ED62A-COM2A
ESTRUTURAS DE DADOS

Aula 06 - Árvores Binárias

Prof. Rafael G. Mantovani

26/04/2019

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Apucarana - PR, Brasil
Abril, 2019

Universidade Tecnológica Federal do Paraná (UTFPR)
Engenharia de Computação

Roteiro

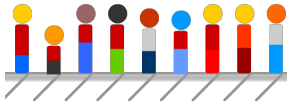
1. Introdução
2. Árvores Binárias
3. Propriedades e Definições
4. Inserção em Árvores Binárias
5. Pesquisa em Árvores Binárias
6. Referências

Roteiro

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

Introdução

Pilhas

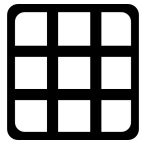


Filas

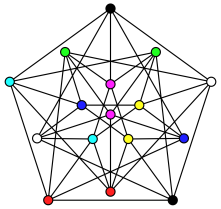
Árvores

Problema

Vetores



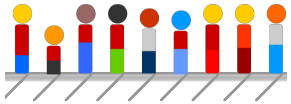
Grafos



Etc

Introdução

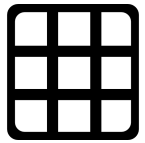
~~Pilhas~~



~~Filas~~

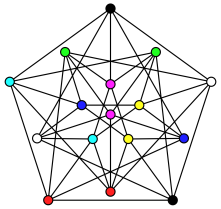
Árvores

~~Vetores~~



Problema

Grafos



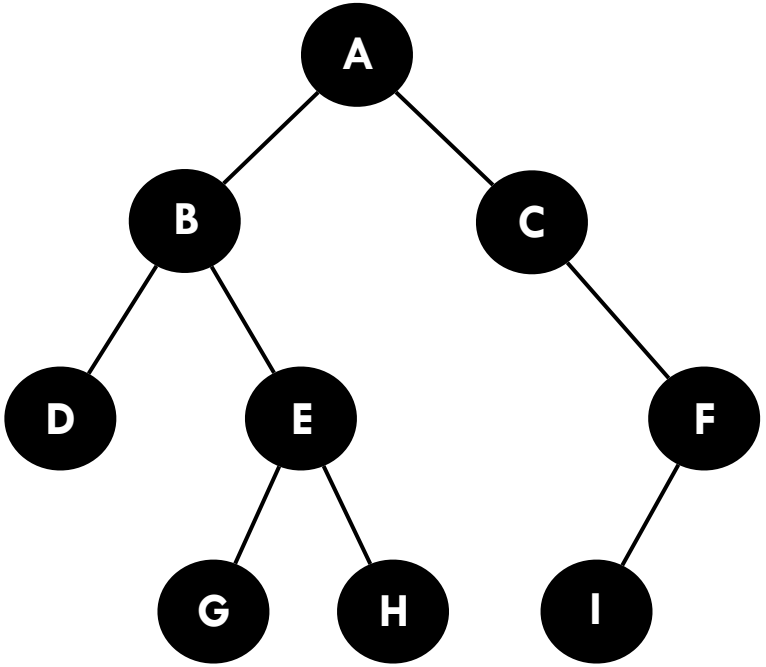
Etc

Introdução

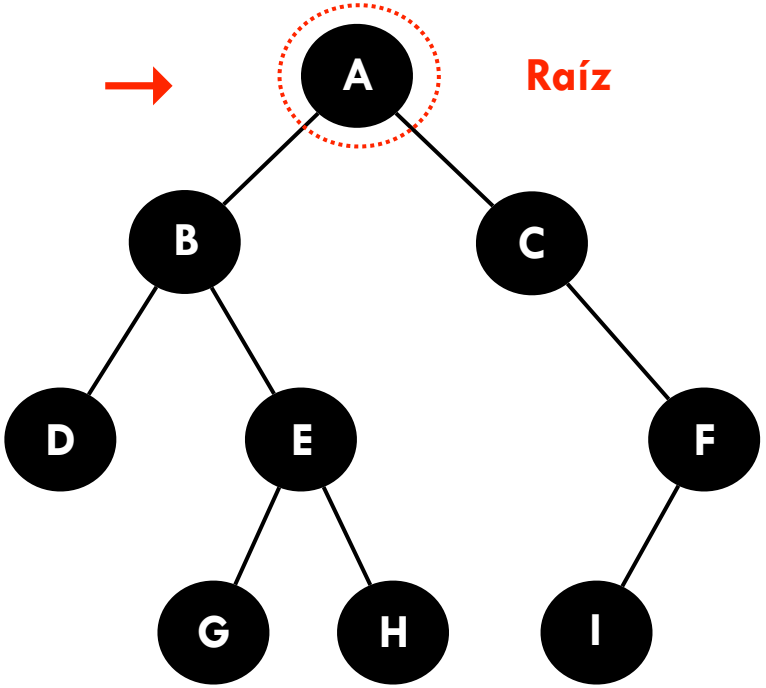
Introdução

Introdução

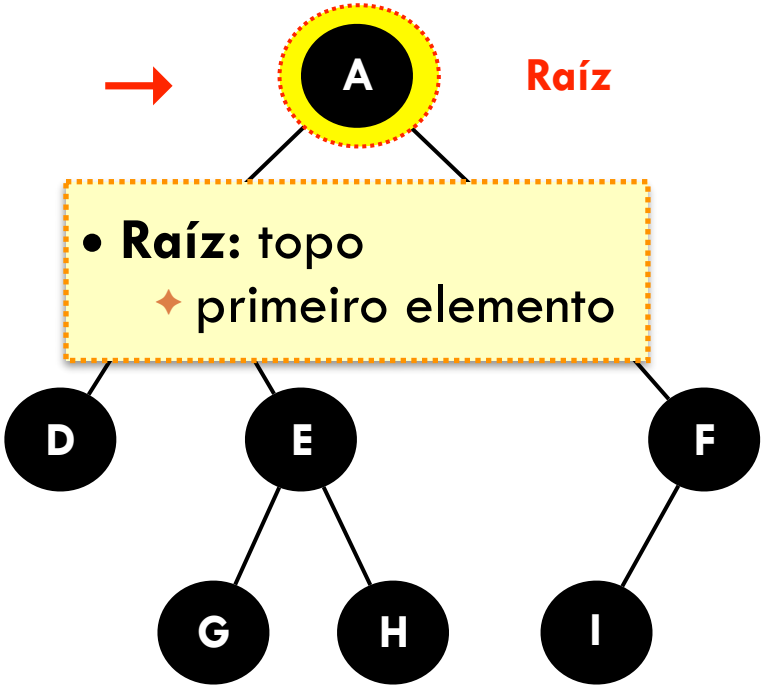
Introdução



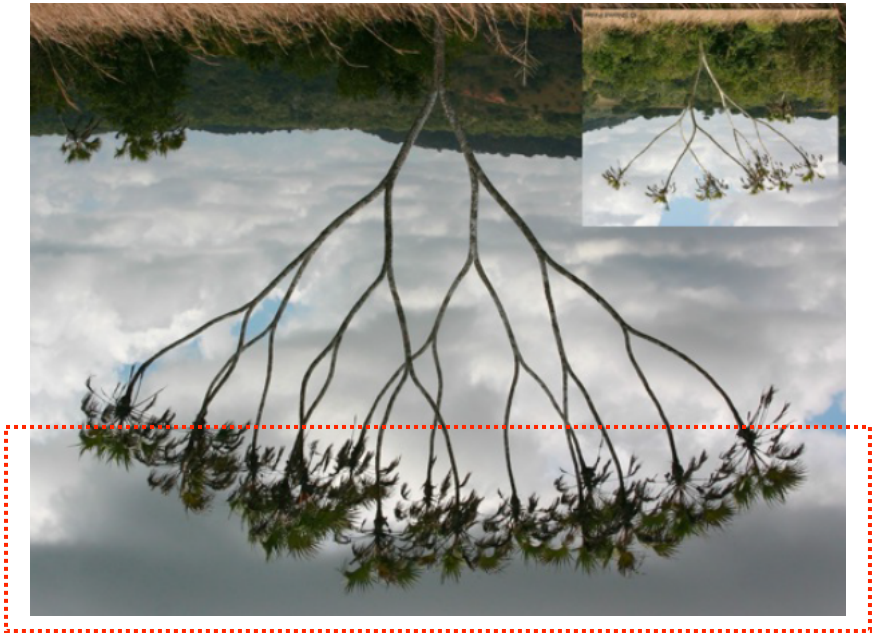
Introdução

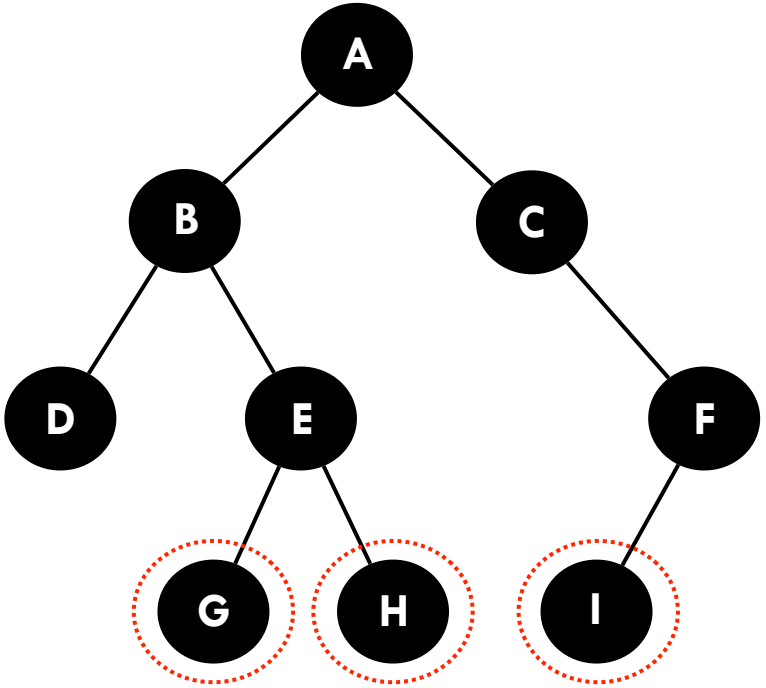


Introdução



Introdução

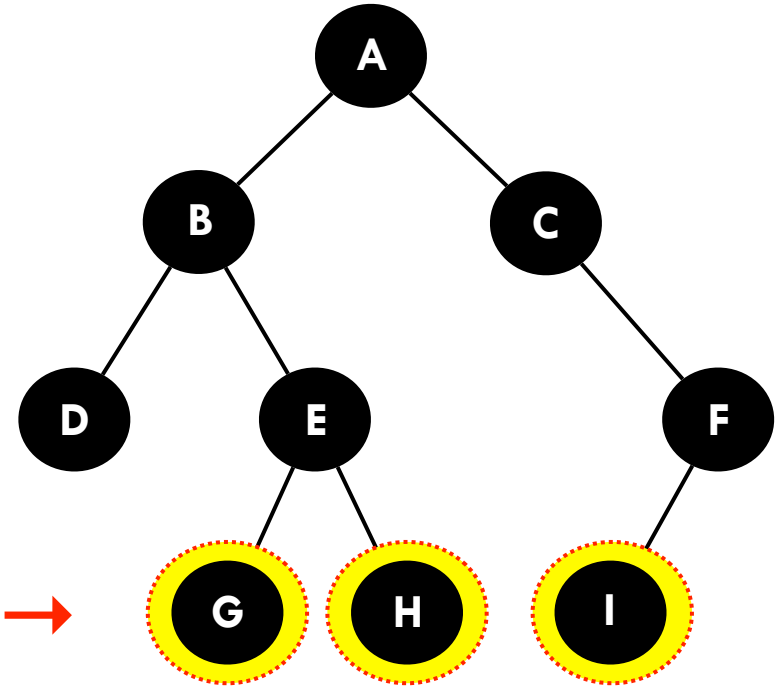




Folhas

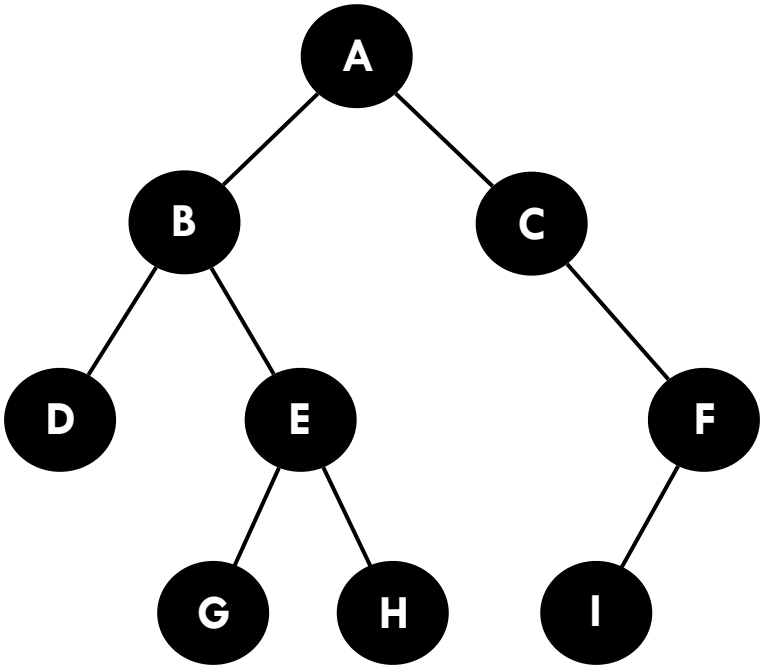
Introdução

- **Folhas:** base
  - nós terminais
  - filhos vazios (NULL)

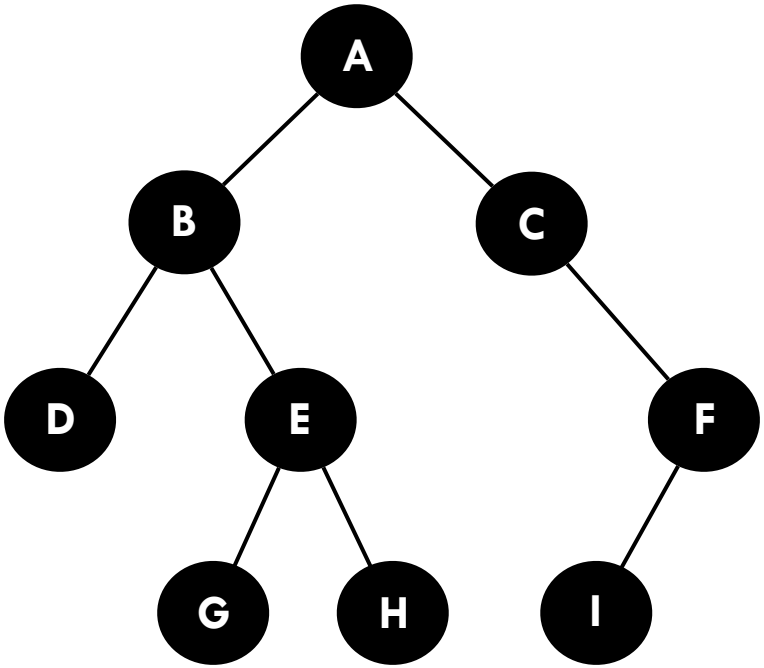


Folhas

Árvores

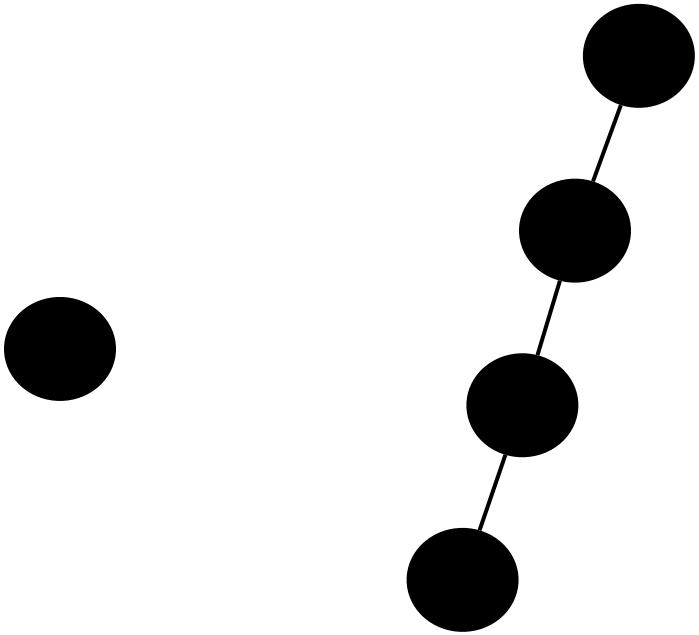


Árvores



- **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

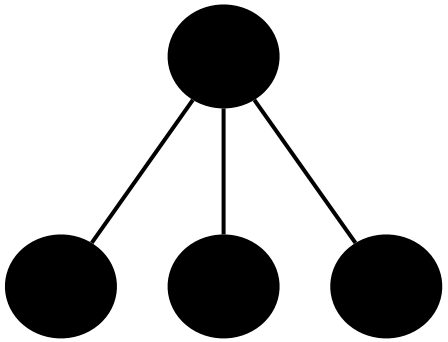


(a)
árvore vazia

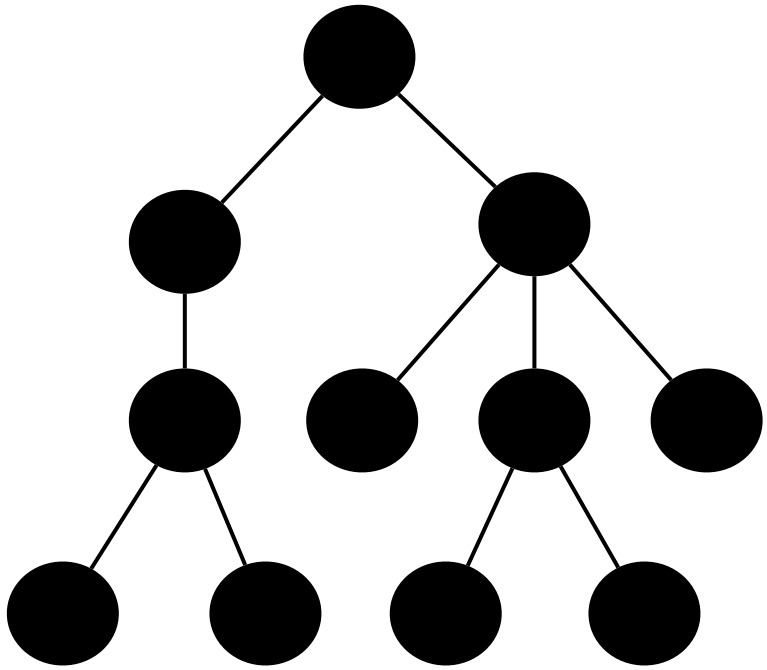
(b)

(c)

Exemplos de árvores



(d)

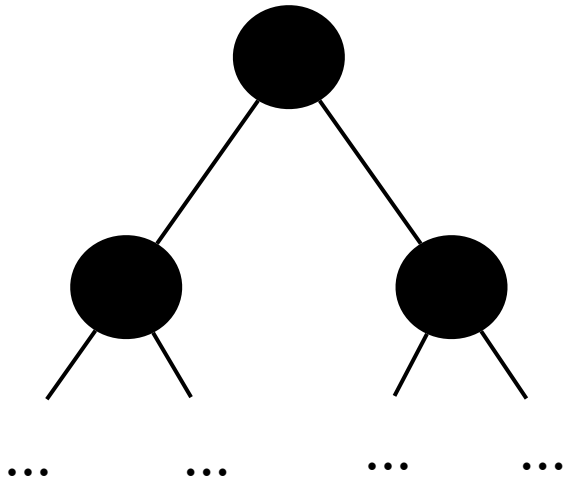


(e)

