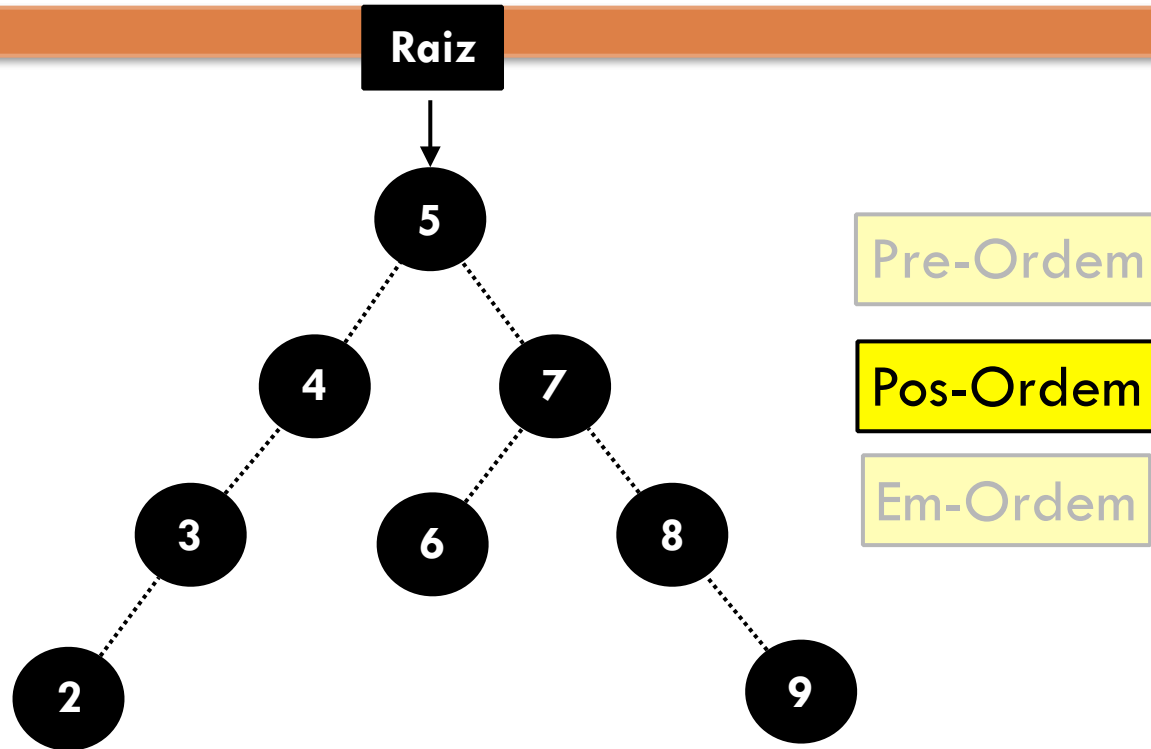
Percurso: Pre-ordem



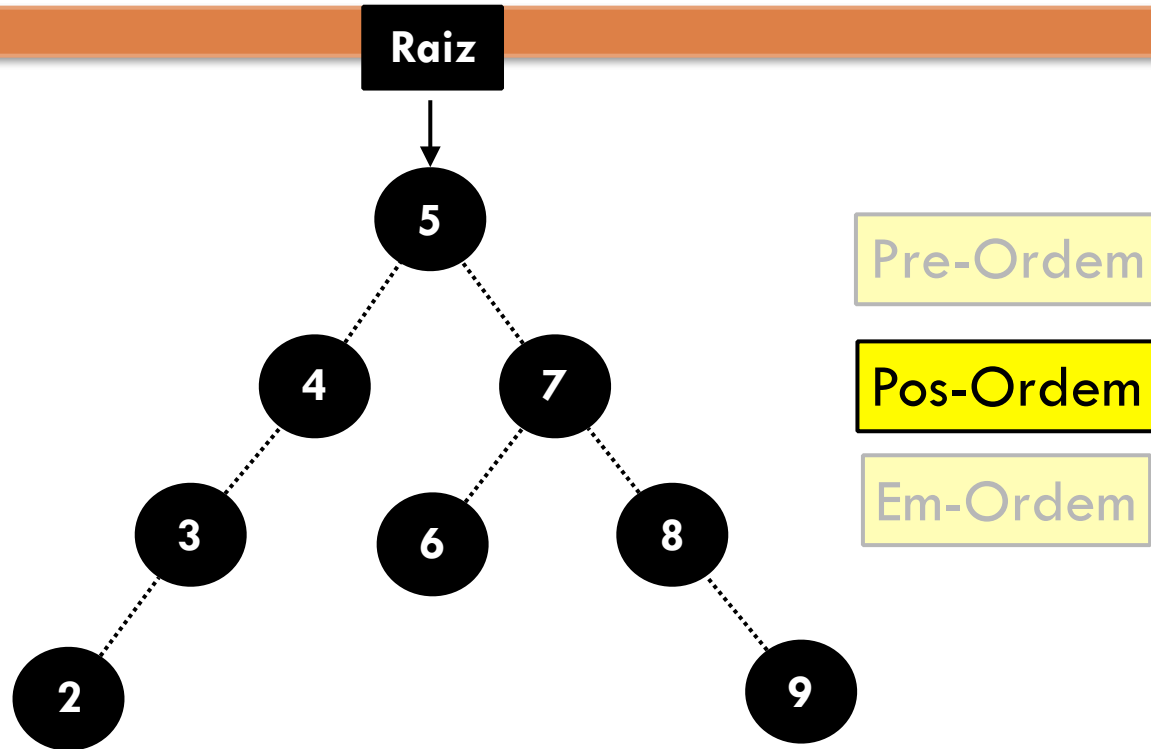
Pre-Ordem (*arvore)

1. imprime o valor do nó corrente // (*arvore)
2. Pre-Ordem((*arvore)->esquerda)
3. Pre-Ordem((*arvore)->direita)