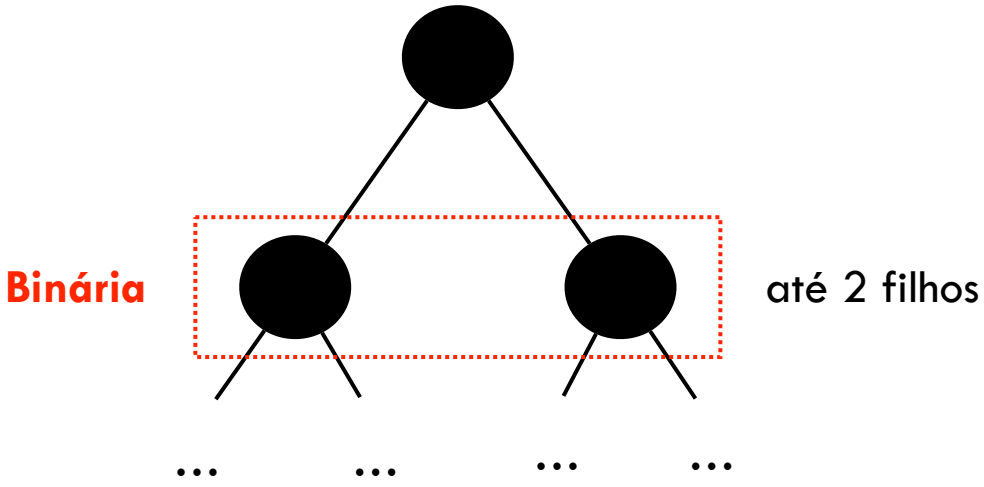
Roteiro

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

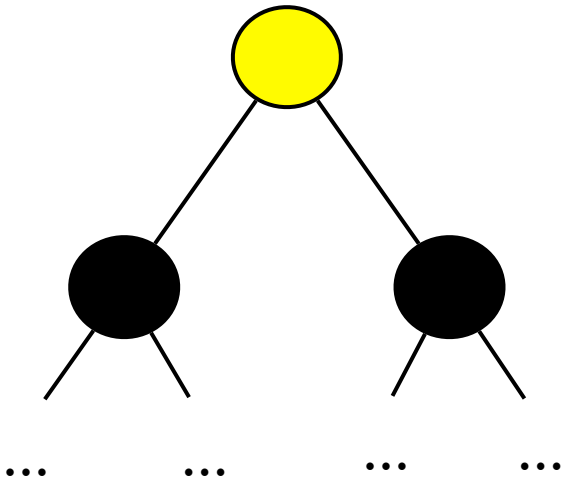
Árvore Binária



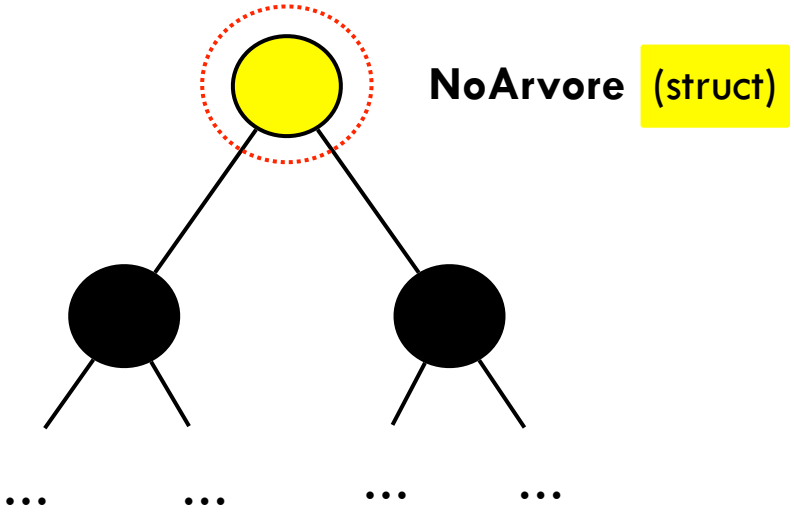
Árvore Binária



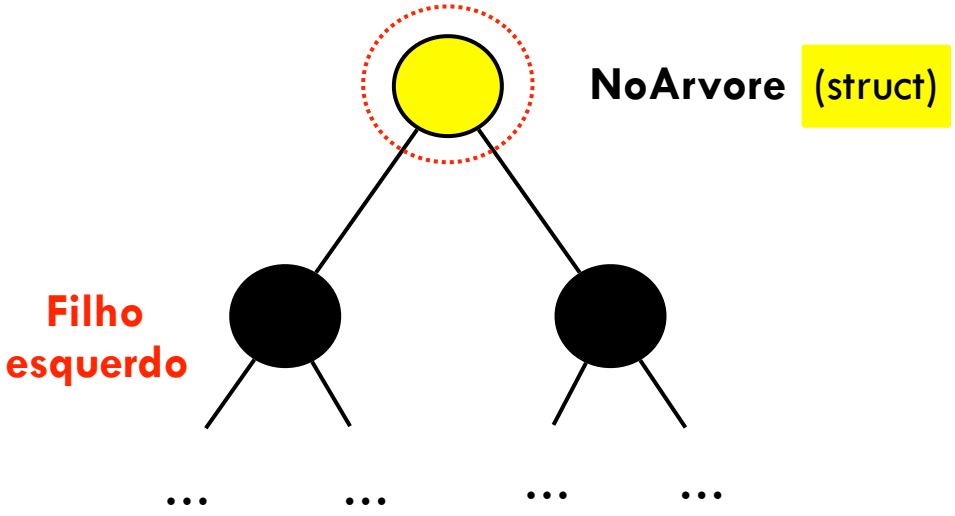
Árvore Binária



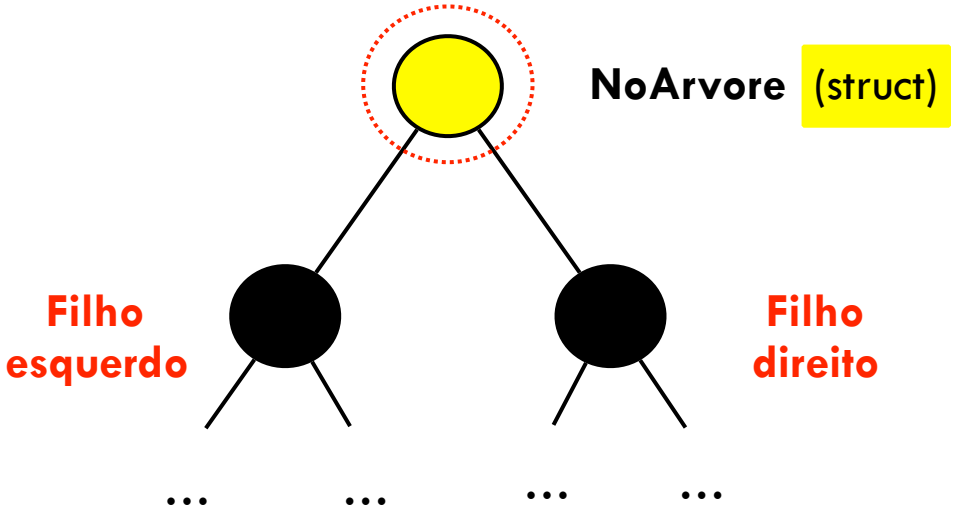
Árvore Binária



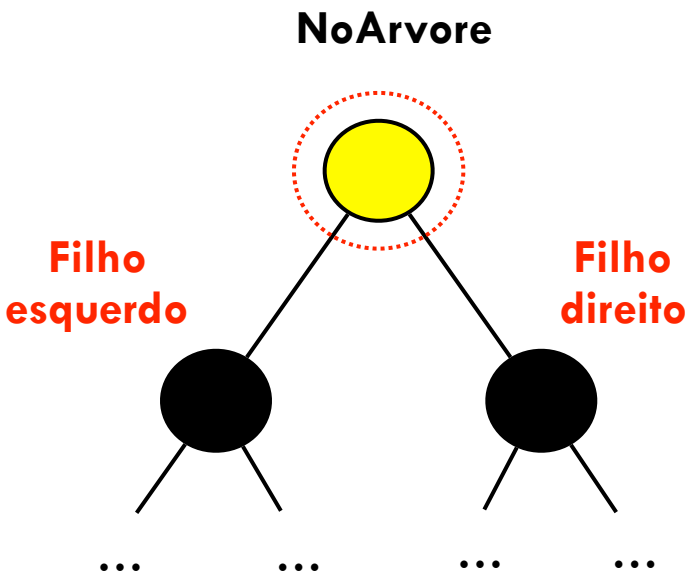
Árvore Binária



Árvore Binária

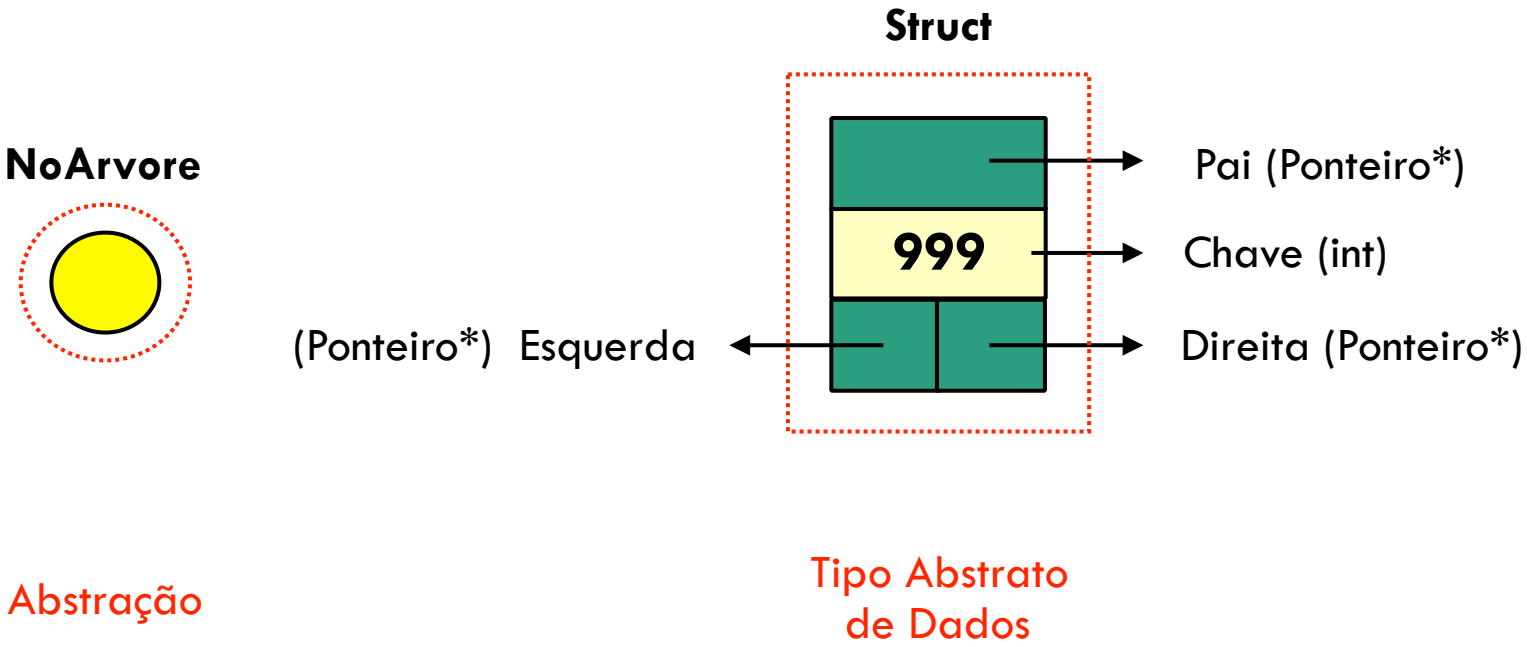


Árvore Binária

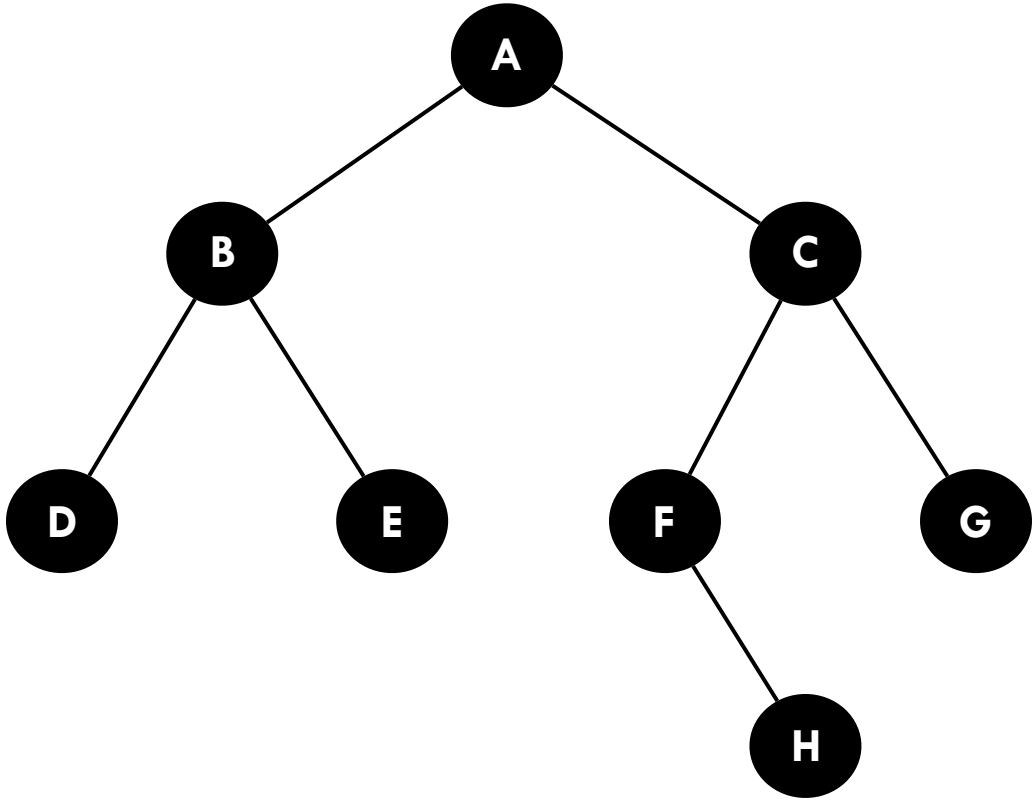



```
NoArvore
1. inteiro chave
2. /* … */
3. NoArvore* filho à direita
4. NoArvore* filho à esquerda
5. NoArvore* pai [opcional]
```

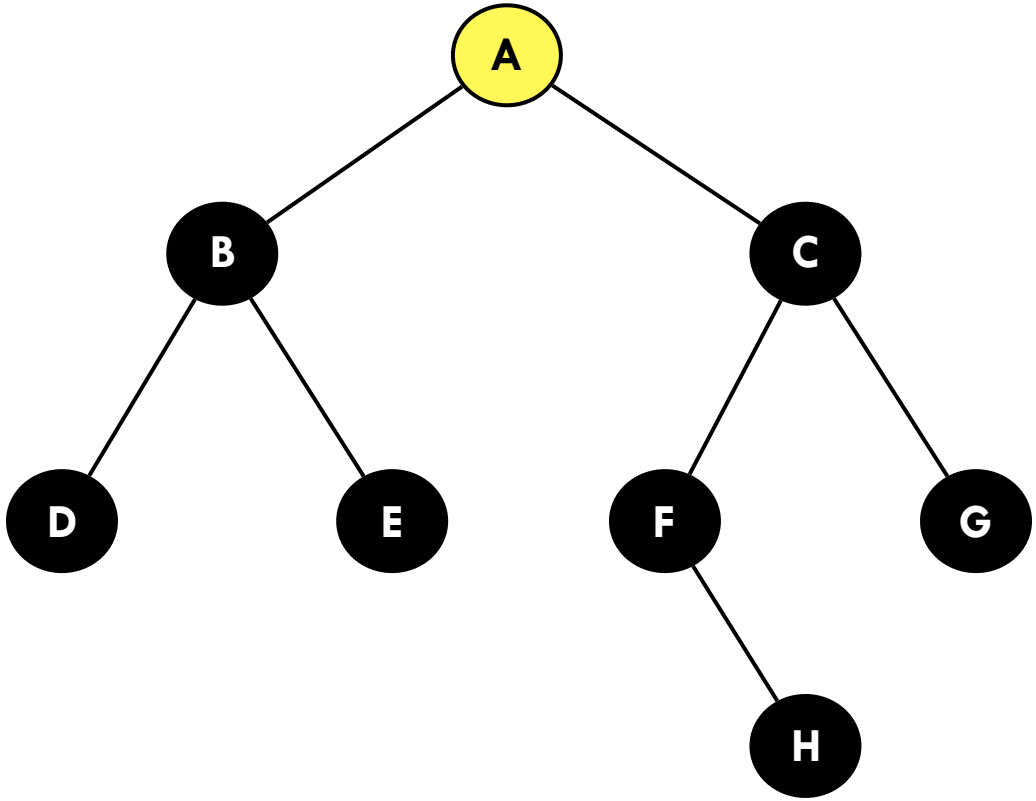
Árvore Binária



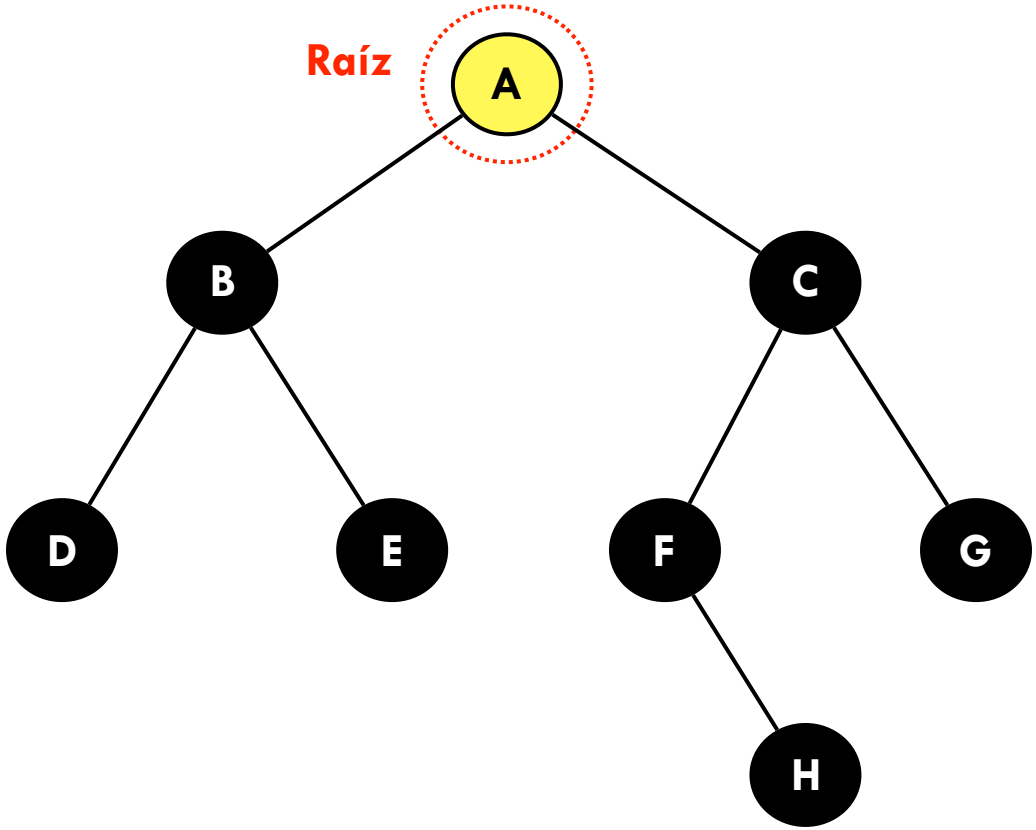
Árvore Binária



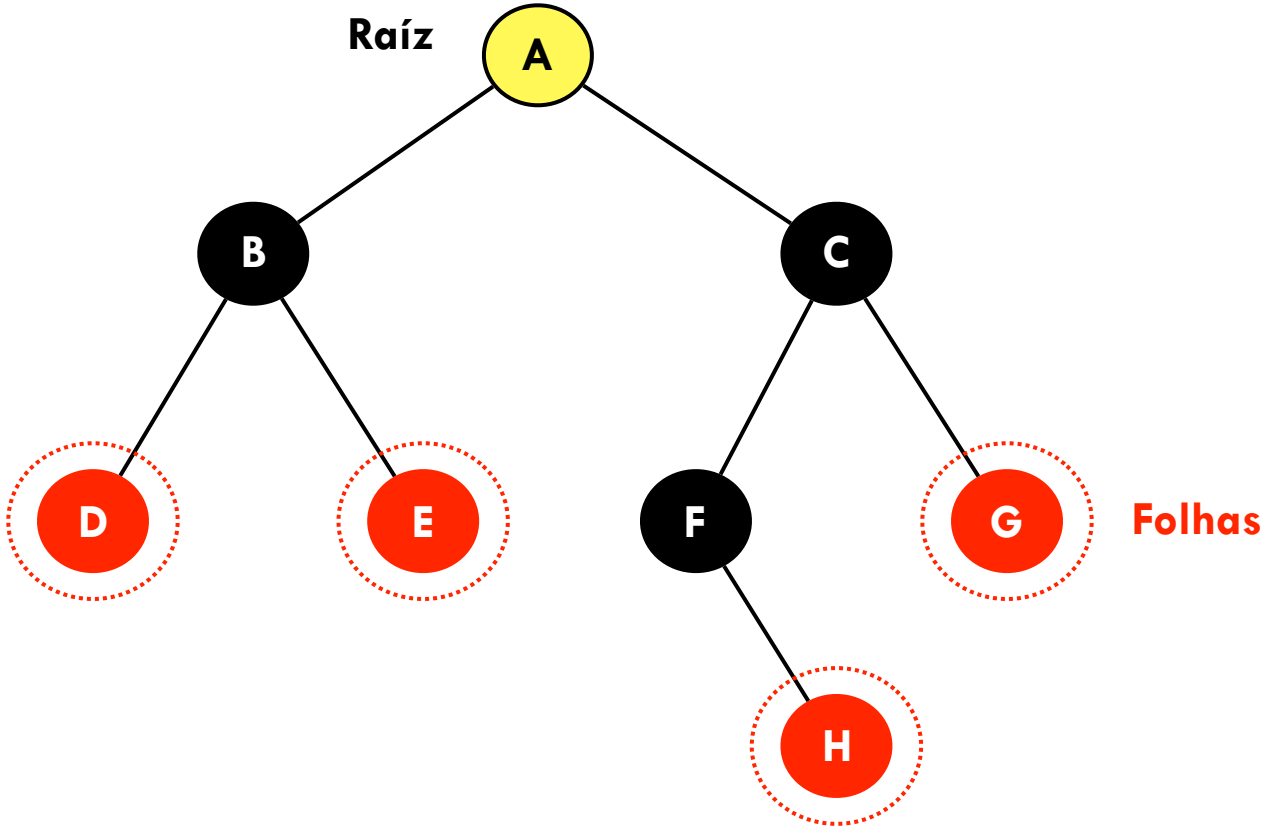
Árvore Binária



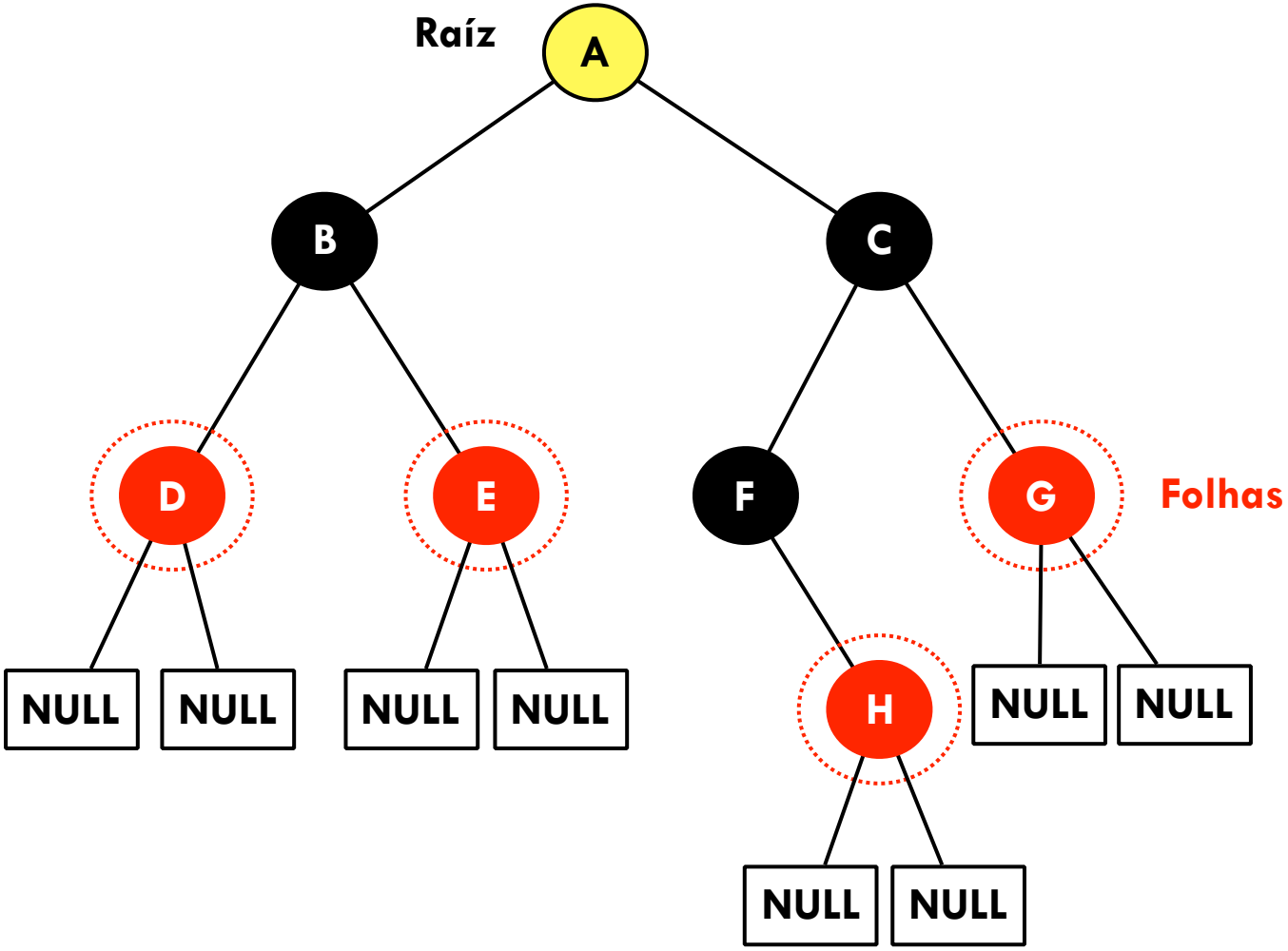
Árvore Binária



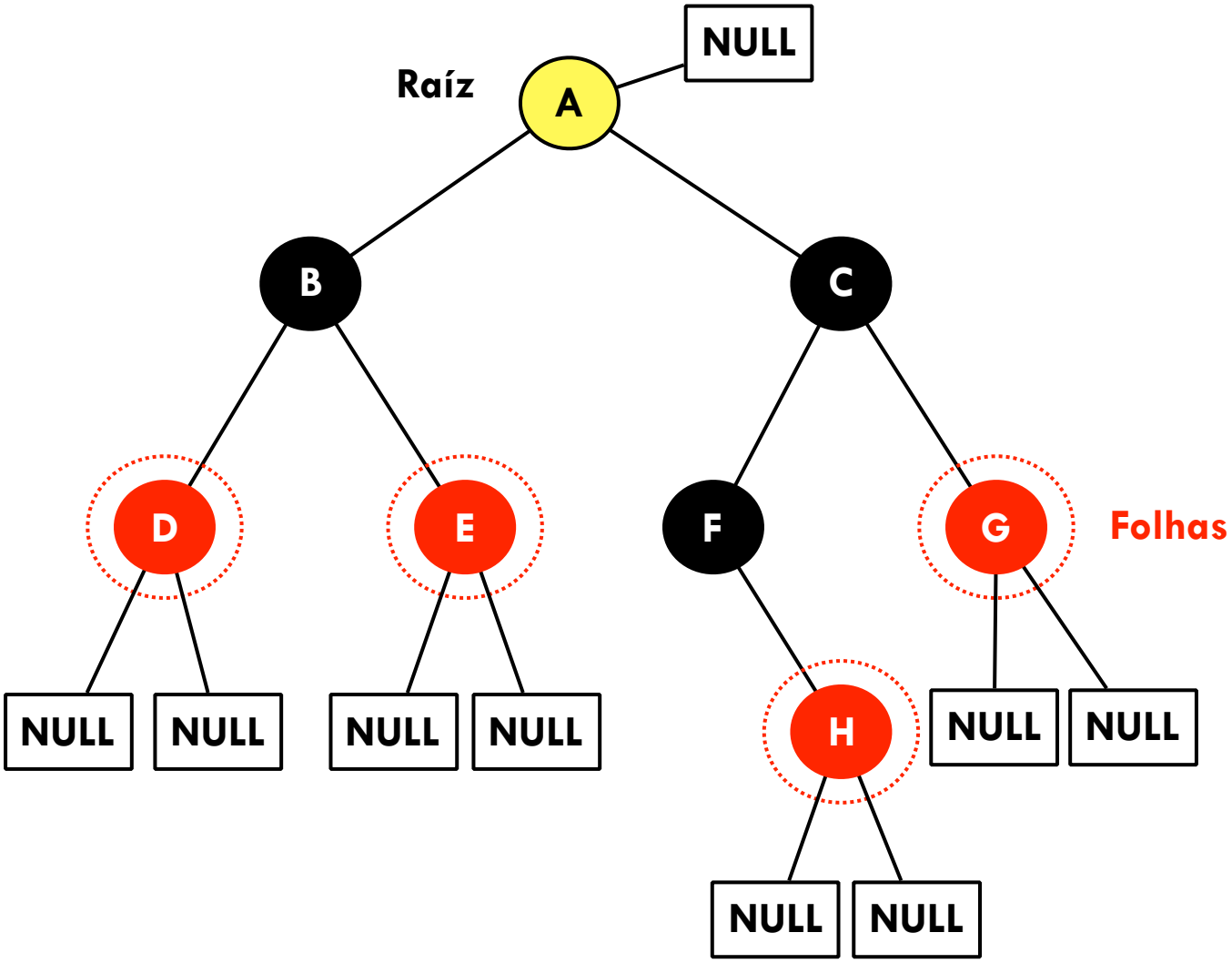
Árvore Binária



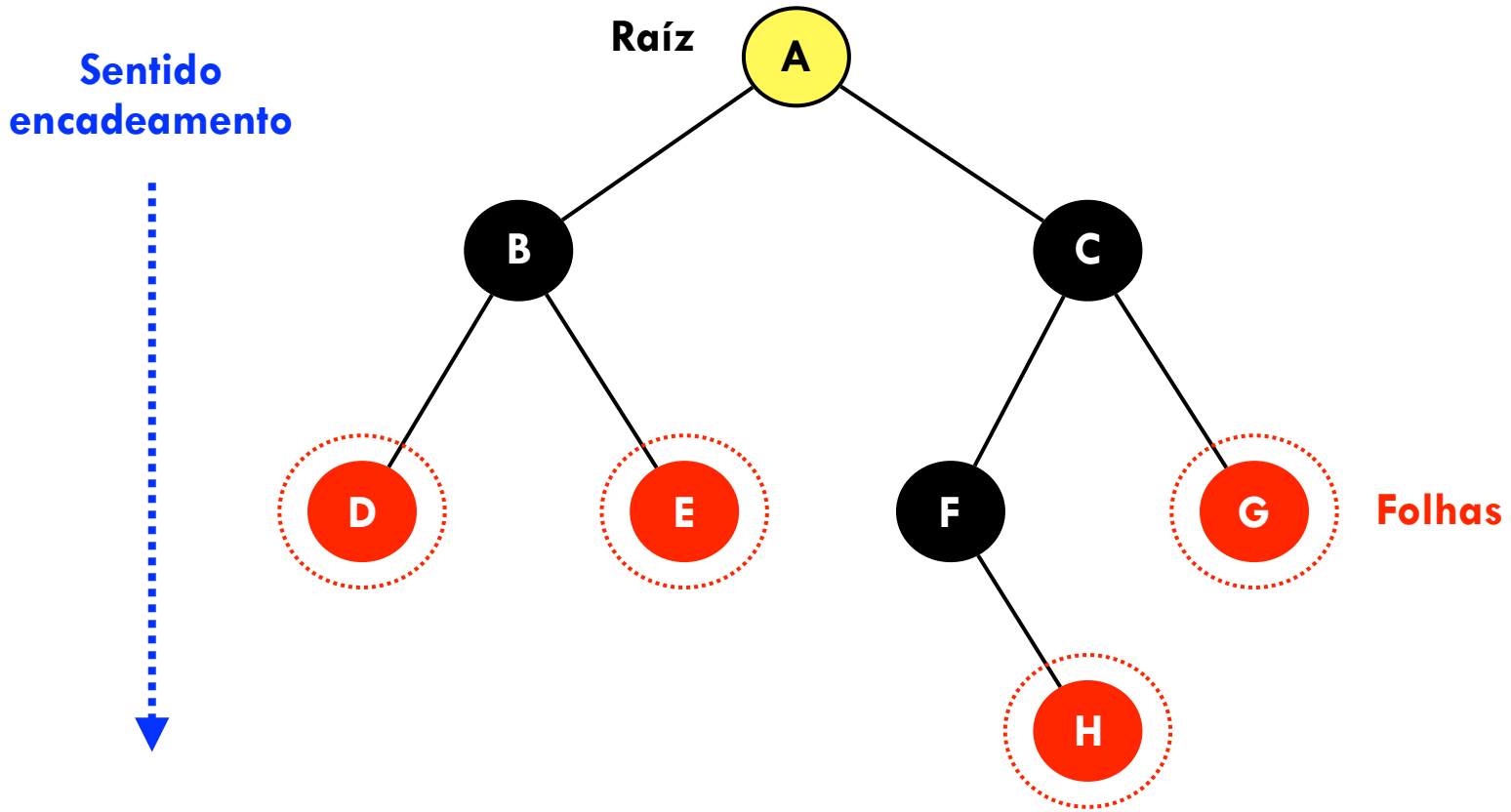
Árvore Binária



Árvore Binária



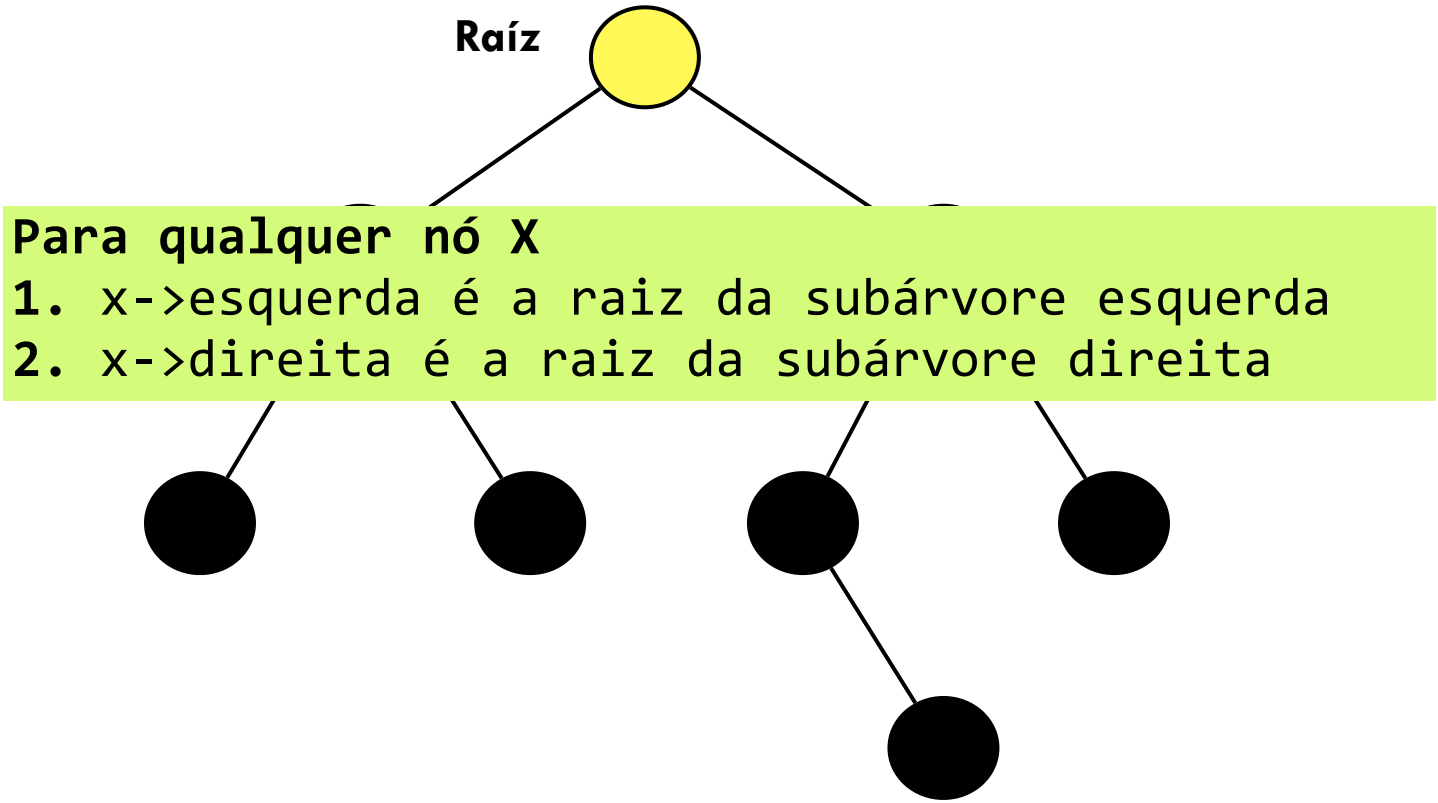
Árvore Binária



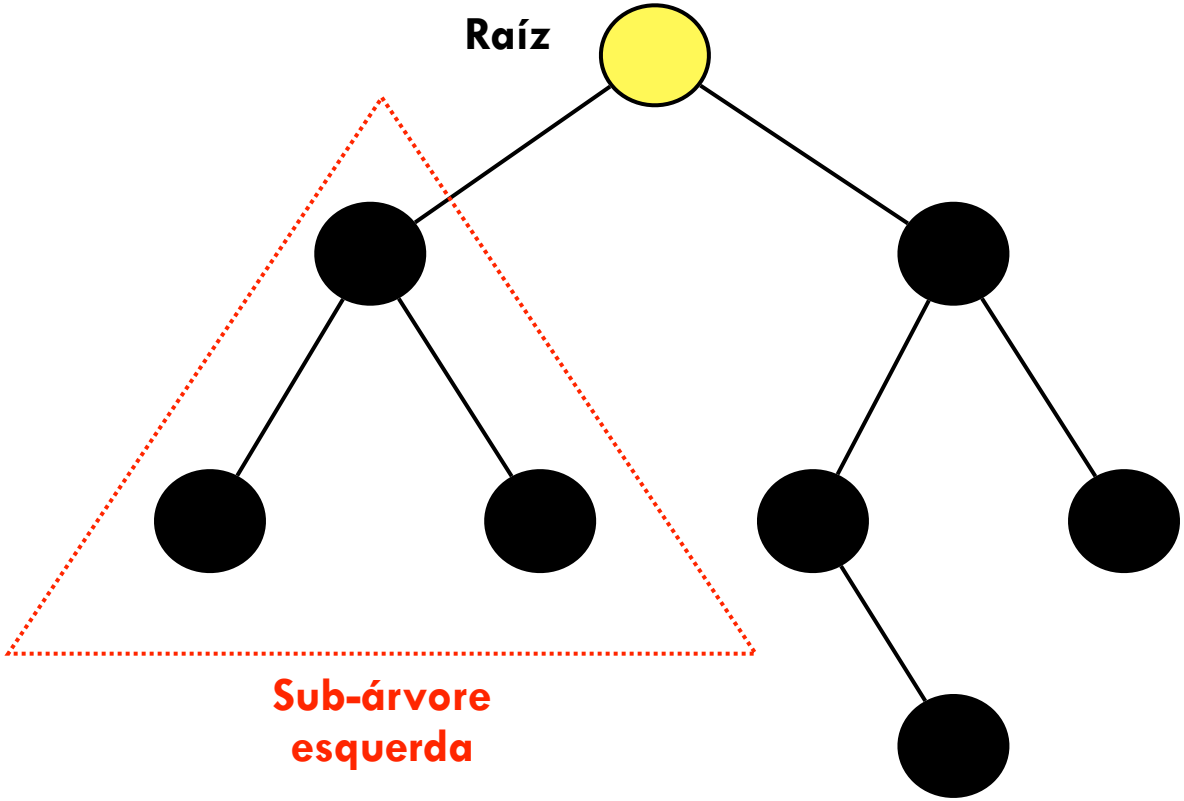
Roteiro

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

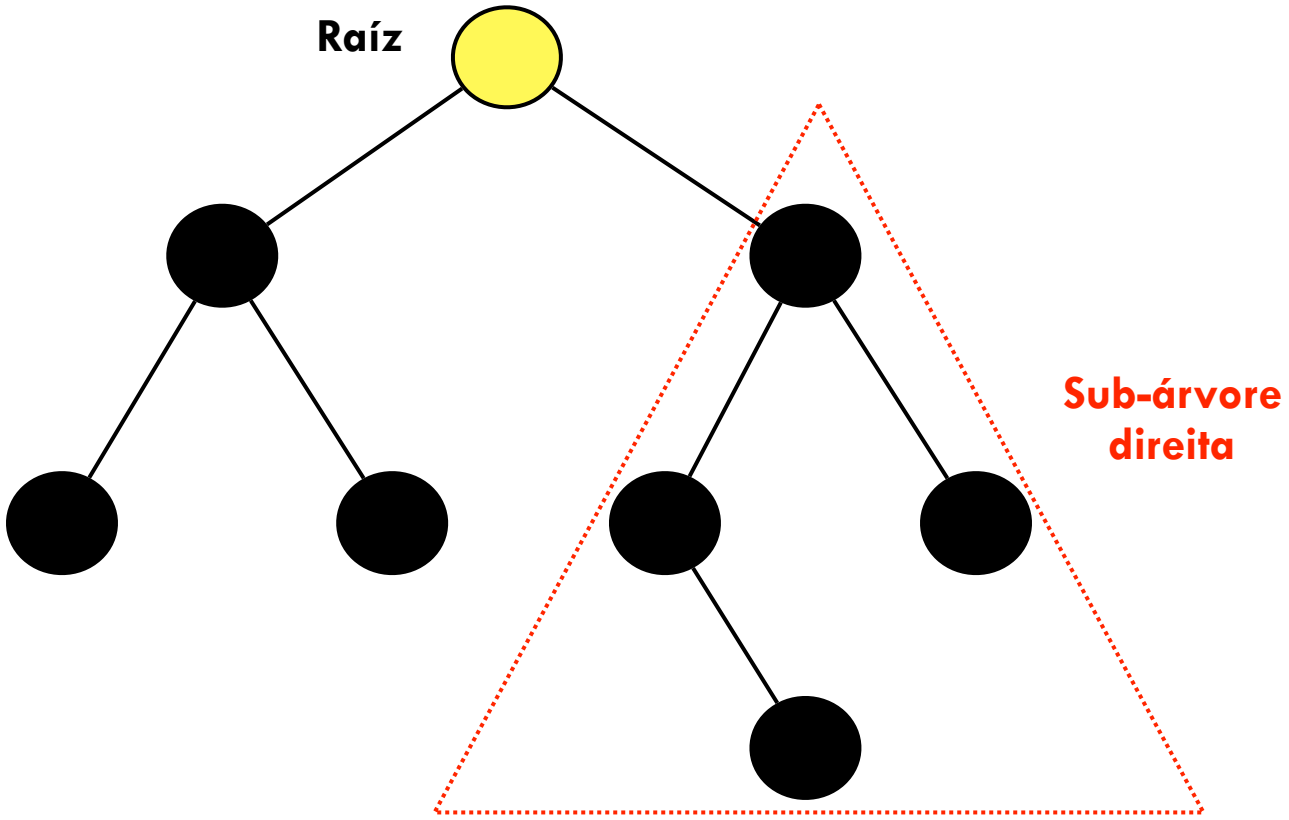
Sub-árvores


Raíz
Para qualquer nó X
1. x->esquerda é a raiz da subárvore esquerda
2. x->direita é a raiz da subárvore direita