Percurso: Pos-ordem



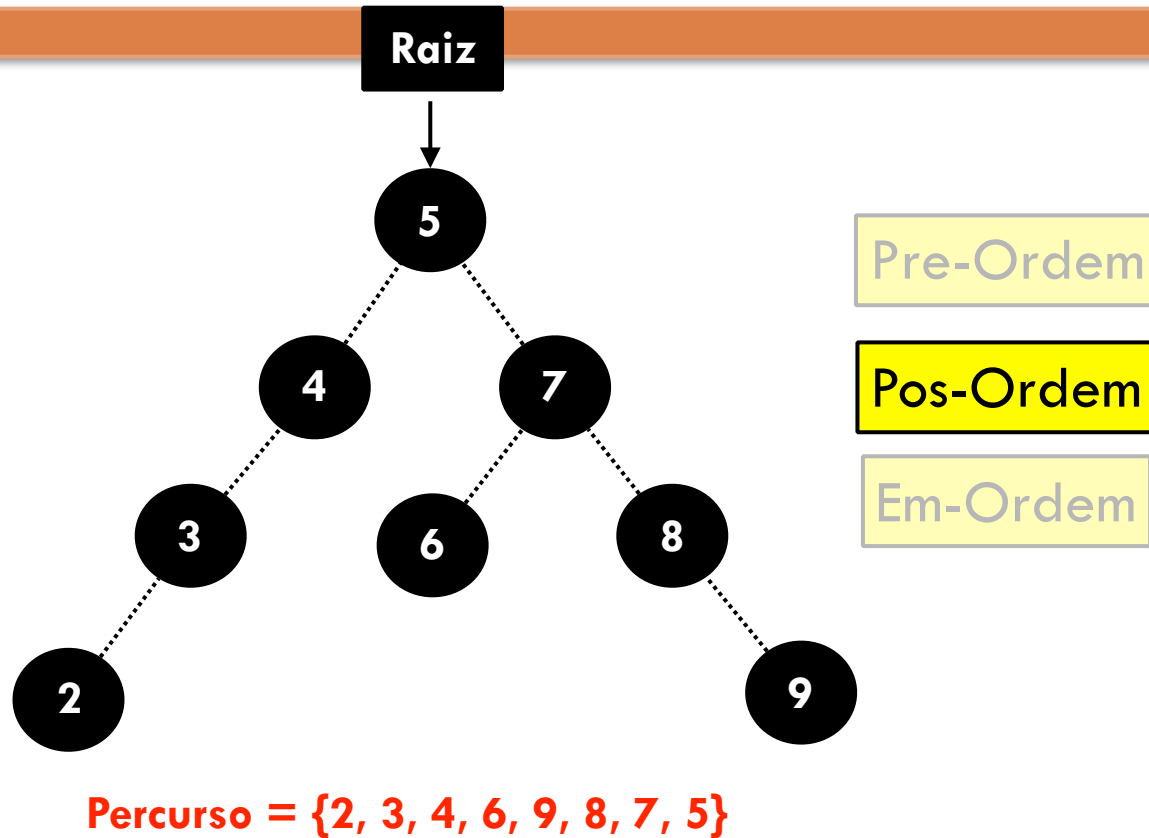
Percurso: Pos-ordem



Pos-Ordem (*arvore)

1. Pos-Ordem((*arvore)->esquerda)
2. Pos-Ordem((*arvore)->direita)
3. imprime o valor do nó corrente // (*arvore)

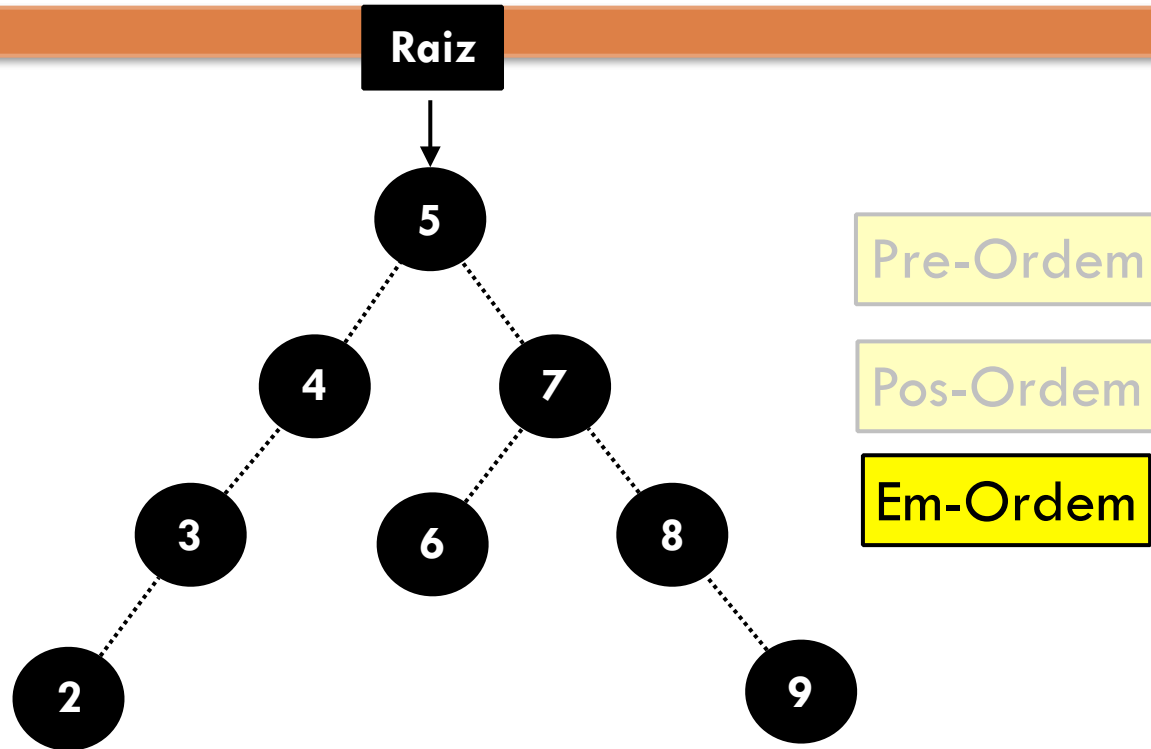
Percurso: Pos-ordem



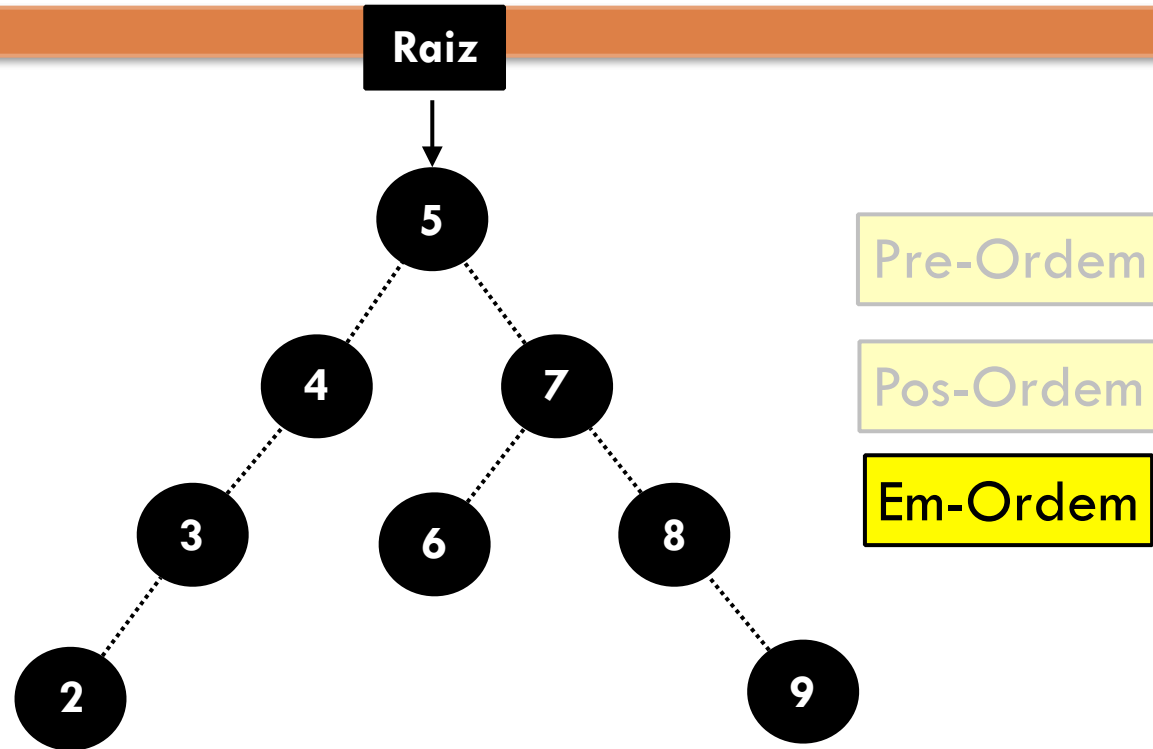
Pos-Ordem (*arvore)

1. Pos-Ordem((*arvore)->esquerda)
2. Pos-Ordem((*arvore)->direita)
3. imprime o valor do nó corrente // (*arvore)

Percursos



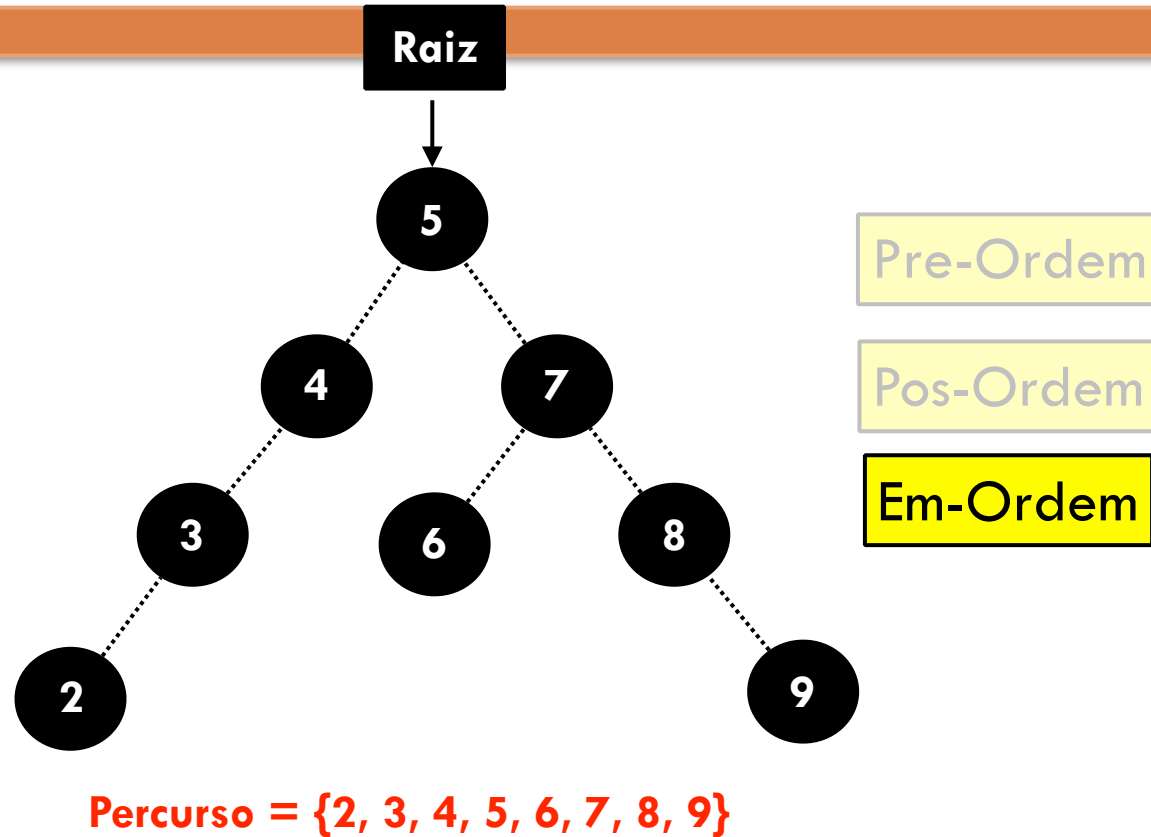
Percursos



Em-Ordem (*arvore)

1. Em-Ordem((*arvore)->esquerda)
2. imprime o valor do nó corrente // (*arvore)
3. Em-Ordem((*arvore)->direita)

Percursos



Em-Ordem (*arvore)