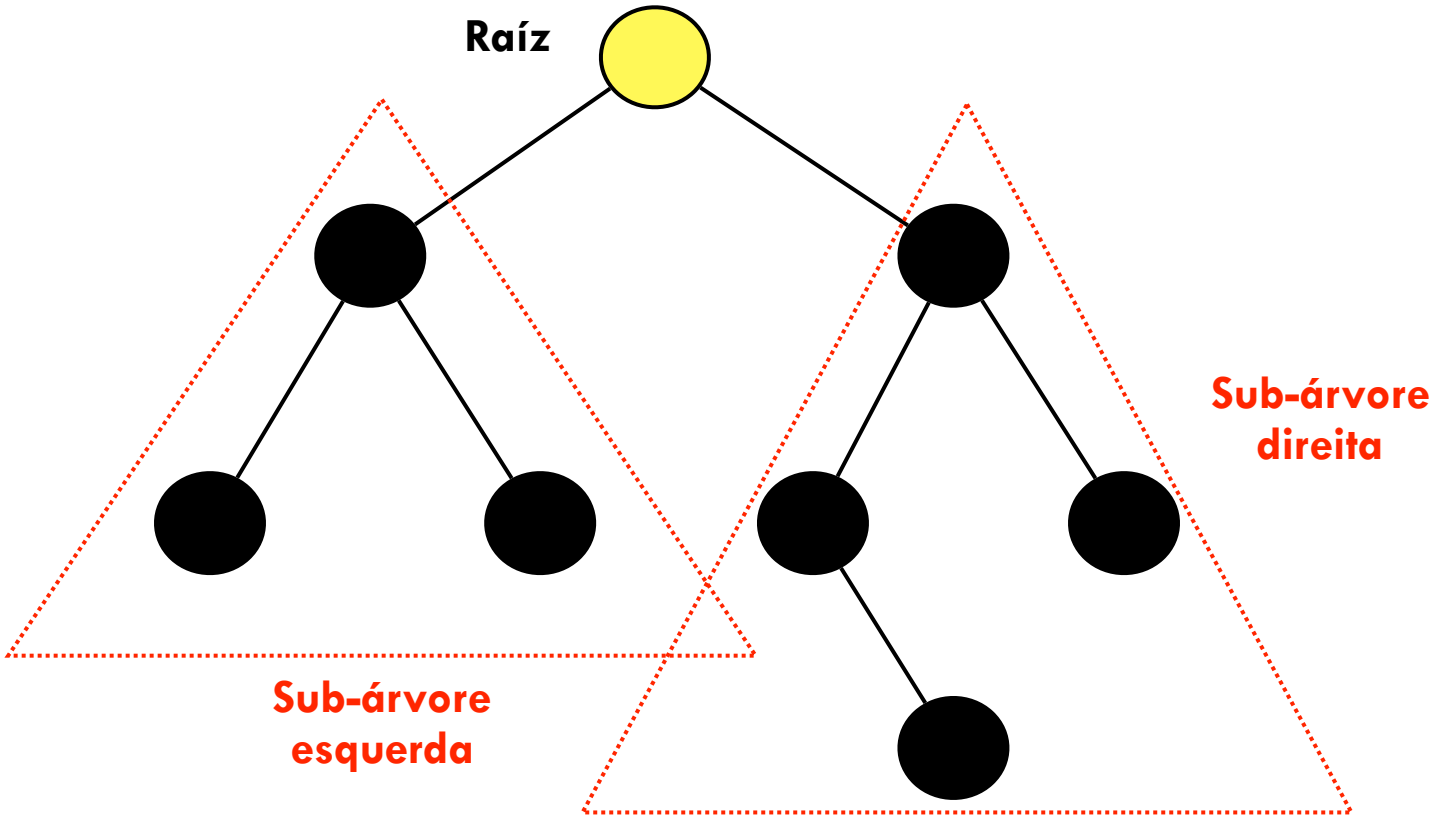
Sub-árvores



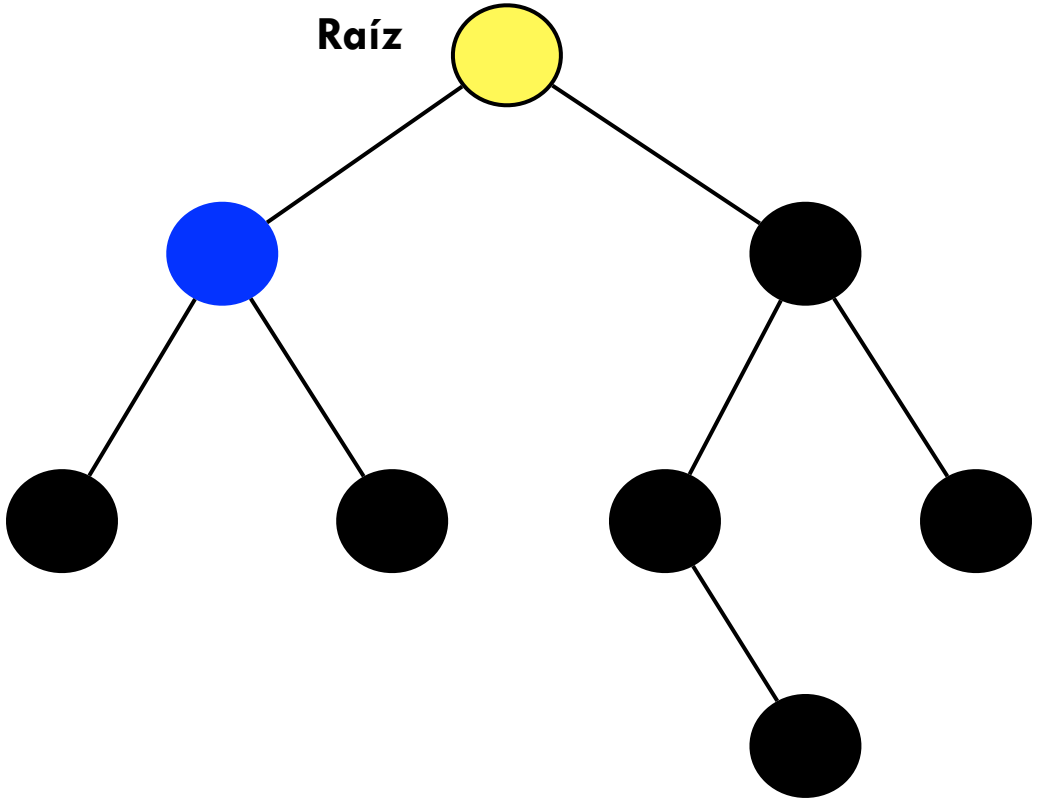
Sub-árvores



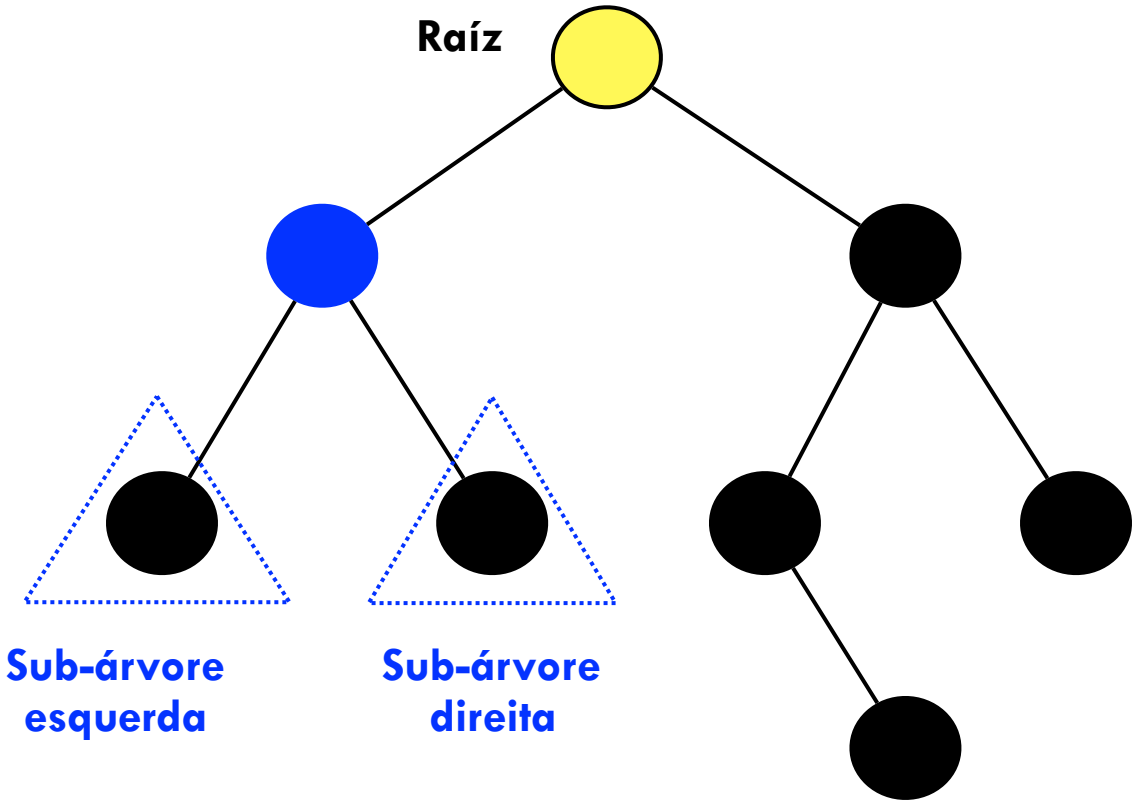
Sub-árvores



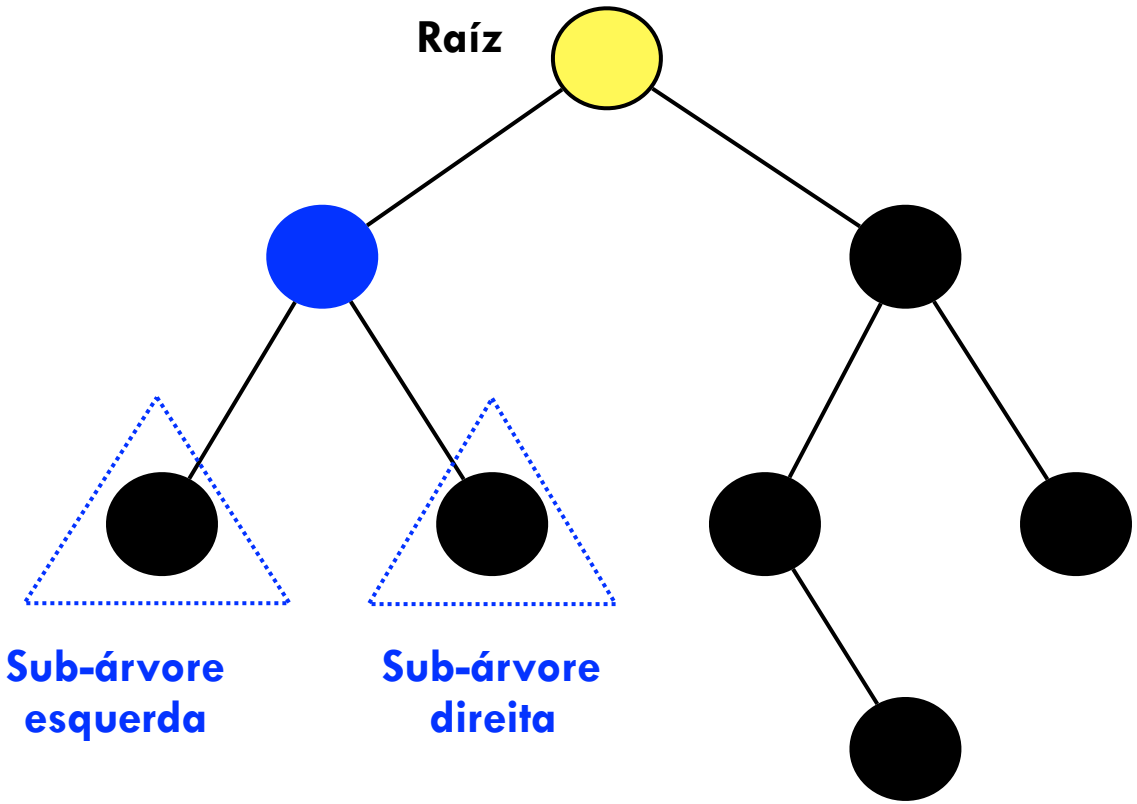
Sub-árvores



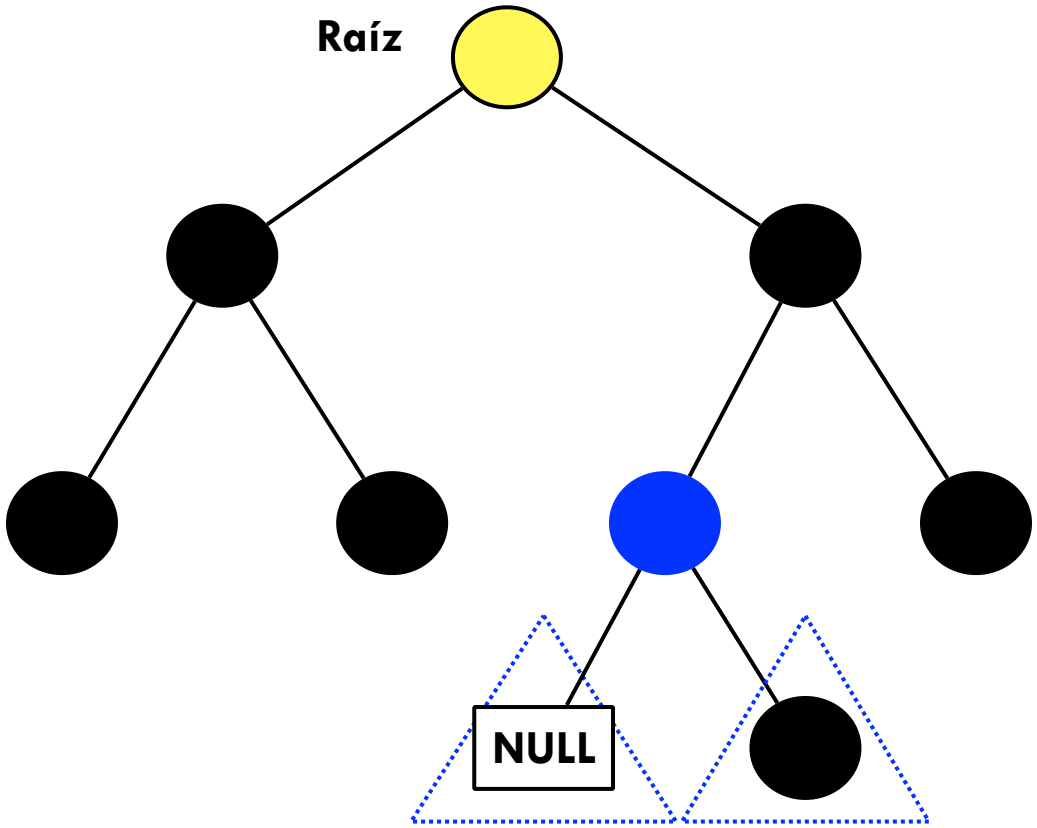
Sub-árvores



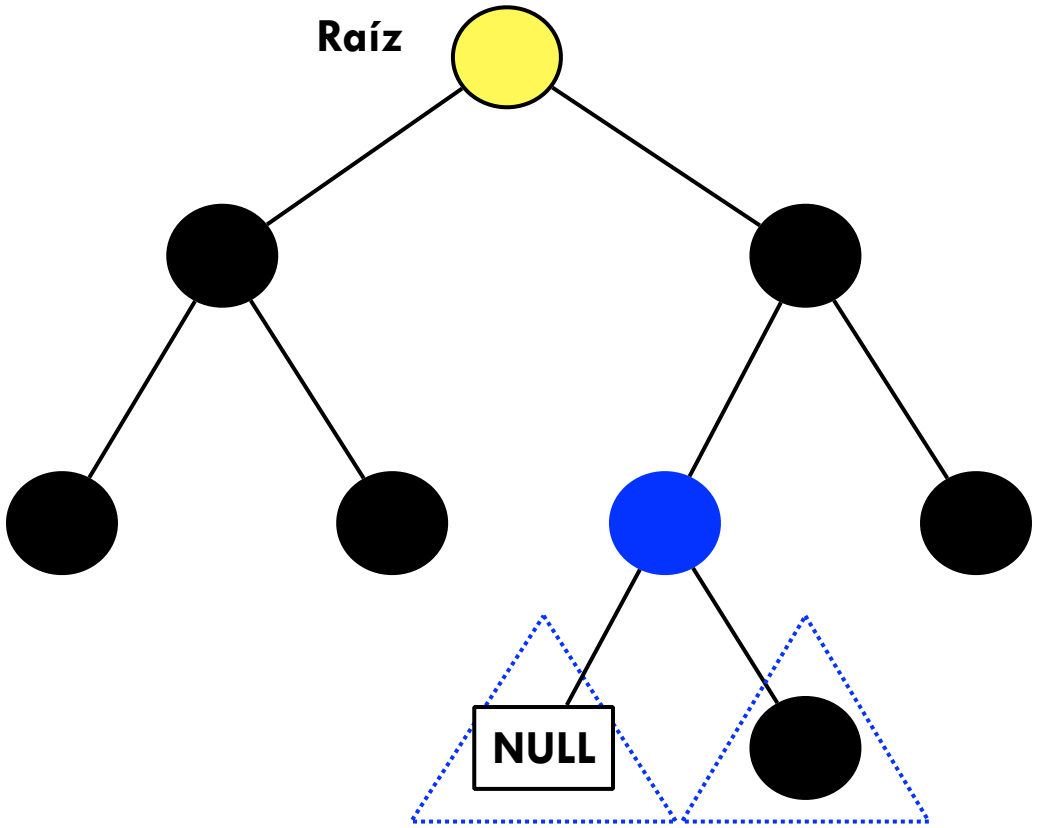
Sub-árvores



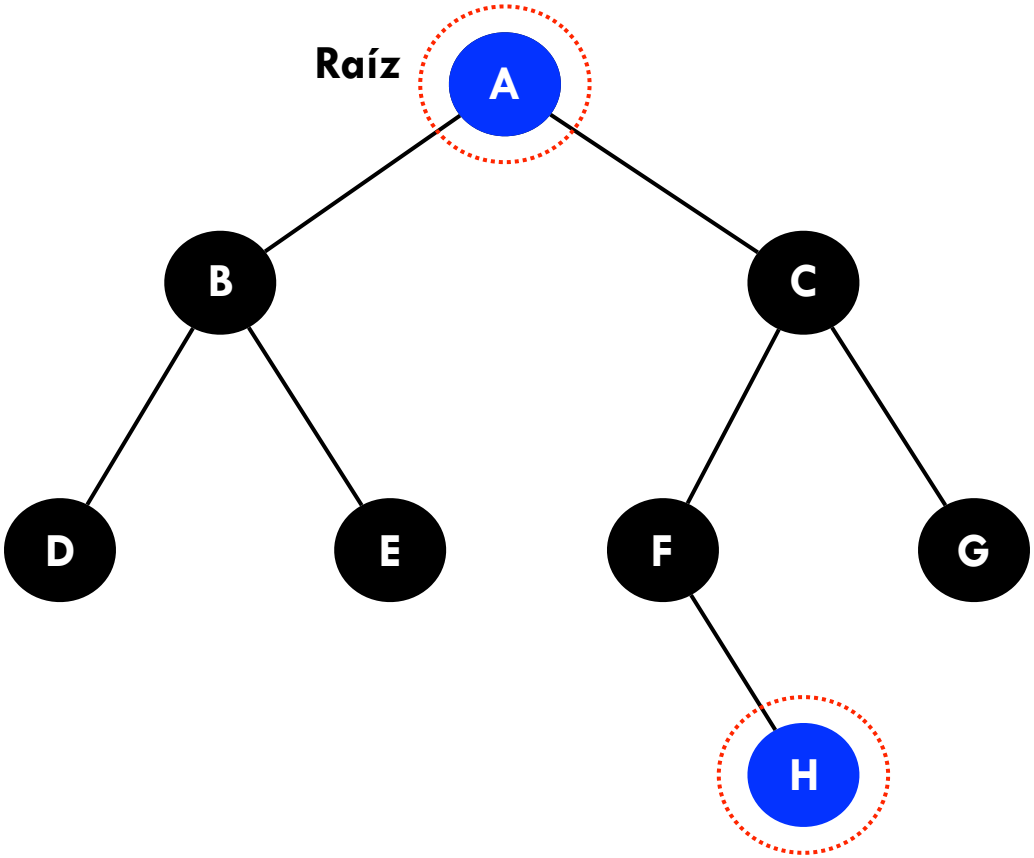
Sub-árvores



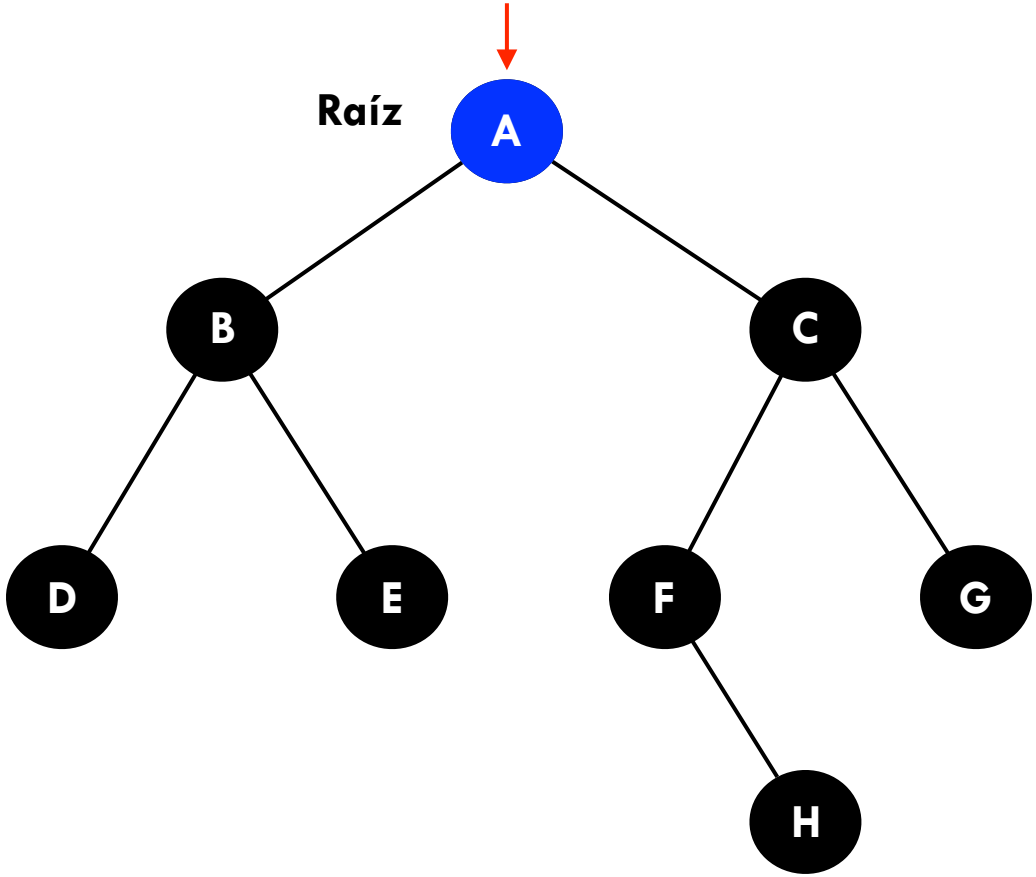
Sub-árvores



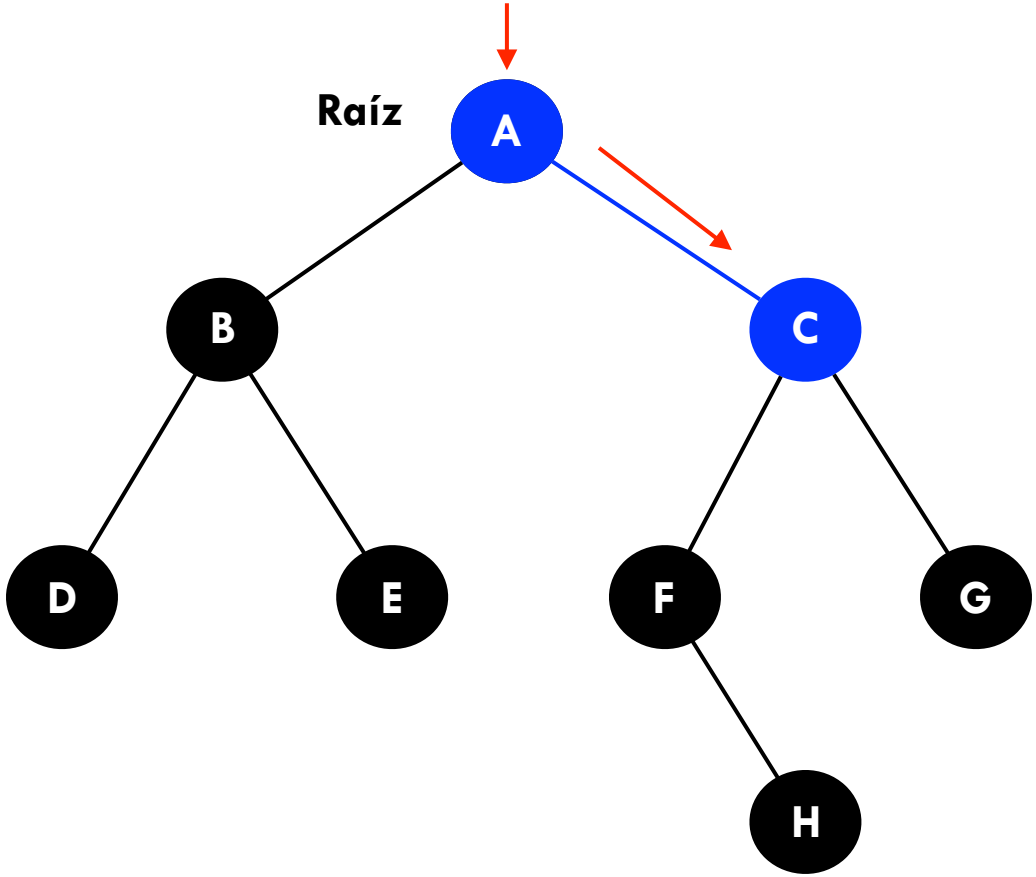