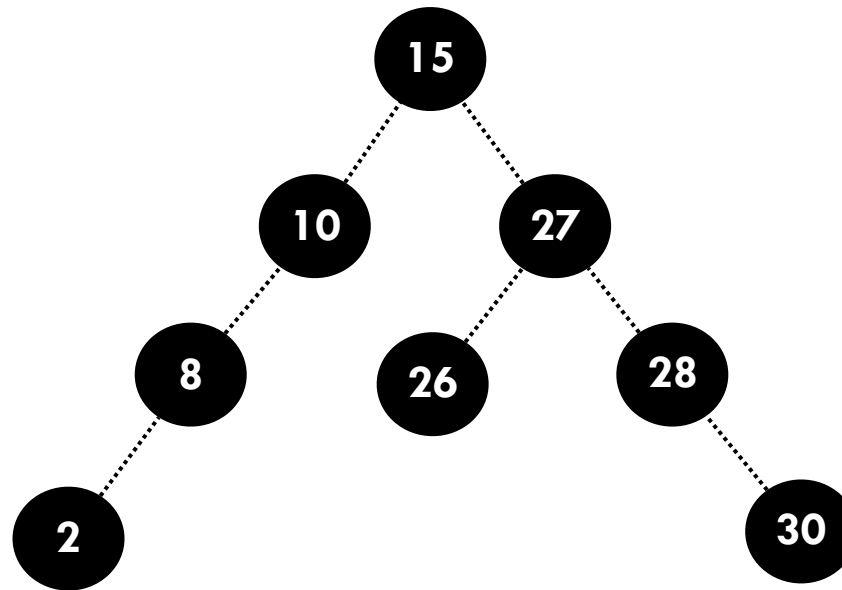
1. Em-Ordem((*arvore)->esquerda)
2. imprime o valor do nó corrente // (*arvore)
3. Em-Ordem((*arvore)->direita)

Consultas / Pesquisa

- Recebe o ponteiro da raiz (*arvore) e uma chave de consulta chave (int)
 - se existir = TRUE
 - se não existir = FALSE

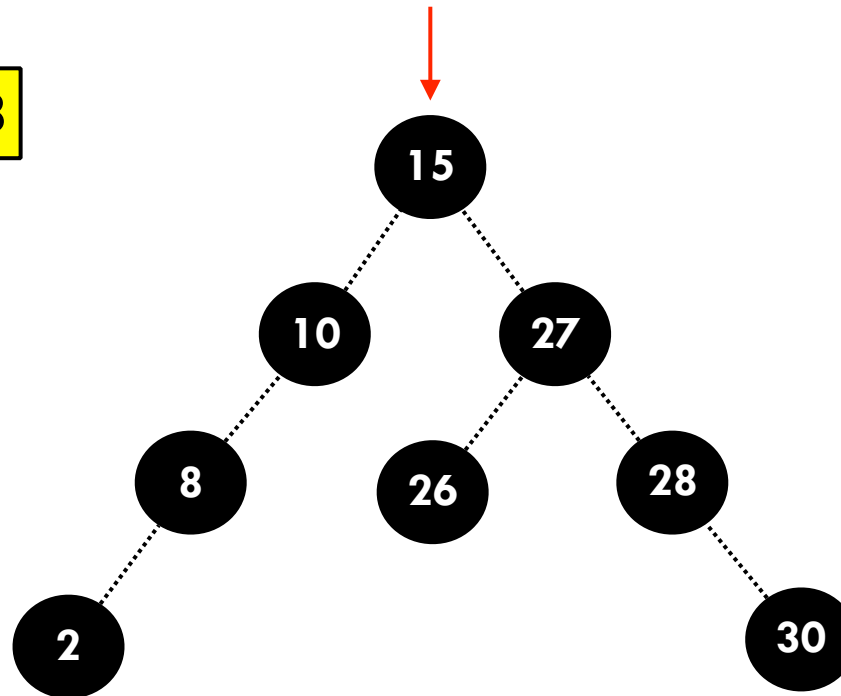
- acessar raiz
 - percorrer a arvore usando a propriedade de busca
 - encontrar a posição do elemento no arranjo

Consultas / Pesquisa



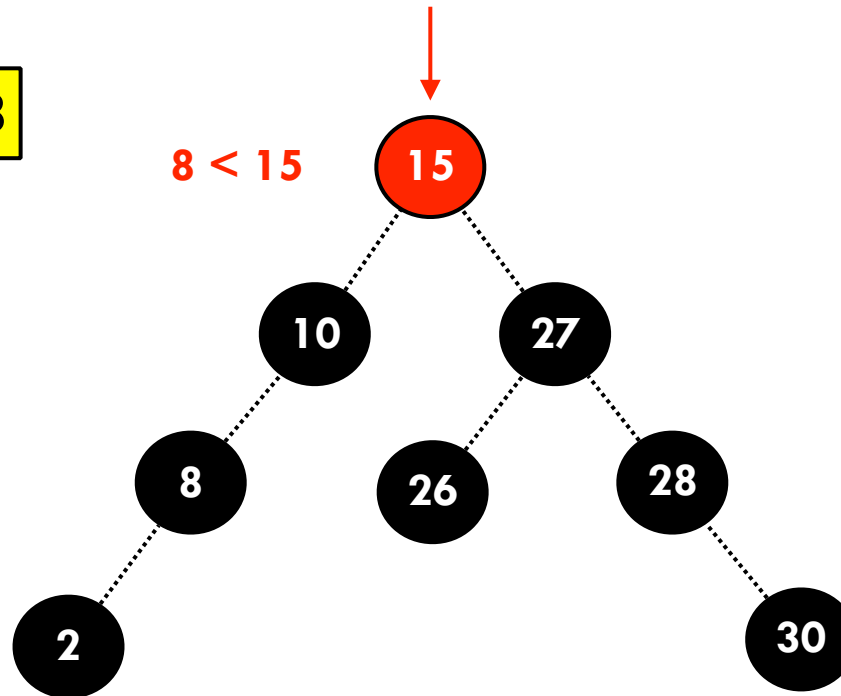
Consultas / Pesquisa

Procurar $x = 8$



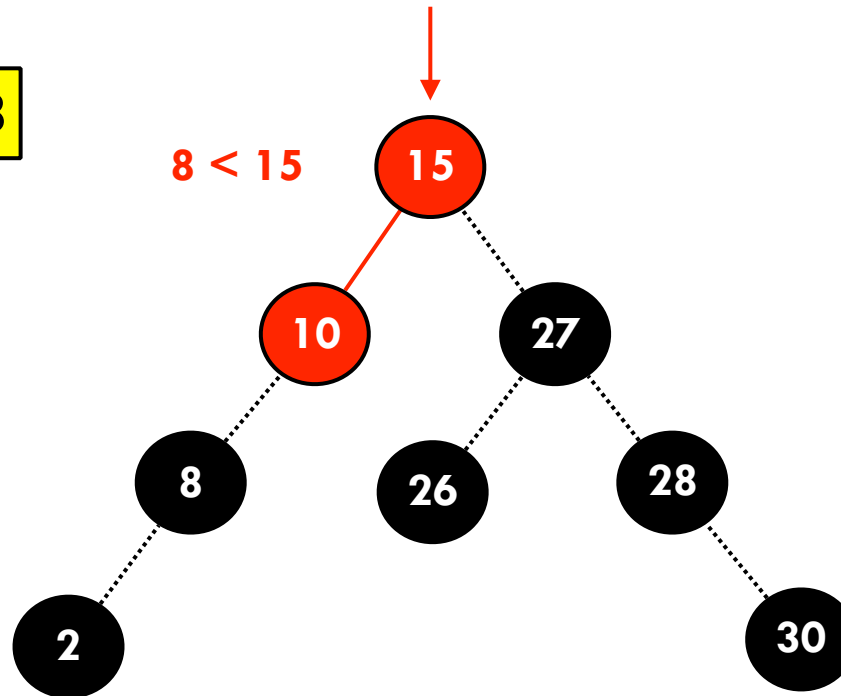
Consultas / Pesquisa

Procurar $x = 8$



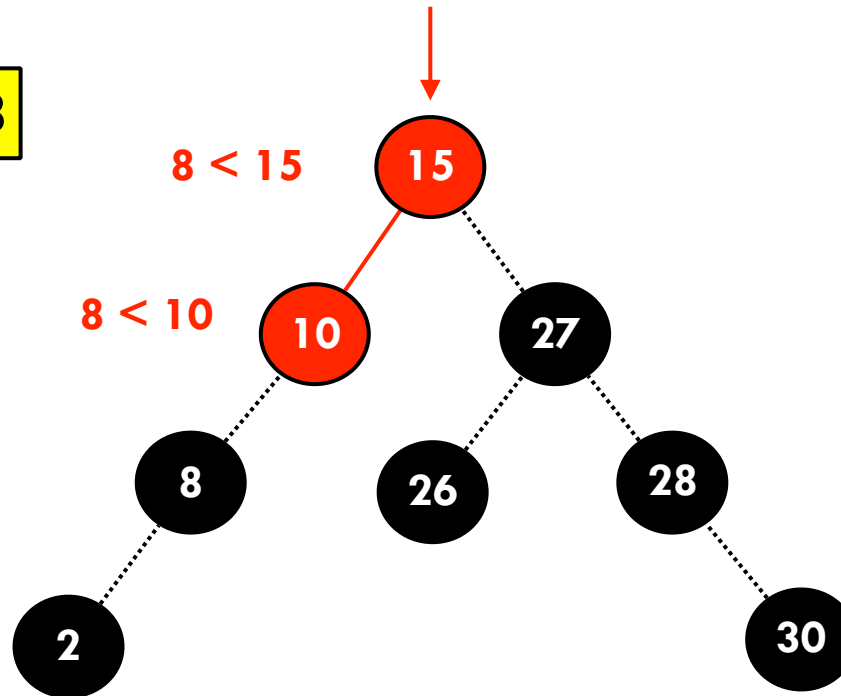
Consultas / Pesquisa

Procurar $x = 8$