Caminho



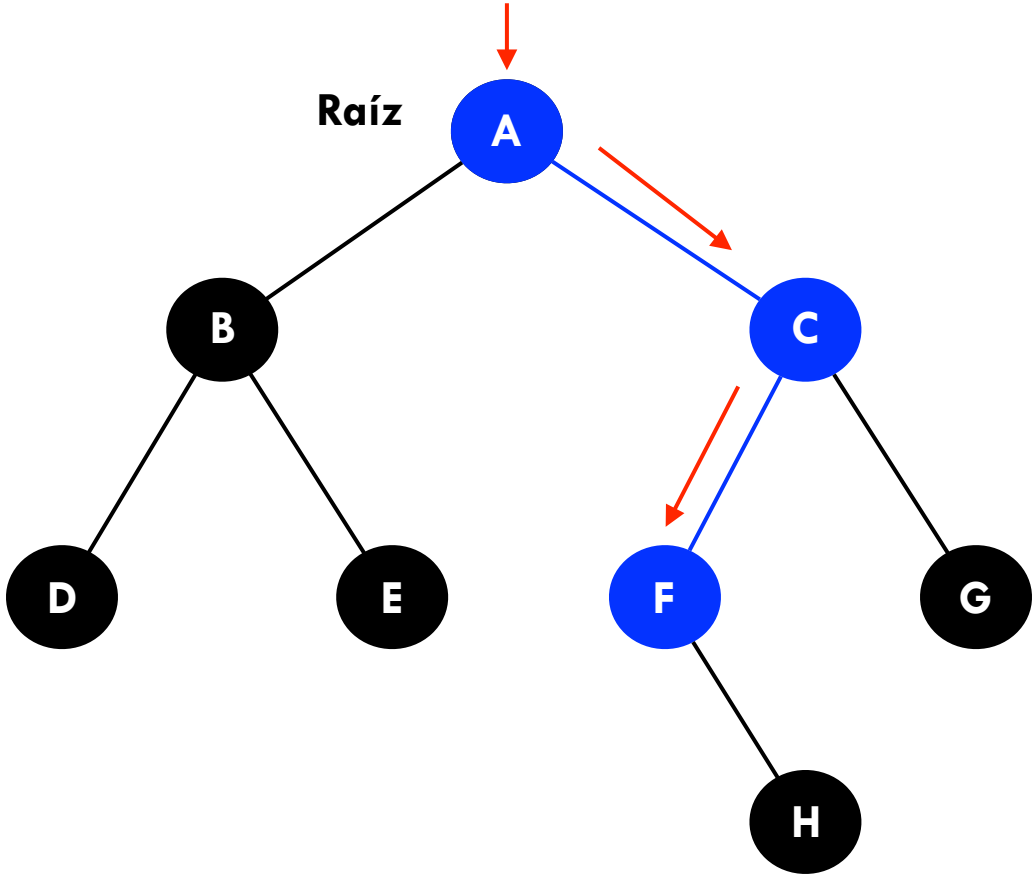
Caminho



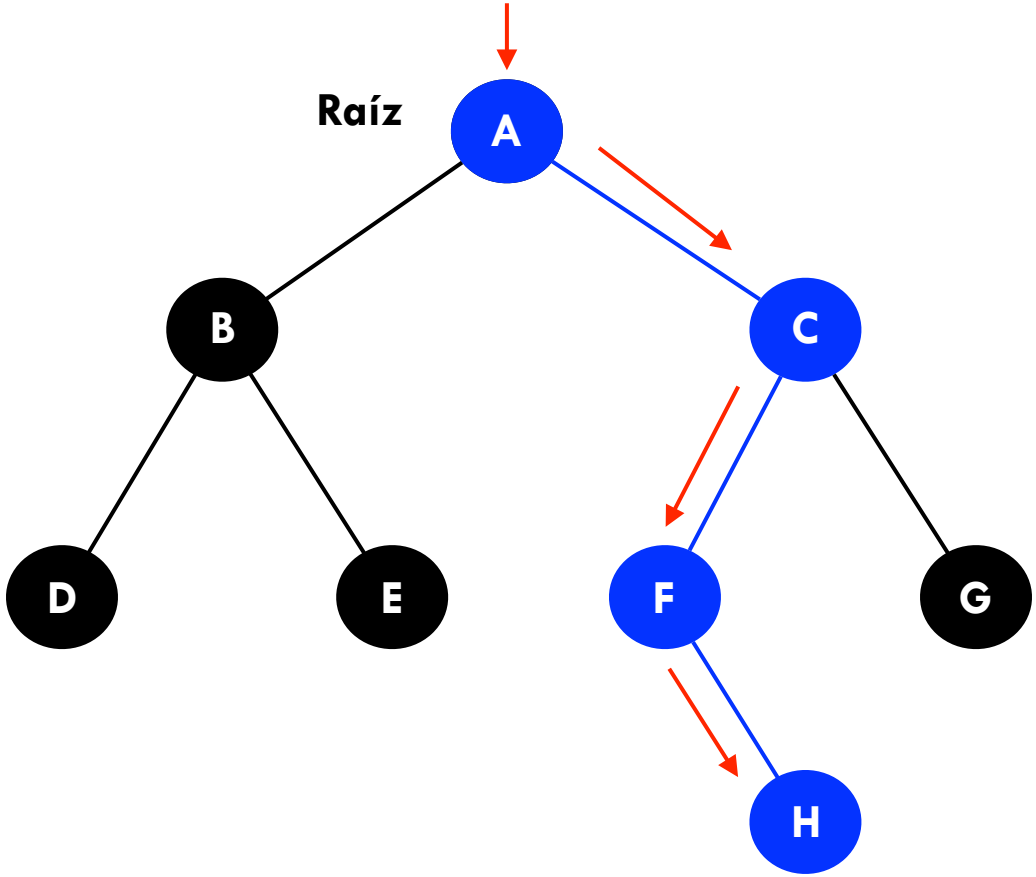
Caminho



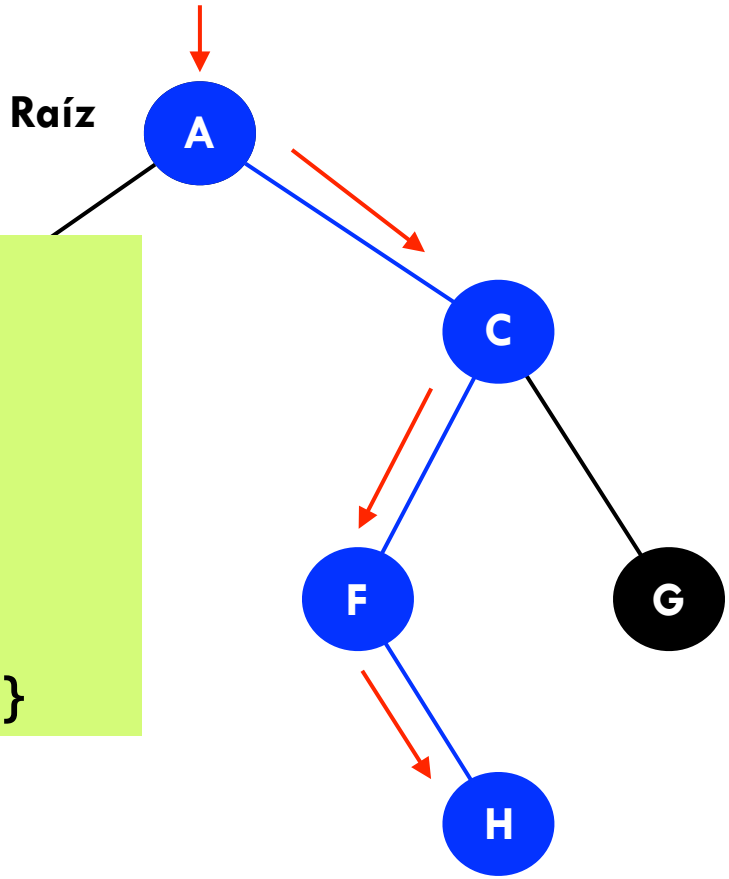
Caminho



Caminho



Caminho

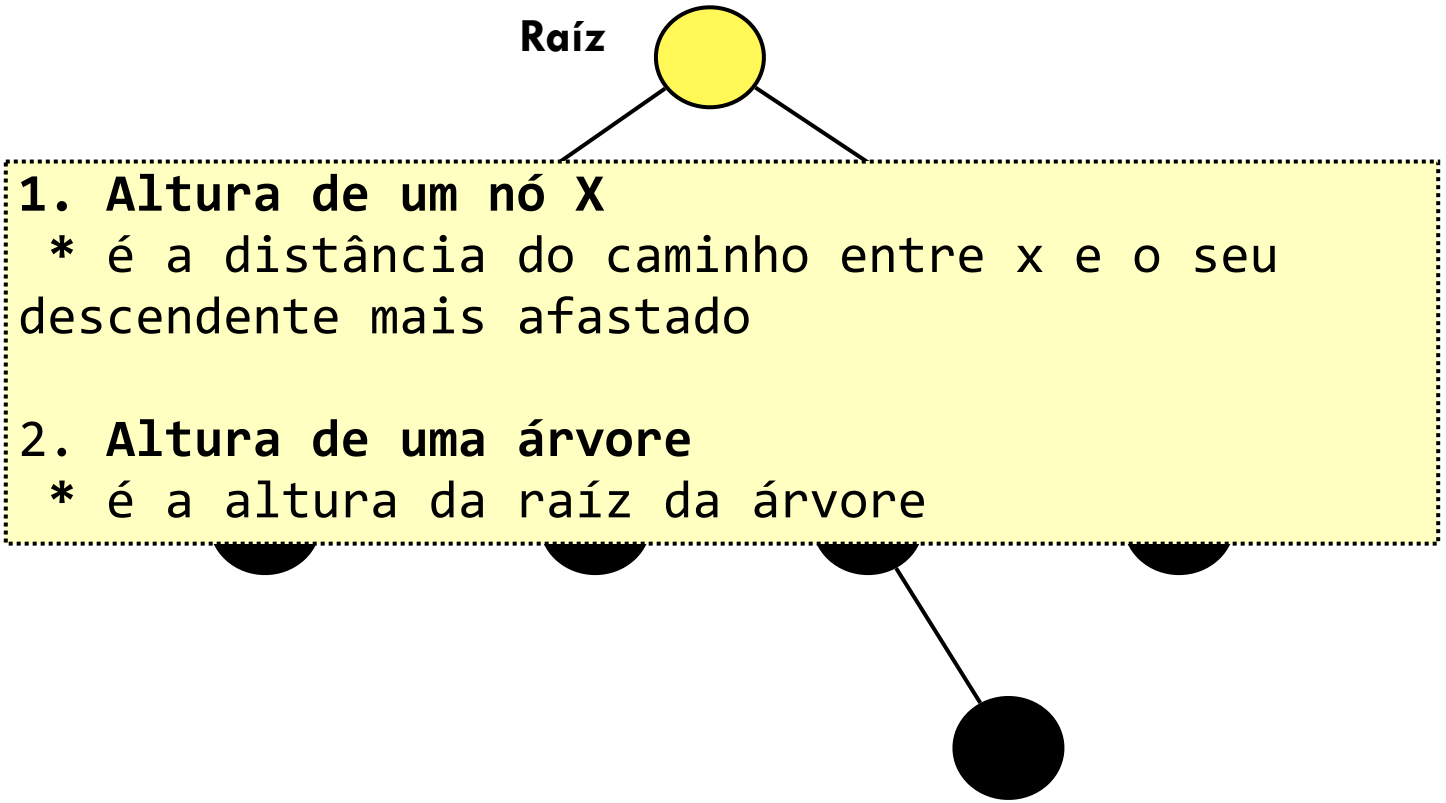


Caminho

1. Raiz (A)
2. Direita (C)
3. Esquerda (F)
4. Direita (H)

C = {A, C, F, H}

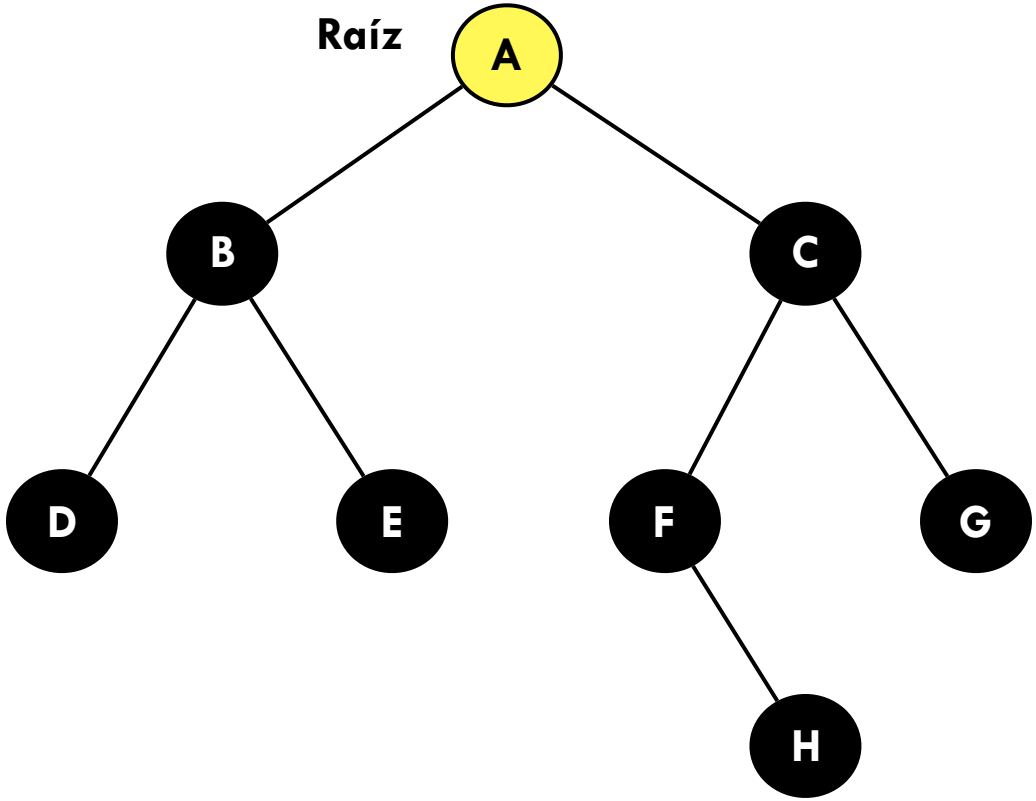
Altura



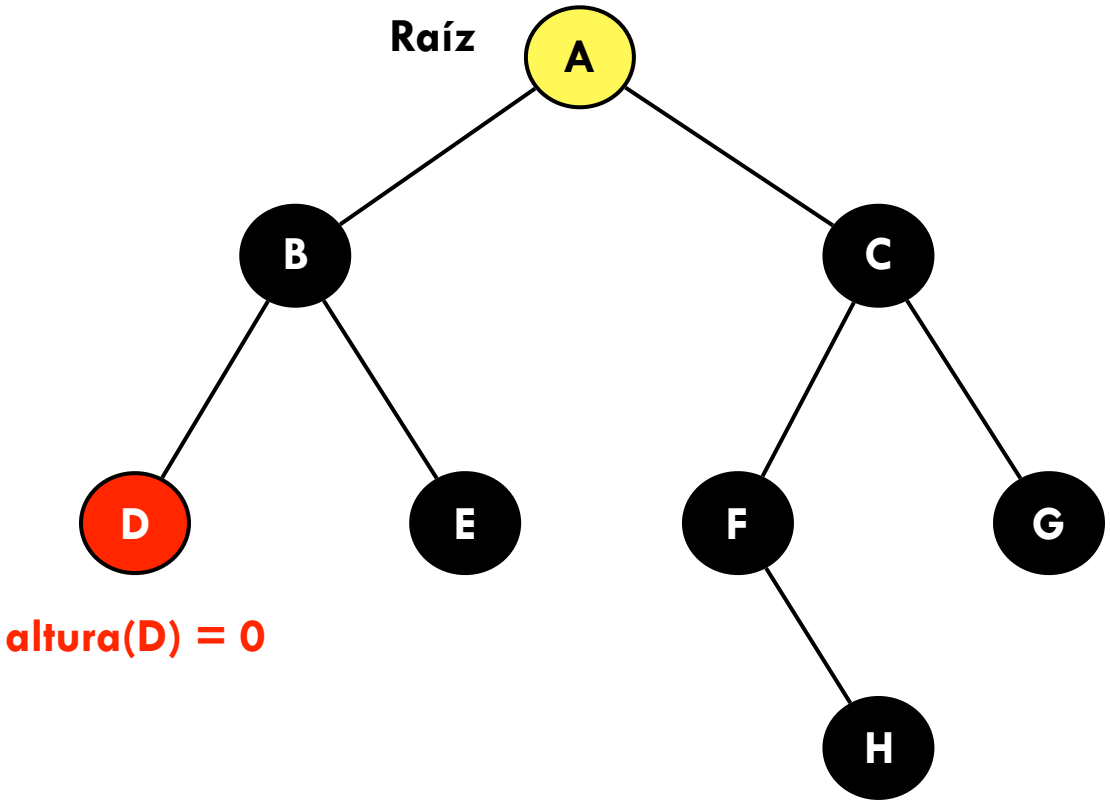
1. Altura de um nó X
 * é a distância do caminho entre x e o seu descendente mais afastado