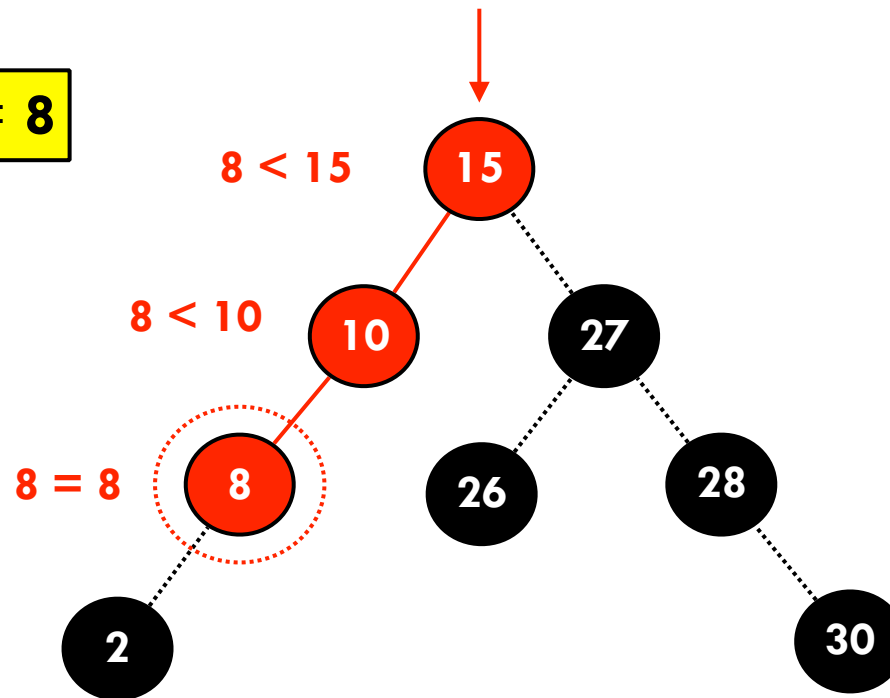
Consultas / Pesquisa

Procurar $x = 8$



Consultas / Pesquisa

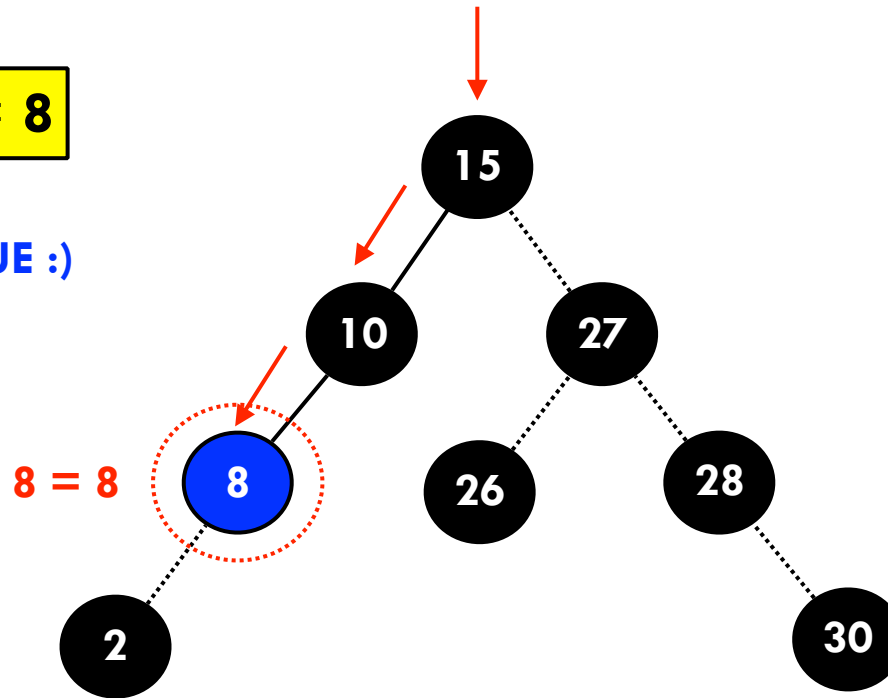
Procurar $x = 8$



Consultas / Pesquisa

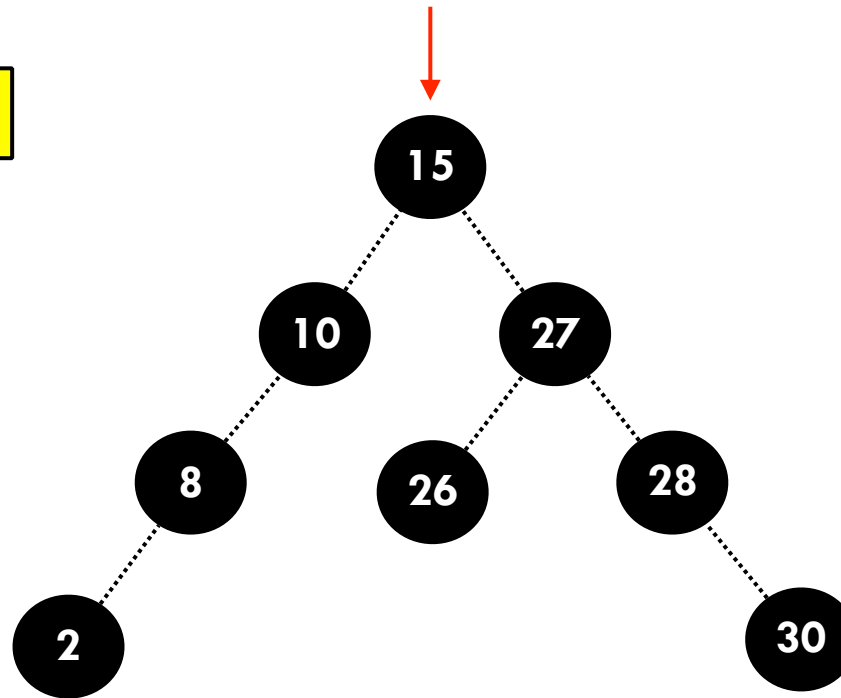
Procurar $x = 8$

Resultado: **TRUE** :)