2. **Altura de uma árvore**
 * é a altura da raíz da árvore

Altura

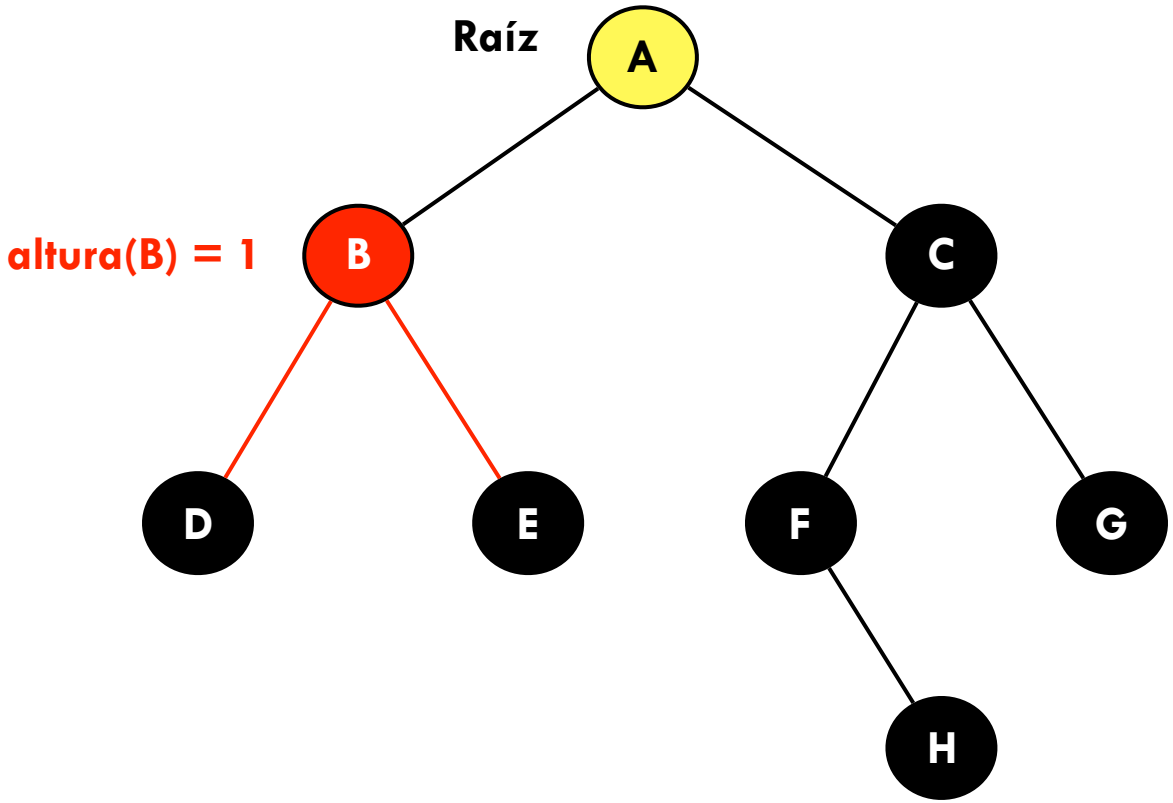


Altura

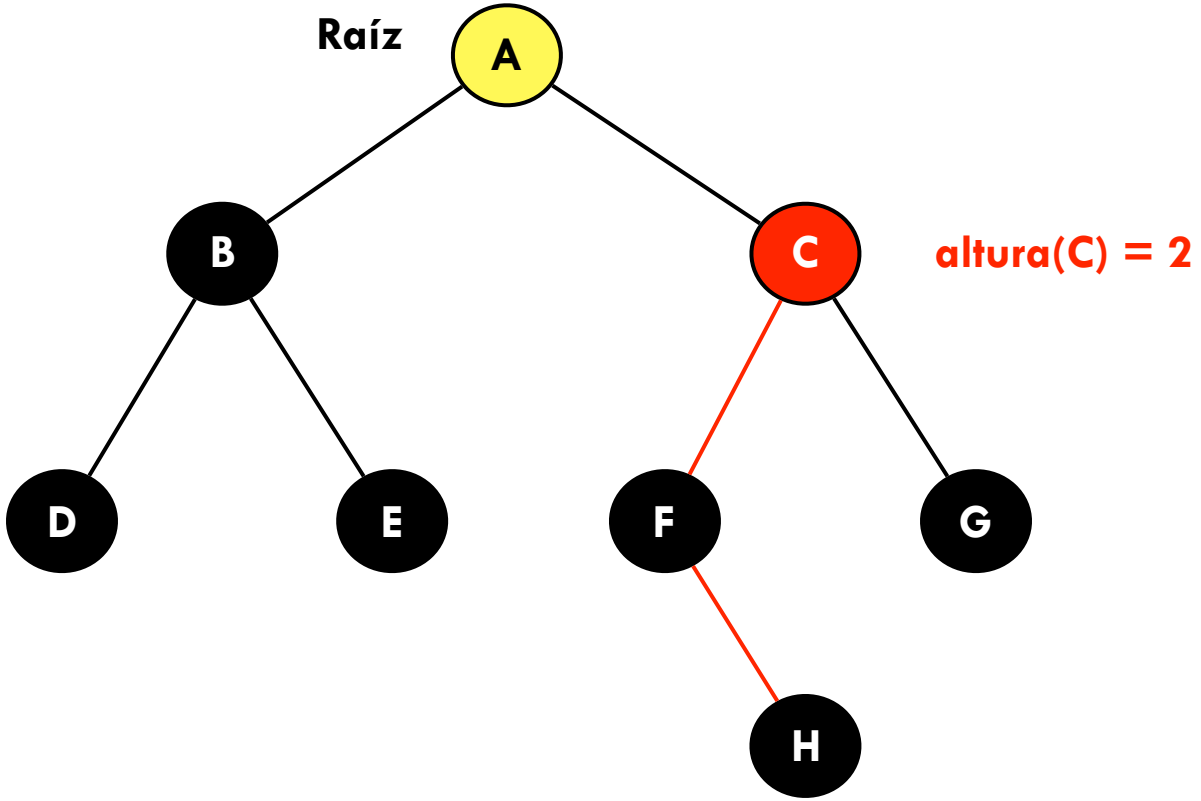


altura(D) = 0

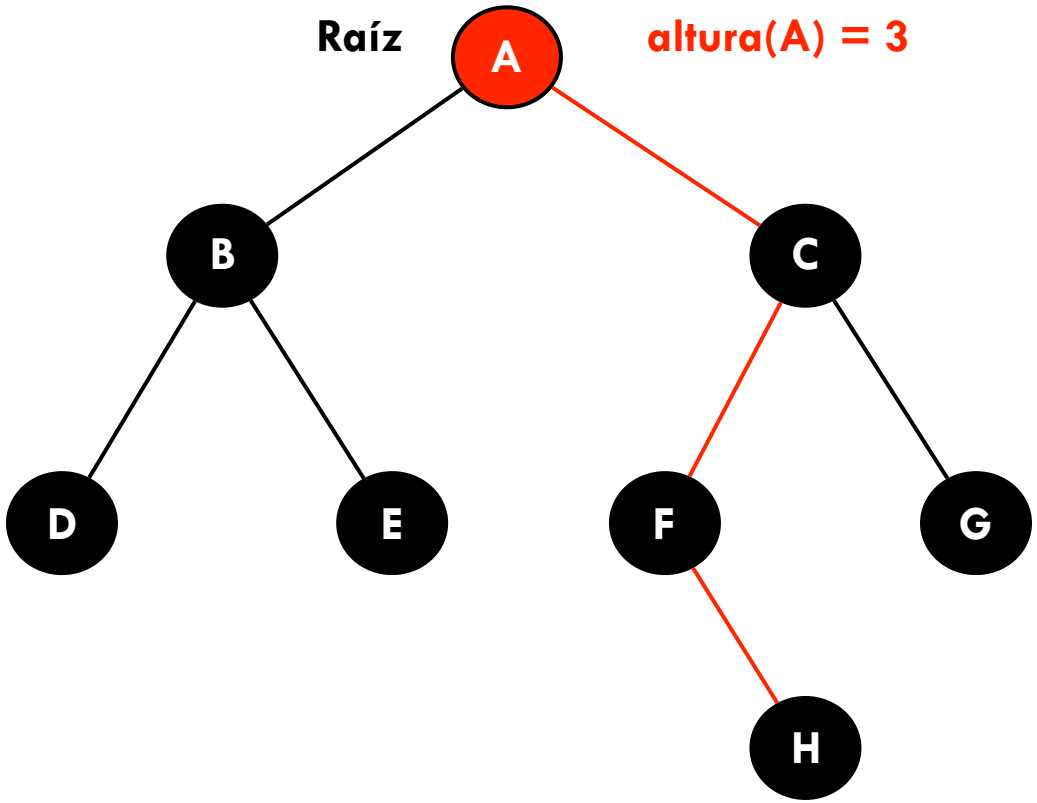
Altura



Altura



Altura



Exercício 01

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

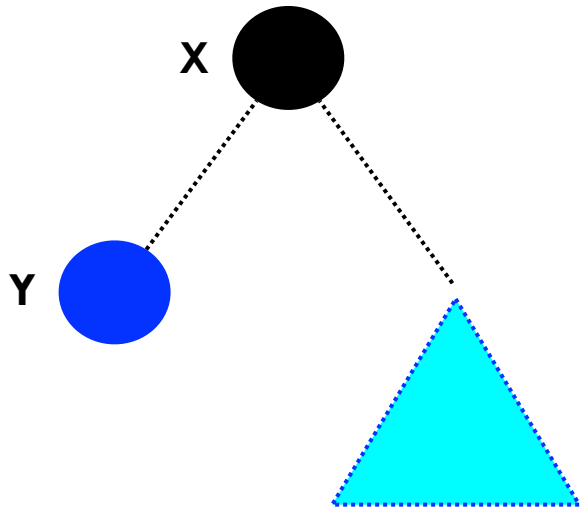
- Repita com a maior altura possível.

Árvores binárias de busca

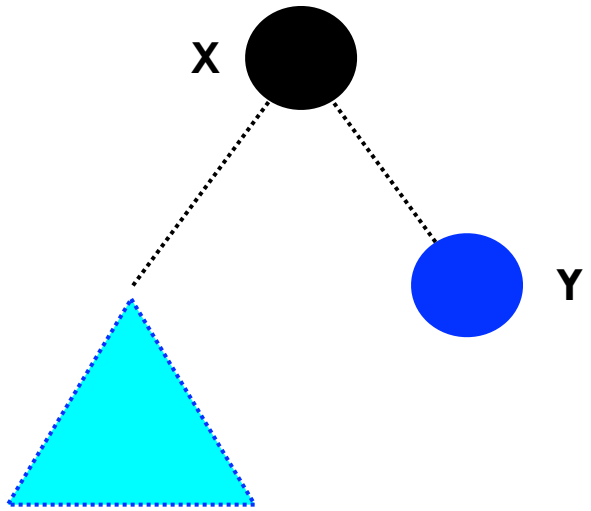
- **Propriedade**
  - Dados: nós de árvore X e Y
    - Se Y é um nó da subárvore esquerda de X,
      - então Y.chave $\leq$ X.chave.
    - Se Y é um nó da subárvore direita de X,
      - então Y.chave $>$ X.chave

Árvores binárias de busca

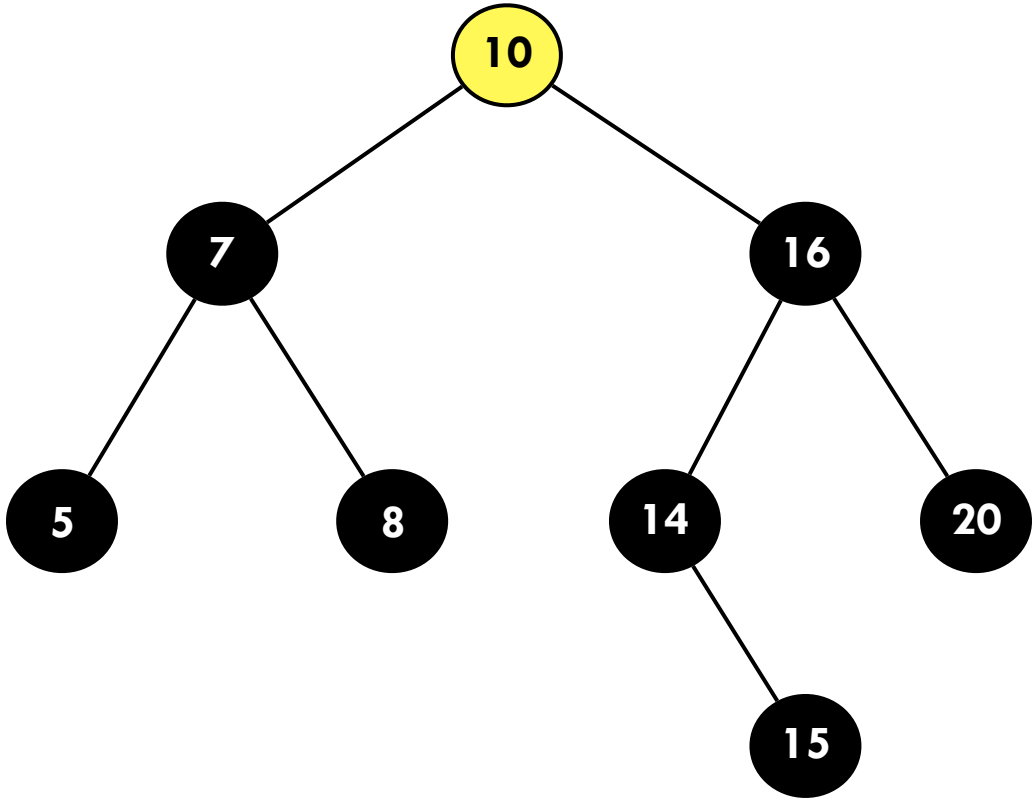
Se y.chave ≤ x.chave



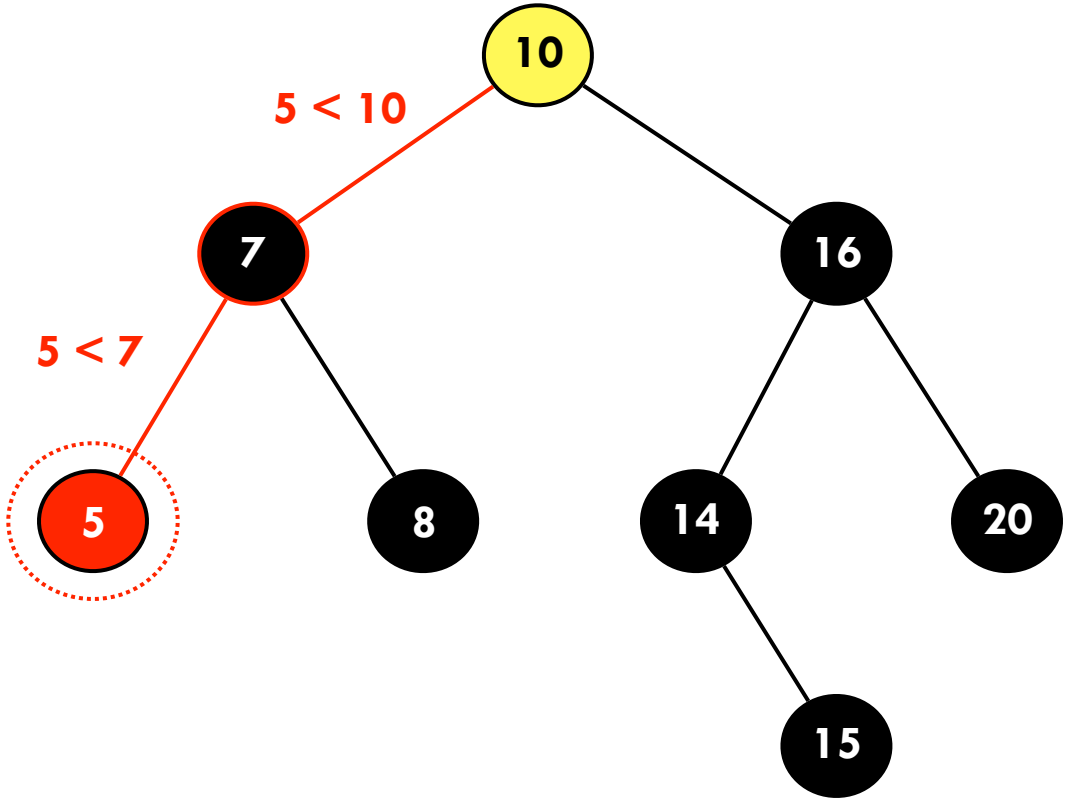
Se y.chave > x.chave



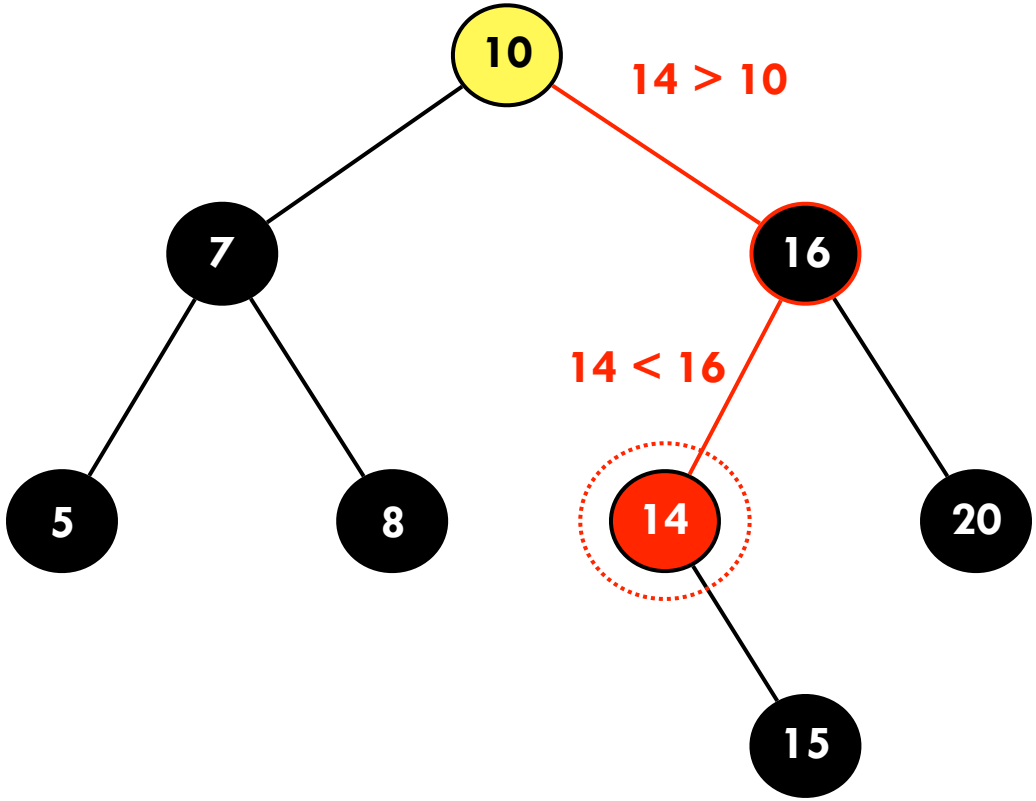
Exemplo



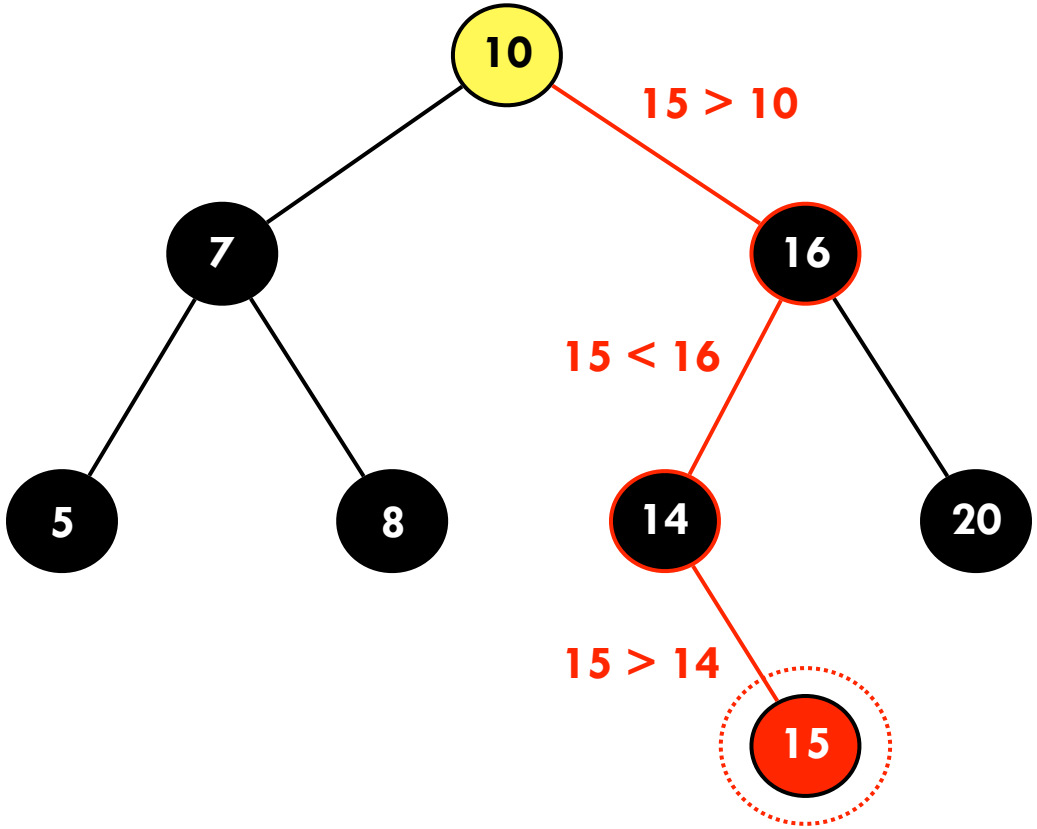
Exemplo



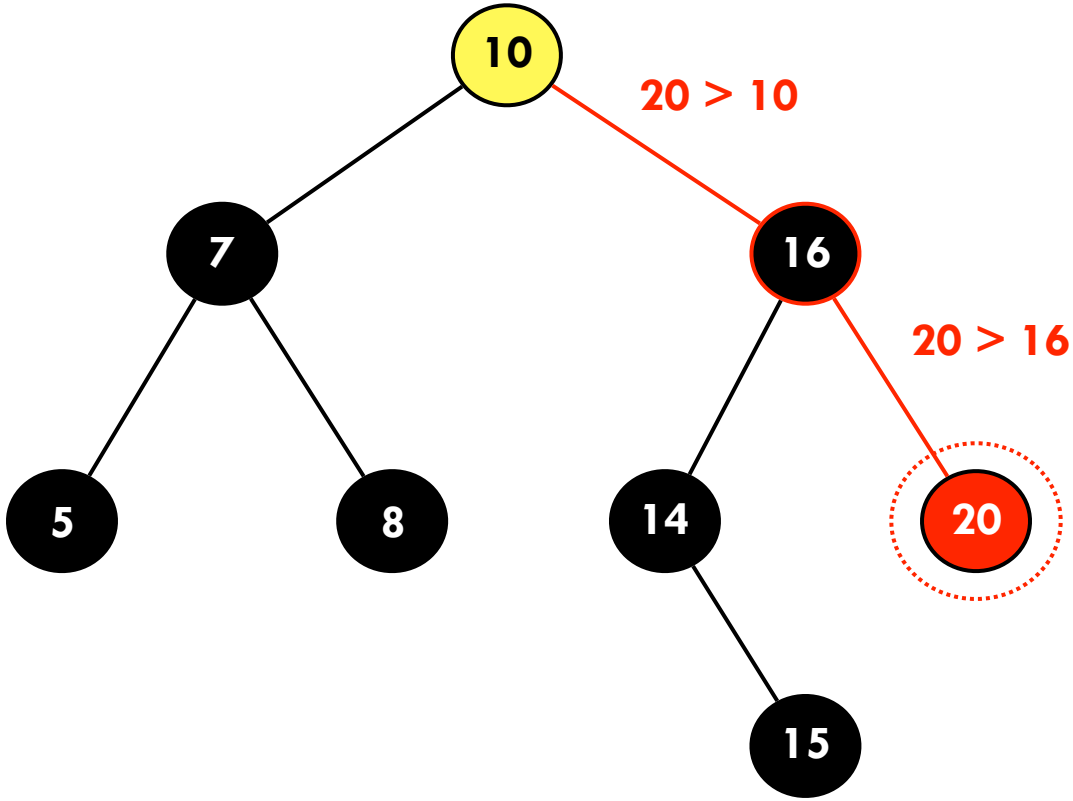
Exemplo



Exemplo



Exemplo



Exercício 02

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

Exercício 03

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

Roteiro

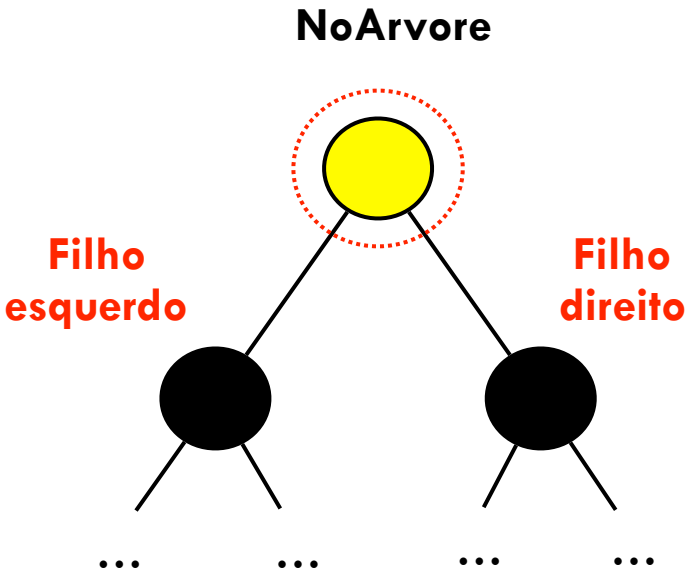
1. Introdução
2. Árvores Binárias
3. Propriedades e Definições
4. **Inserção em Árvores Binárias**
5. Pesquisa em Árvores Binárias
6. Referências

Operações

Dada uma árvore **T**, chave **k**, elemento **x**:

- **iniciar/destruir → iniciar e destruir a árvore**
- **pesquisar(S, k) → procurar k em S [TRUE/FALSE]**
- **inserir(S, k) → inserir k em S**
- **remover(S, k) → remover k em S**
- **minimo(S) → menor valor armazenado em S**
- **maximo(X) → maior valor armazenado em S**
- **proximo(S, x) → elemento sucessor a x**
- **anterior(S, x) → elemento antecessor a x**
- **tamanho(S) → tamanho de S**
- **vazia(S) → S está vazia? [TRUE/FALSE]**
- ~~cheia(S) → S está cheia? [TRUE/FALSE]~~
- **percorrer(T) : visitar os nós de T**

Tipos de Árvore Binária



tipo **NoArvore**

Item	Item armazenado
Dir	*NoArvore (direita)
Esq	*NoArvore (esquerda)

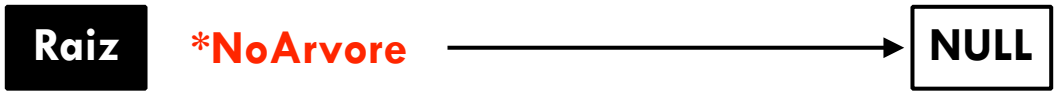
Raiz *NoArvore

Arvore Binária

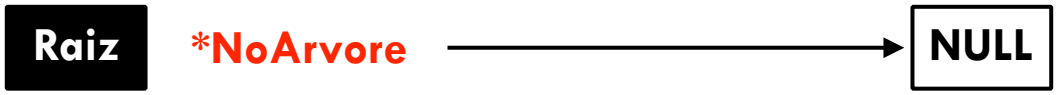
Tipos de Árvore Binária

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

typedef struct NoArvore *Ponteiro;

typedef struct NoArvore{
  Item elemento;
  Ponteiro direita;
  Ponteiro esquerda;
} NoArvore;

/* Definir arvore binária na main */

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

Inicialização



Inicialização




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

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

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

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

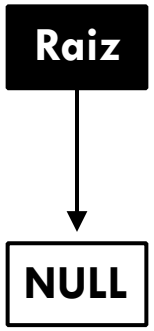
Inserção

- 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

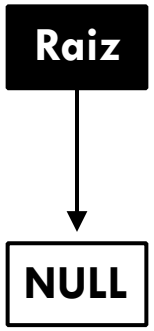
Inserção

Versão:
- Iterativa
- Recursiva



Inserção

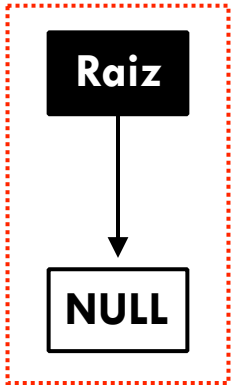
Versão:
- Iterativa
- Recursiva



Inserir x = **5**

Inserção

Versão:
- Iterativa
- Recursiva

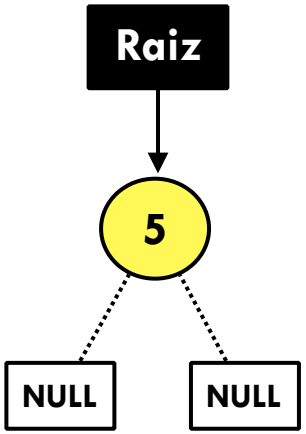


Árvore vazia

Inserir x = **5**

Inserção

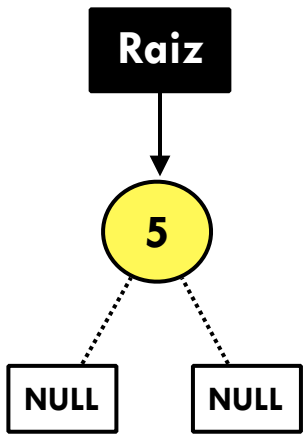
Versão:
- Iterativa
- Recursiva



Inserir x = **5**

Inserção

Versão:
- Iterativa
- Recursiva

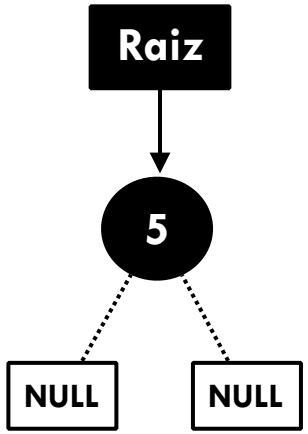


Inserir x = **5**

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

Inserção

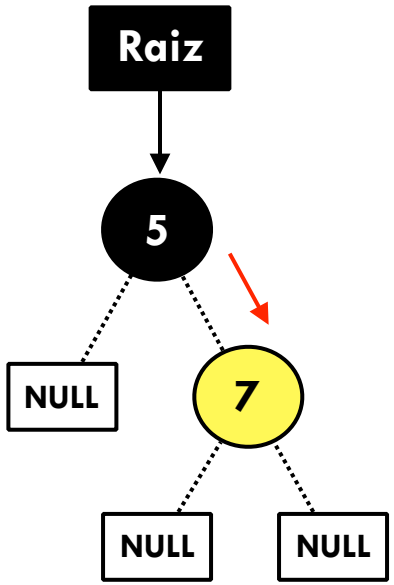
Versão:
- Iterativa
- Recursiva



Inserir x = *7*

Inserção

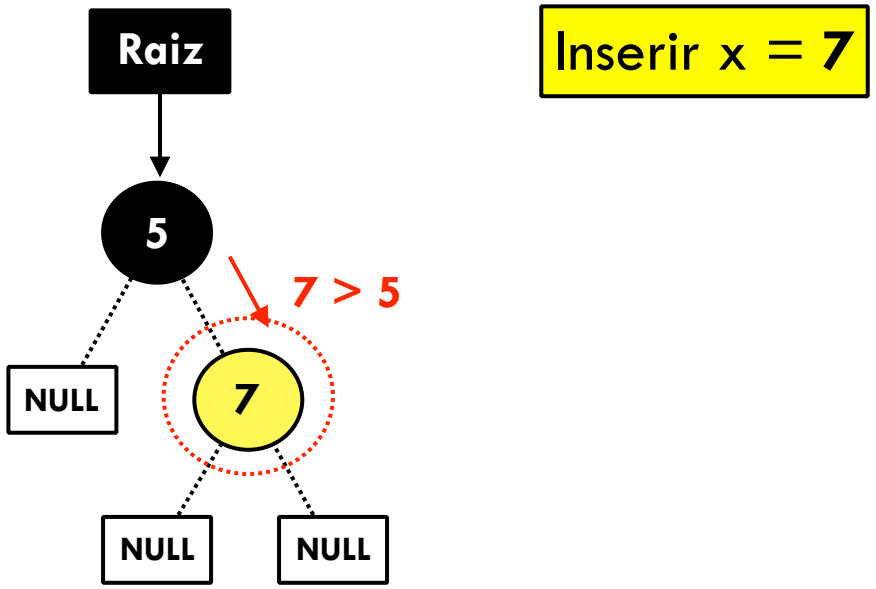
Versão:
- Iterativa
- Recursiva



Inserir x = *7*

Inserção

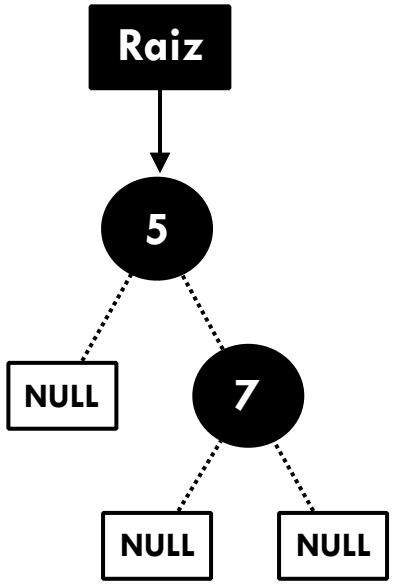
Versão:
- Iterativa
- Recursiva



7 > elemento na raiz (5)
* deve ser inserido na sub-arvore **direita**
* assim mantém-se a propriedade de busca

Inserção

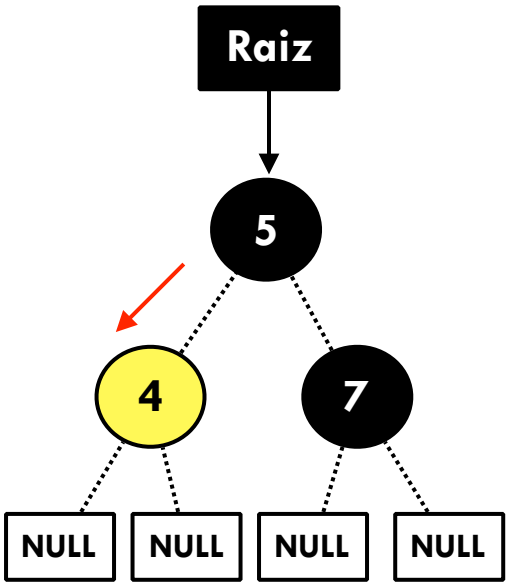
Versão:
- Iterativa
- Recursiva



Inserir x = **4**

Inserção

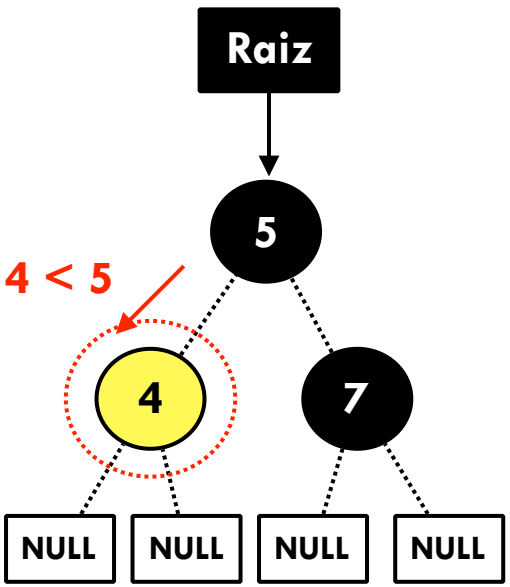
Versão:
- Iterativa
- Recursiva



Inserir x = **4**

Inserção

Versão:
- Iterativa
- Recursiva

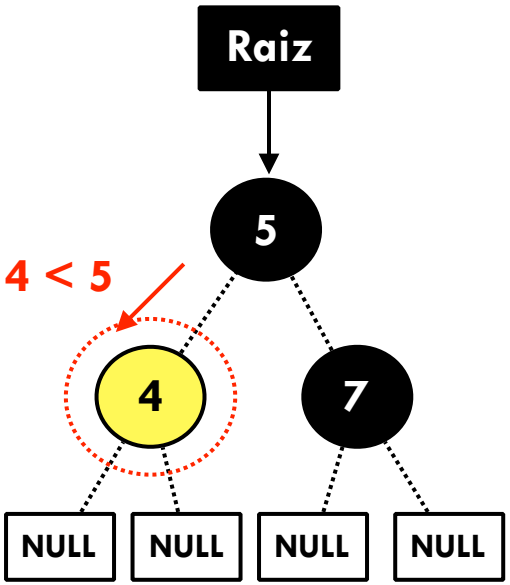


Inserir x = **4**

4 < elemento na raiz (5)
* deve ser inserido na sub-arvore **esquerda**
* assim mantém-se a propriedade de busca

Inserção

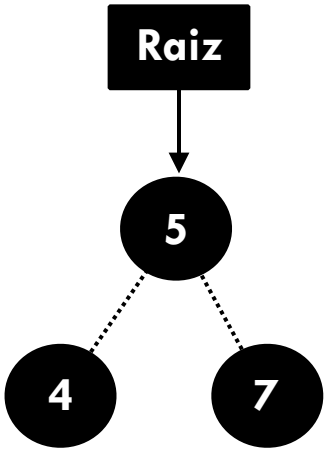
Versão:
- Iterativa
- Recursiva



Inserir x = **4**

Inserção

Versão:
- Iterativa
- Recursiva



Inserir x = **3**

Inserir x = **2**

Inserir x = **6**

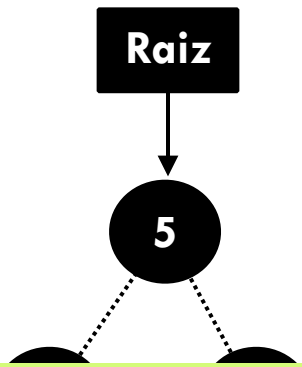
Inserir x = **8**

Inserir x = **9**

Como fica a árvore?