Consultas / Pesquisa

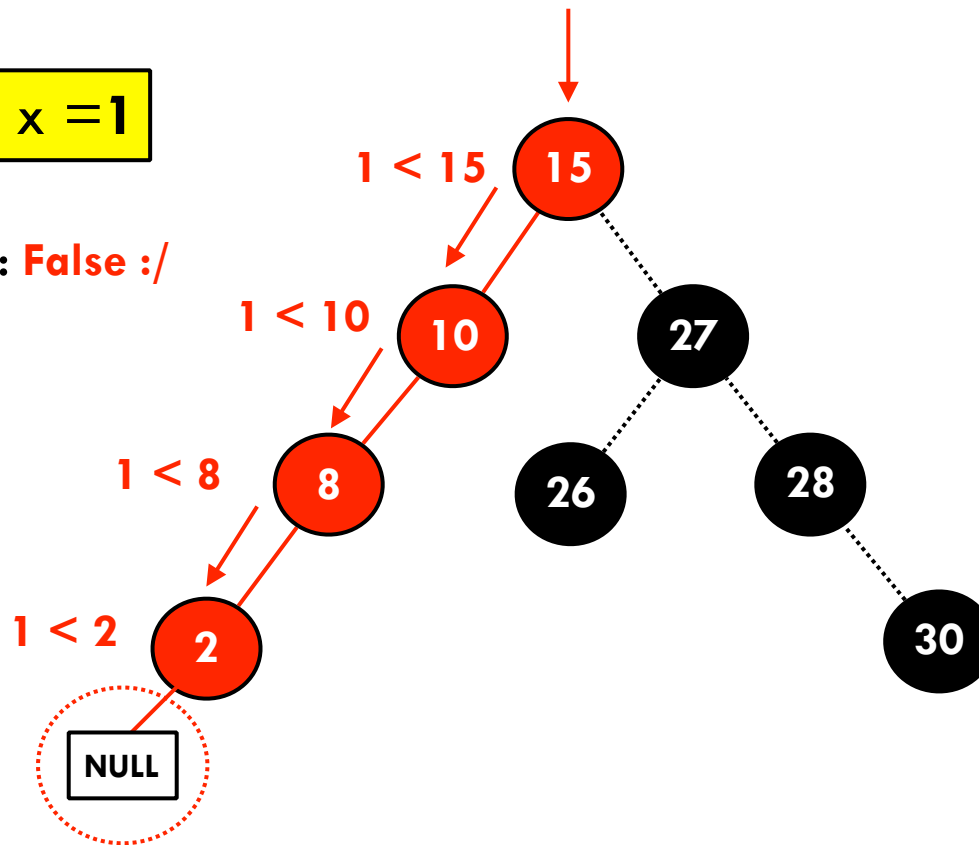
Procurar $x = 1$



Consultas / Pesquisa

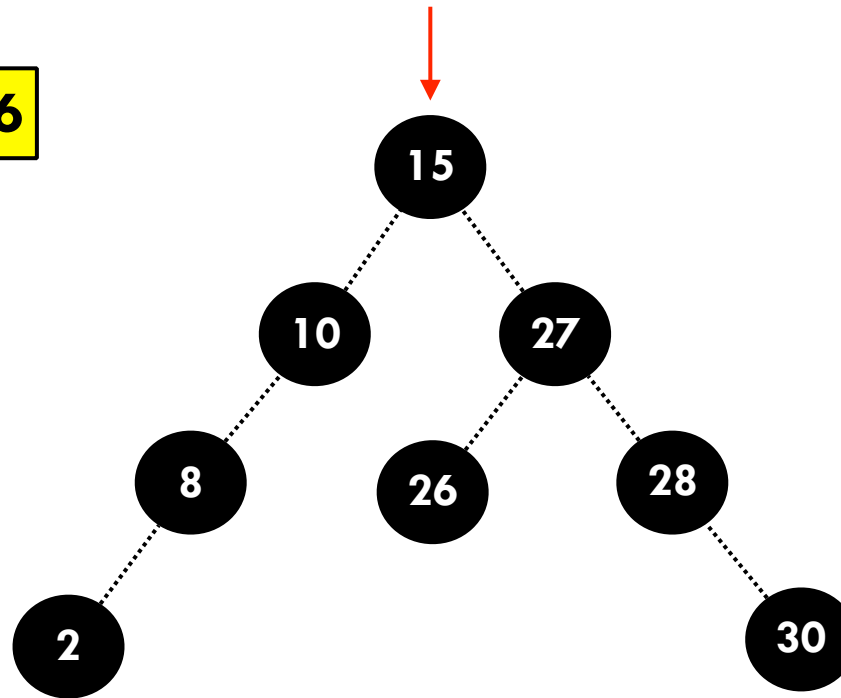
Procurar $x = 1$

Resultado: **False** :/



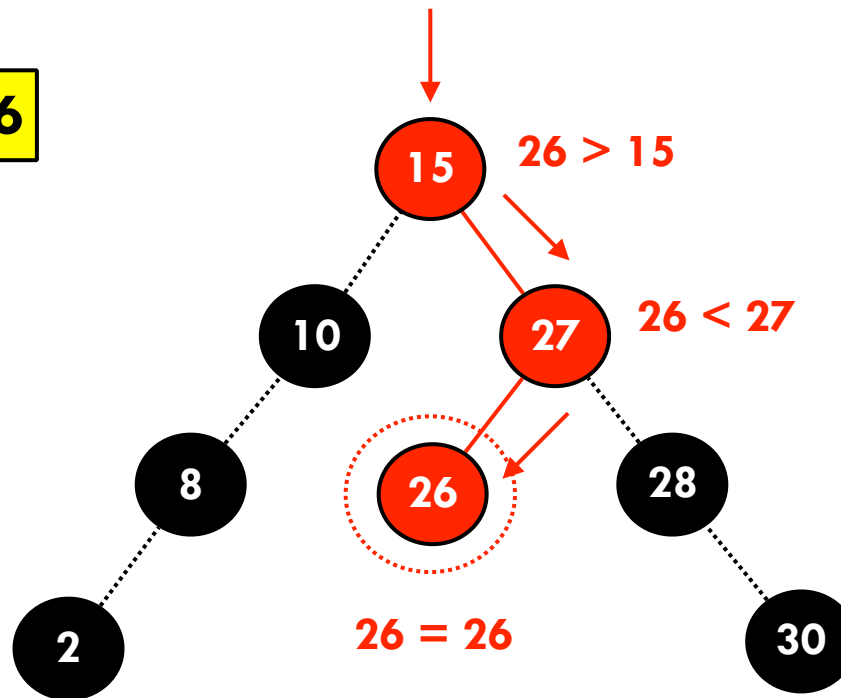
Consultas / Pesquisa

Procurar $x = 26$



Consultas / Pesquisa

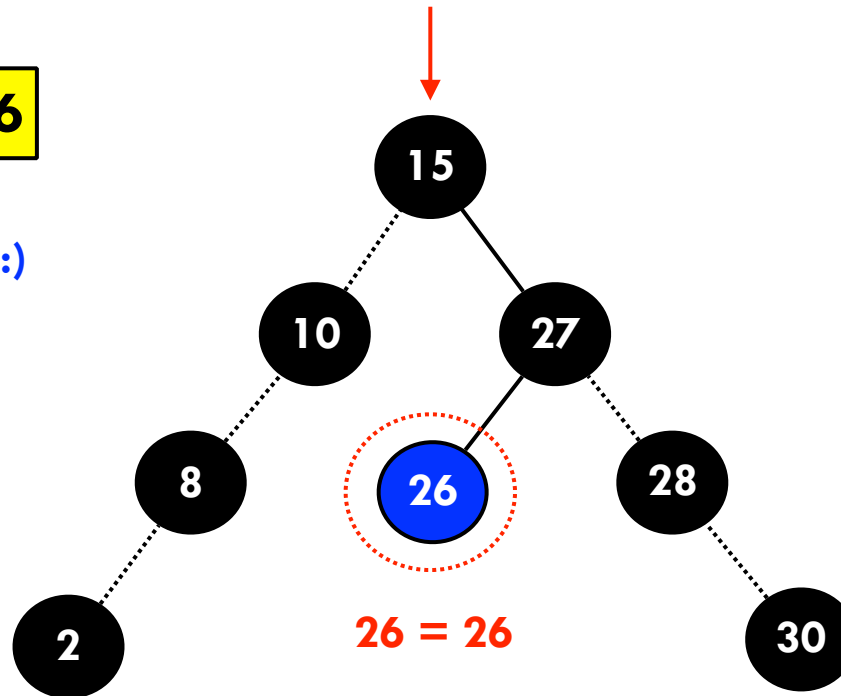
Procurar $x = 26$



Consultas / Pesquisa

Procurar $x = 26$

Resultado: **TRUE** :)



Consultas / Pesquisa

- Recebe o ponteiro da raiz (*arvore) e uma chave de consulta chave (int)
 - se existir = TRUE
 - se não existir = FALSE

```
bool procurar(PtrArvore *arvore, int chave, Objeto *ret) {  
    // não achou o elemento  
    se (*arvore == NULL) return (false)  
  
    // achou o elemento  
    se ((*arvore)->elemento.chave == chave) return (true)  
  
    se (chave < (*arvore)->elemento.chave)  
        procurar na subárvore esquerda // (*arvore)->esquerda  
    senão // chave > (*arvore)->elemento.chave  
        procurar na subárvore direita // (*arvore)->direita  
}
```

Consultas / Pesquisa

- Recebe o ponteiro da raiz (*arvore) e uma chave de consulta chave (int)
 - se existir = TRUE
 - se não existir = FALSE

```
bool procurar(PtrArvore *arvore, int chave, Objeto *ret) {  
    // não achou o elemento  
    se (*arvore == NULL) return (false)  
  
    // achou o elemento  
    se ((*arvore)->elemento.chave == chave) return (true)  
  
    se (chave < (*arvore)->elemento.chave)  
        procurar na subárvore esquerda // (*arvore)->esquerda  
    senão // chave > (*arvore)->elemento.chave  
        procurar na subárvore direita // (*arvore)->direita  
}
```

Recursão

Exercício 06

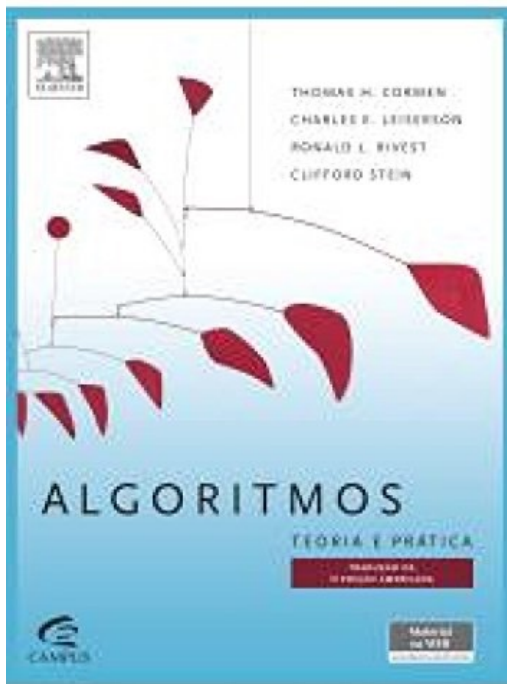


- Implementar a função de busca/pesquisa/consulta para árvores binárias.

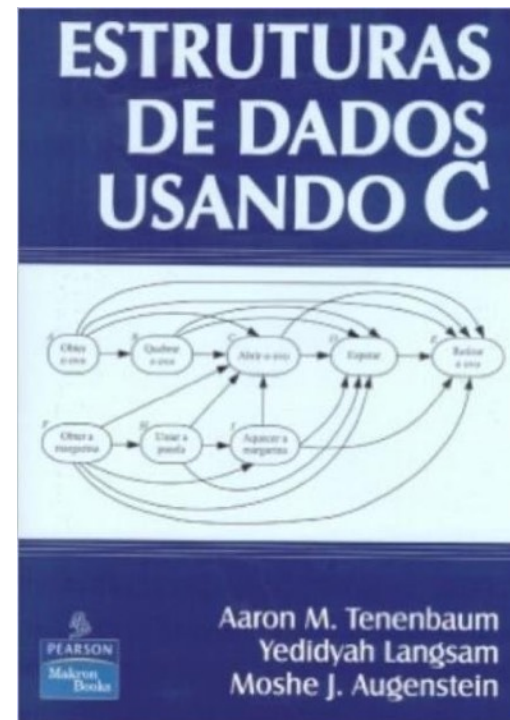
Roteiro

- 1 Introdução
- 2 Árvores Binárias
- 3 Propriedades e Definições
- 4 Inserção em Árvores Binárias
- 5 Pesquisa em Árvores Binárias
- 6 Referências

Referências sugeridas

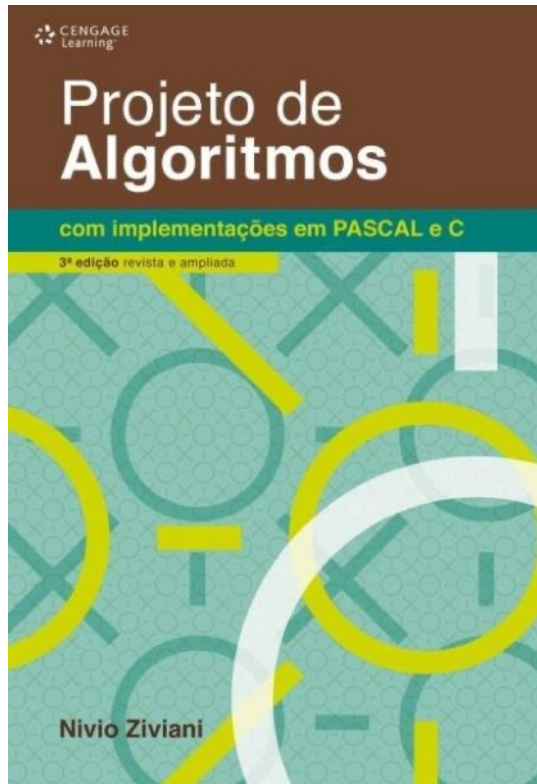


[Cormen et al, 2018]



[Tenenbaum et al, 1995]

Referências sugeridas



[Ziviani, 2010]



[Drozdek, 2017]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br