Inserção

Versão:
- Iterativa
- Recursiva



Inserir x = 3

Inserir x = 2

Inserir x = 6

```
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 (Ponteiro *arvore, Item 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 (Ponteiro *arvore, Item 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 TAB de Árvore Binária e a inserção de elementos.
- Se preciso, implementar funções auxiliares para checar se a estrutura está correta.

Exercício 05

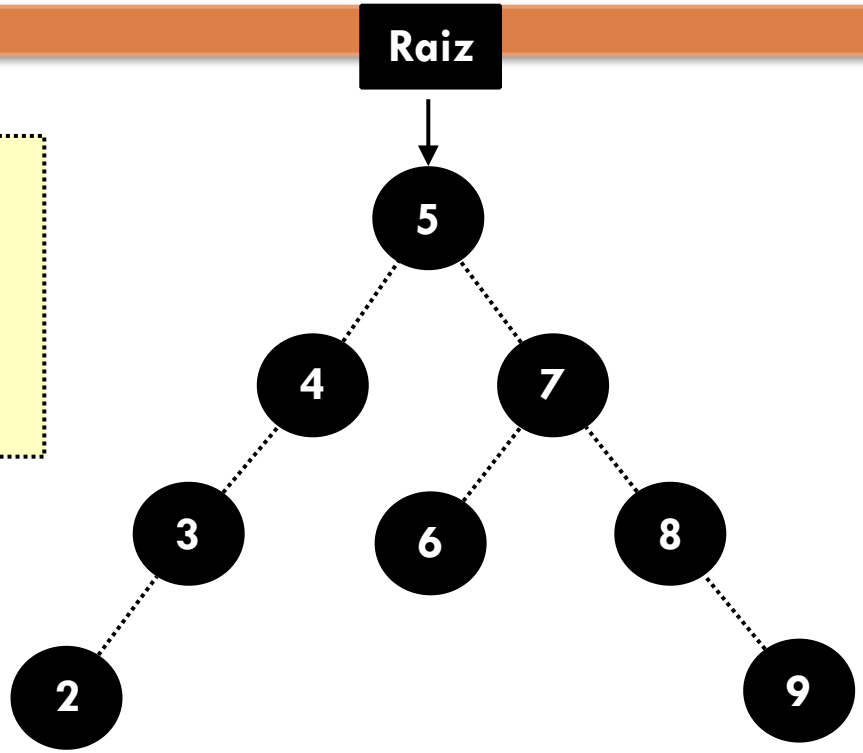
- Como escrever uma versão iterativa para a função de inserção em uma árvore binária?

Roteiro

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

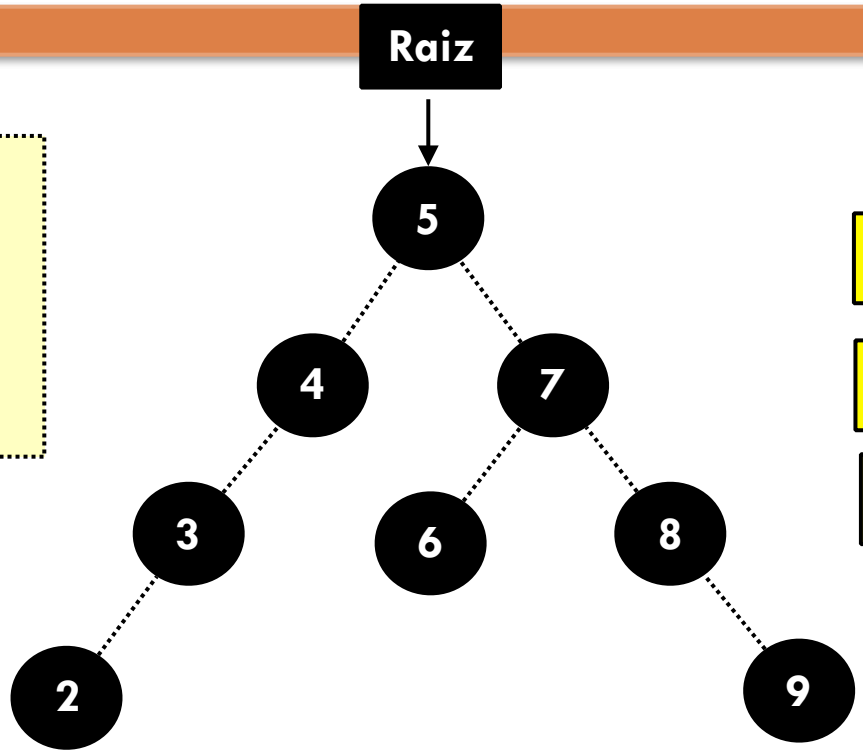
Percursos

Útil para:
- Imprimir
- Consultar
- Remover
- Inserir



Percursos

Útil para:
- Imprimir
- Consultar
- Remover
- Inserir

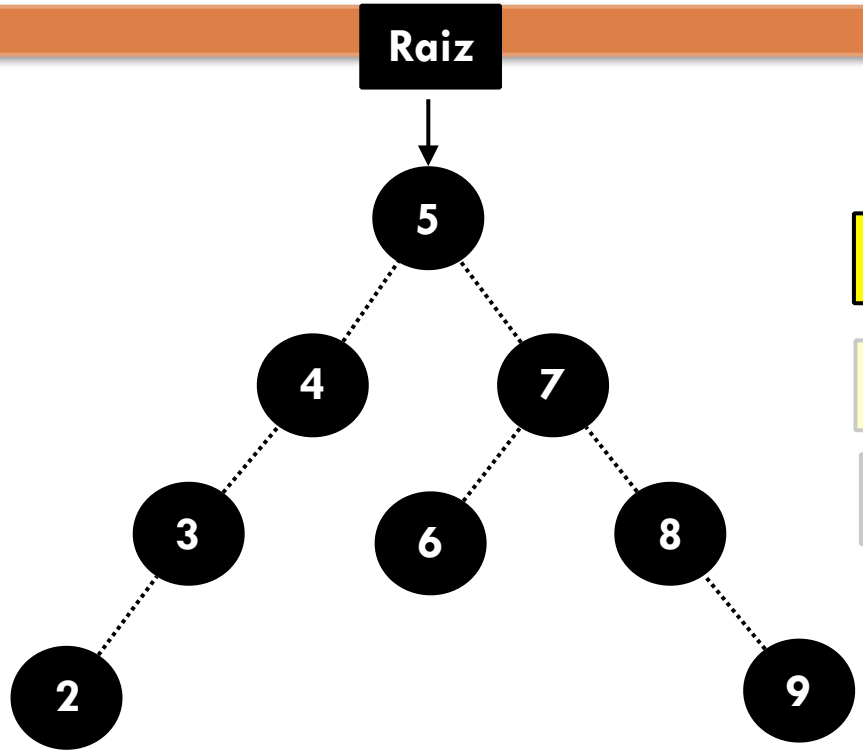


Pre-Ordem

Pos-Ordem

Em-Ordem

Percurso: Pre-ordem

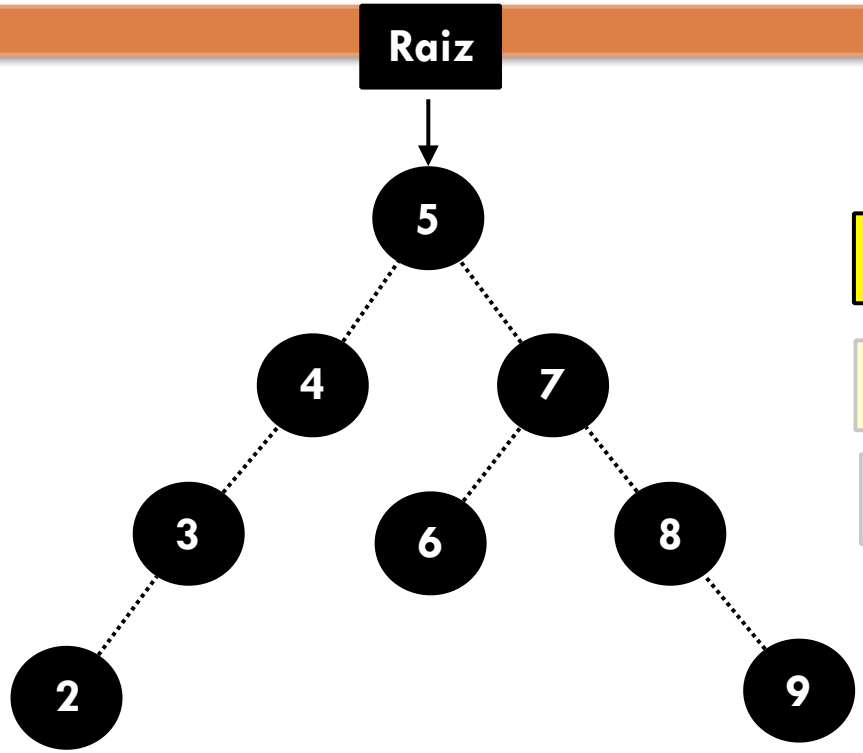


Pre-Ordem

Pos-Ordem

Em-Ordem

Percurso: Pre-ordem



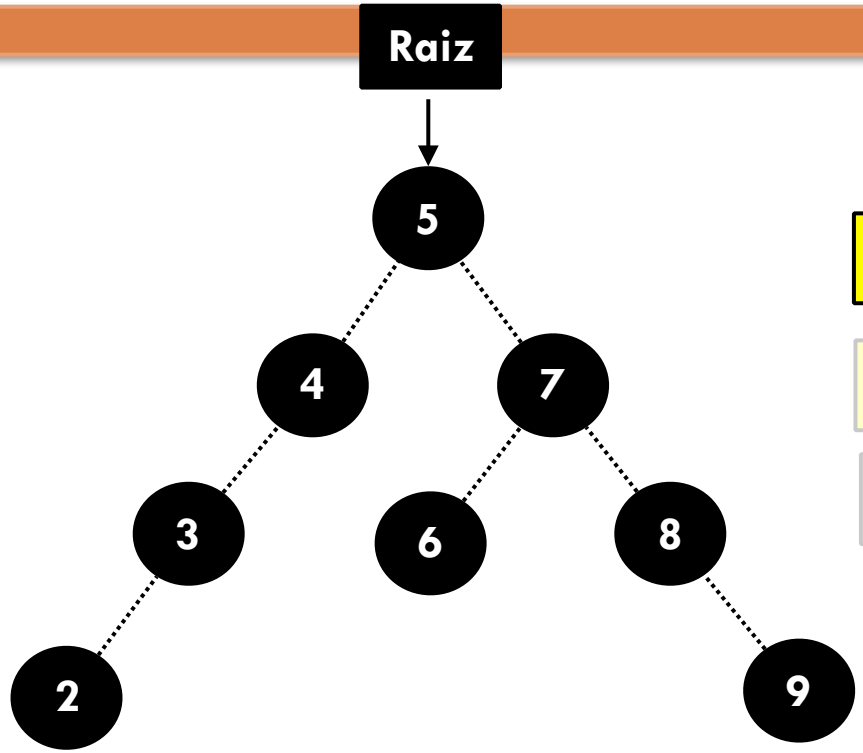
Pre-Ordem

Pos-Ordem

Em-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

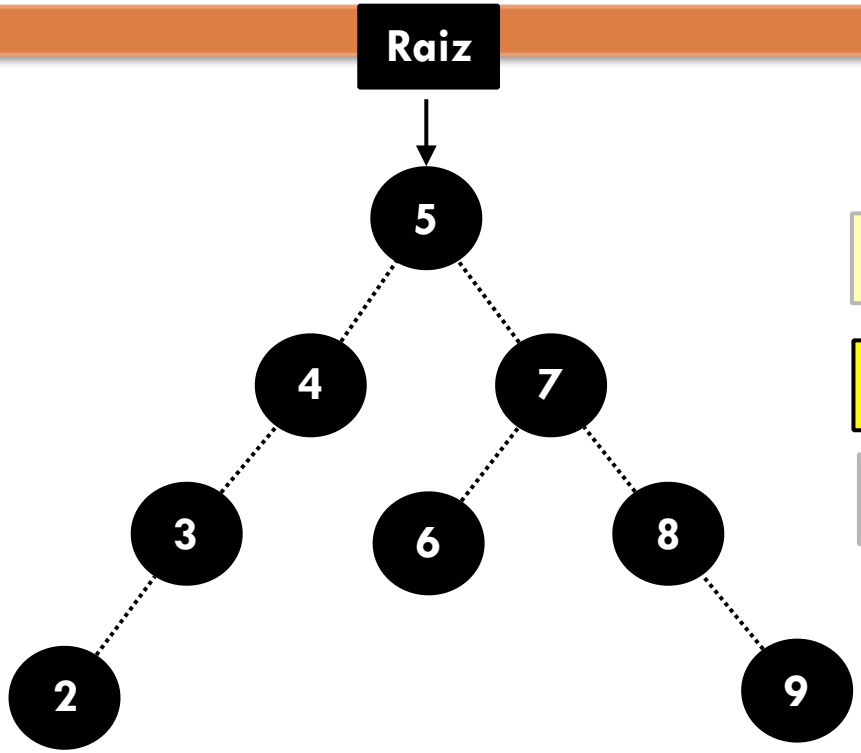
Pos-Ordem

Em-Ordem

Percurso = {5, 4, 3, 2, 7, 6, 8, 9}

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

Percurso: Pos-ordem

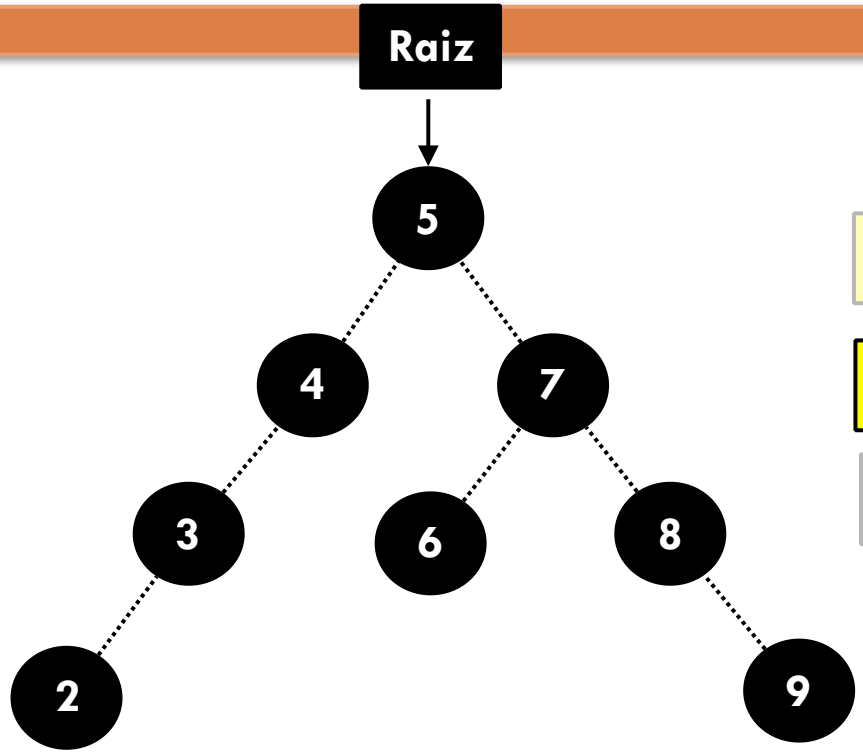


Pre-Ordem

Pos-Ordem

Em-Ordem

Percurso: Pos-ordem



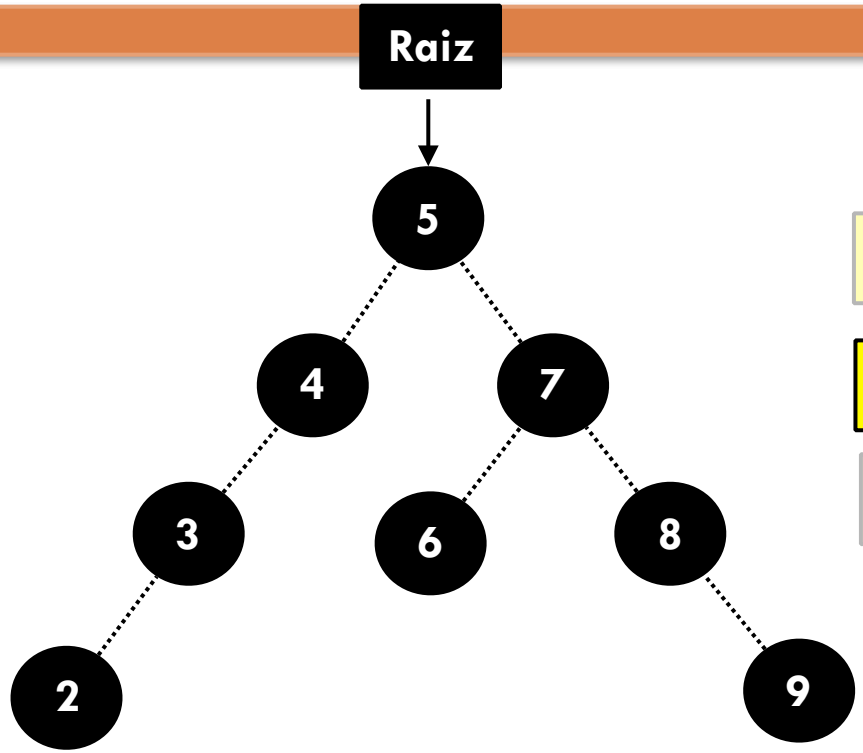
Pre-Ordem

Pos-Ordem

Em-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



Pre-Ordem

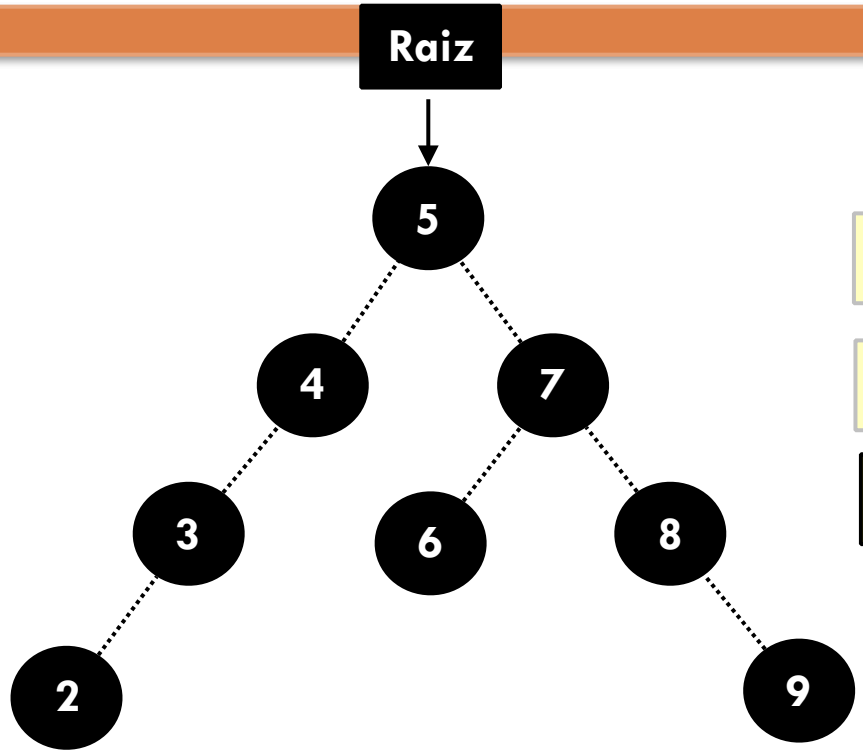
Pos-Ordem

Em-Ordem

Percurso = {2, 3, 4, 6, 9, 8, 7, 5}

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

Percursos

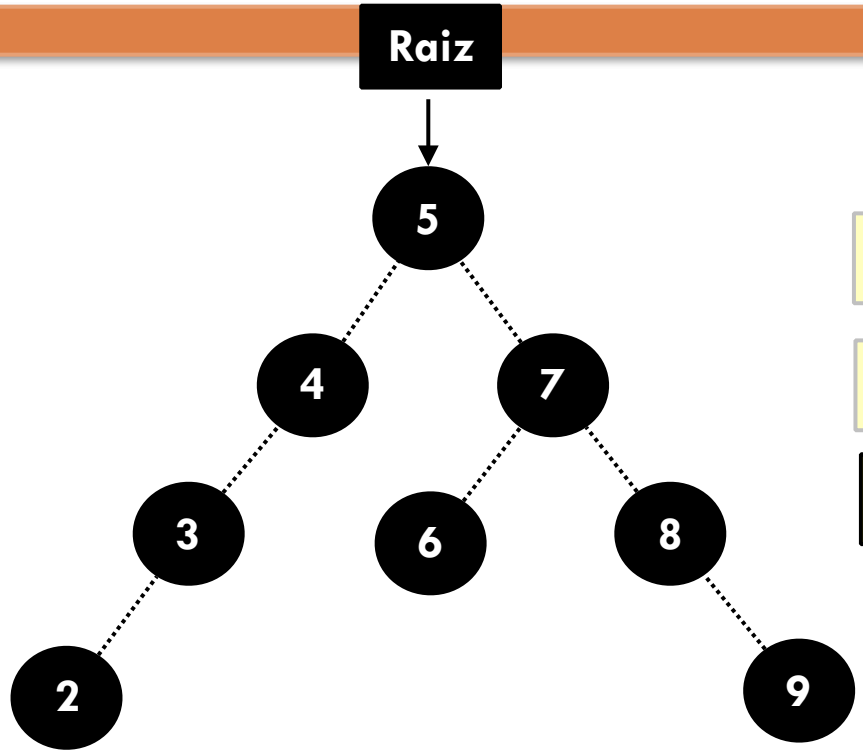


Pre-Ordem

Pos-Ordem

Em-Ordem

Percursos



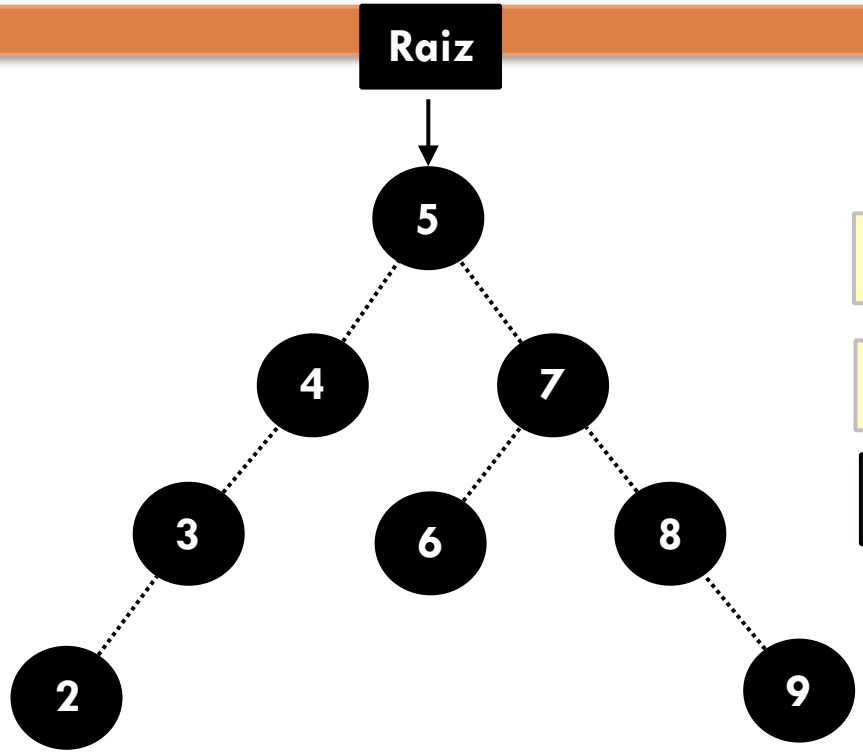
Pre-Ordem

Pos-Ordem

Em-Ordem

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

Percursos



Pre-Ordem

Pos-Ordem

Em-Ordem

Percurso = {2, 3, 4, 5, 6, 7, 8, 9}

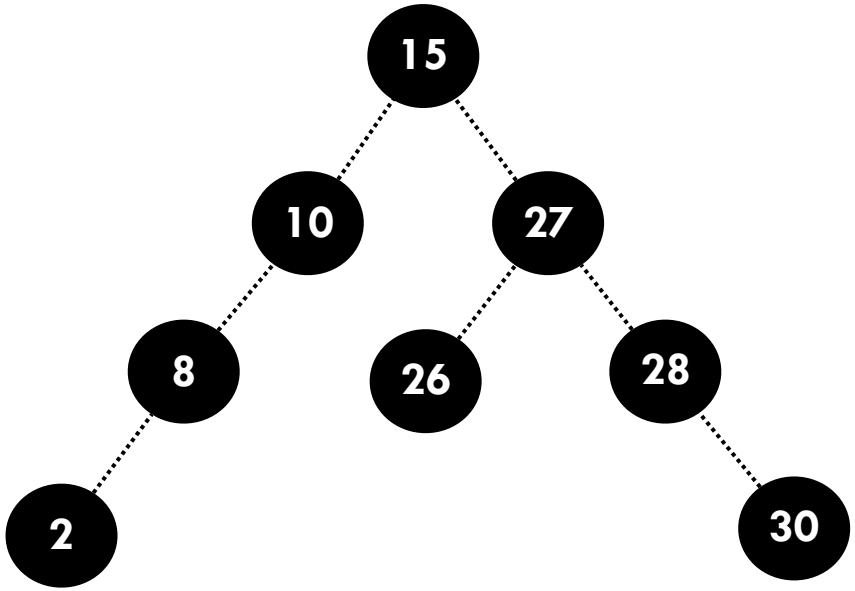
```
Em-Ordem (*arvore)
    1. Em-Ordem((*arvore)->esquerda)
    2. imprime o valor do nó corrente // (*arvore)
    3. Em-Ordem((*arvore)->direita)
```

Consultas / Pesquisa

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

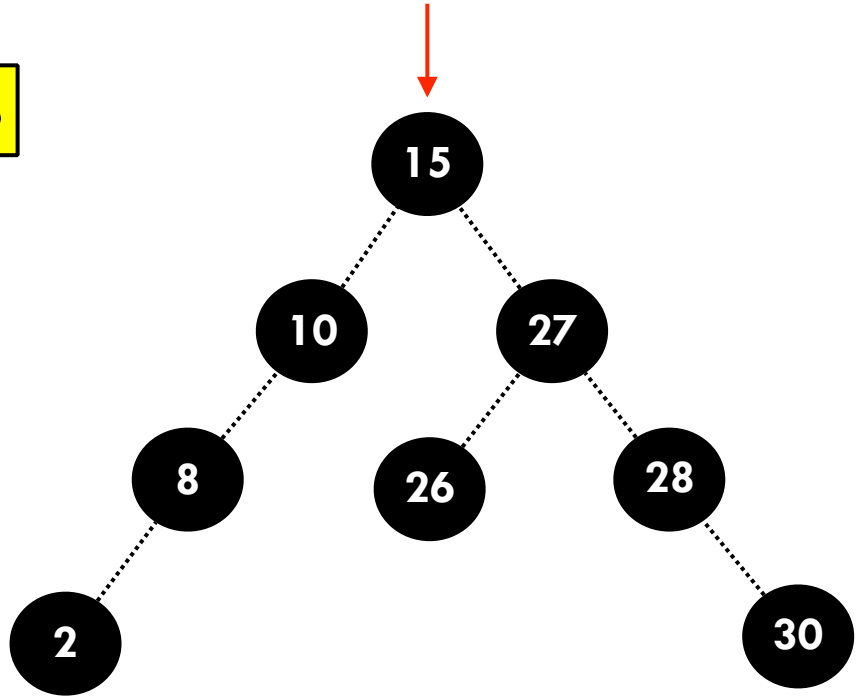
- acessar raiz
  - percorrer a arvore usando a propriedade de busca
  - encontrar a posição do elemento no arranjo

Consultas / Pesquisa



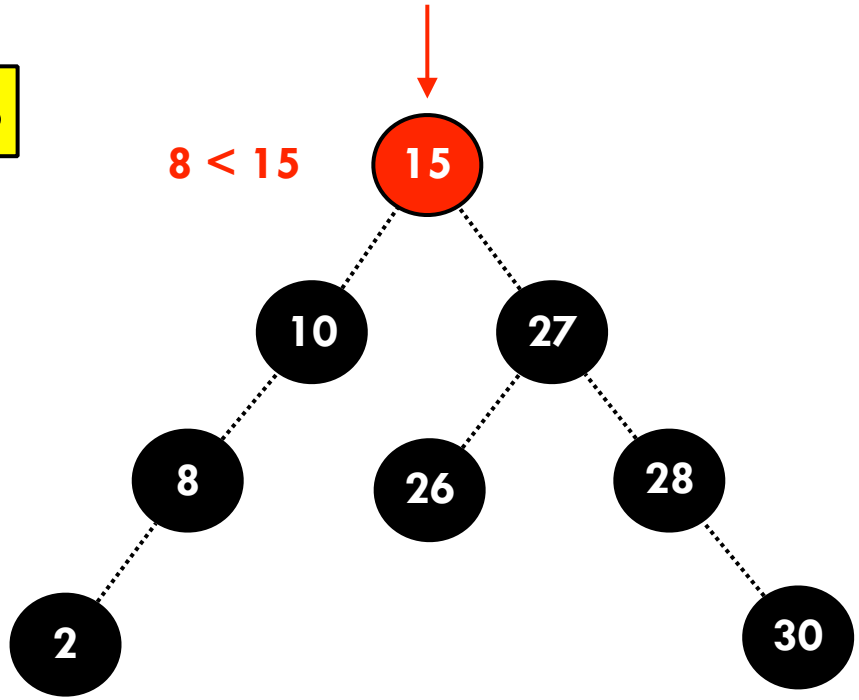
Consultas / Pesquisa

Procurar x = **8**



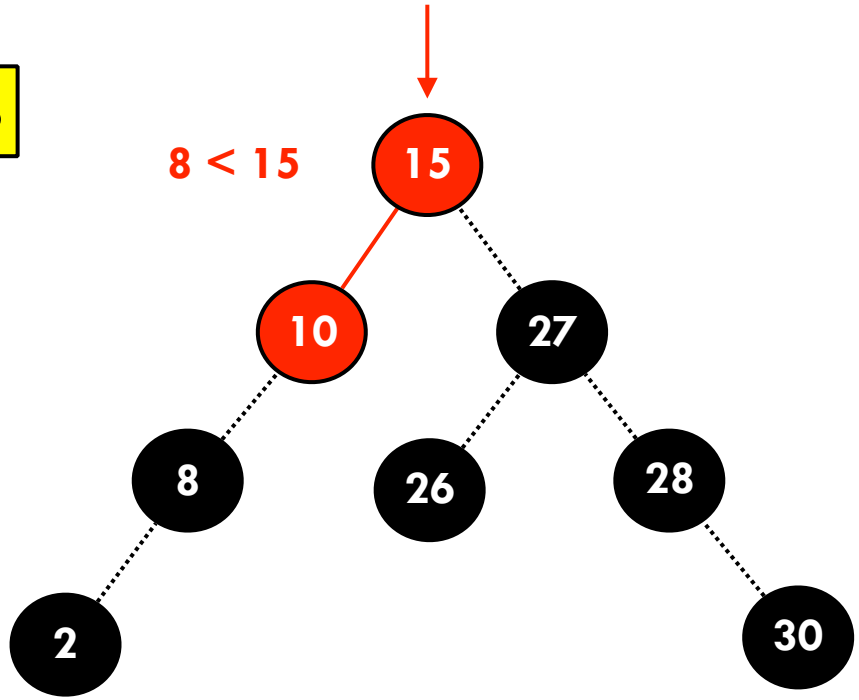
Consultas / Pesquisa

Procurar x = **8**



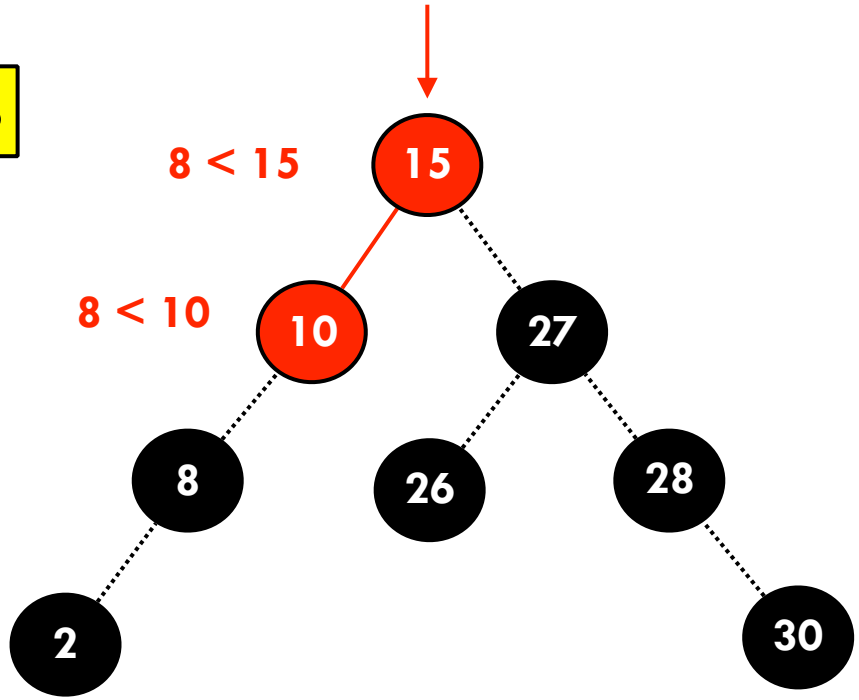
Consultas / Pesquisa

Procurar x = **8**



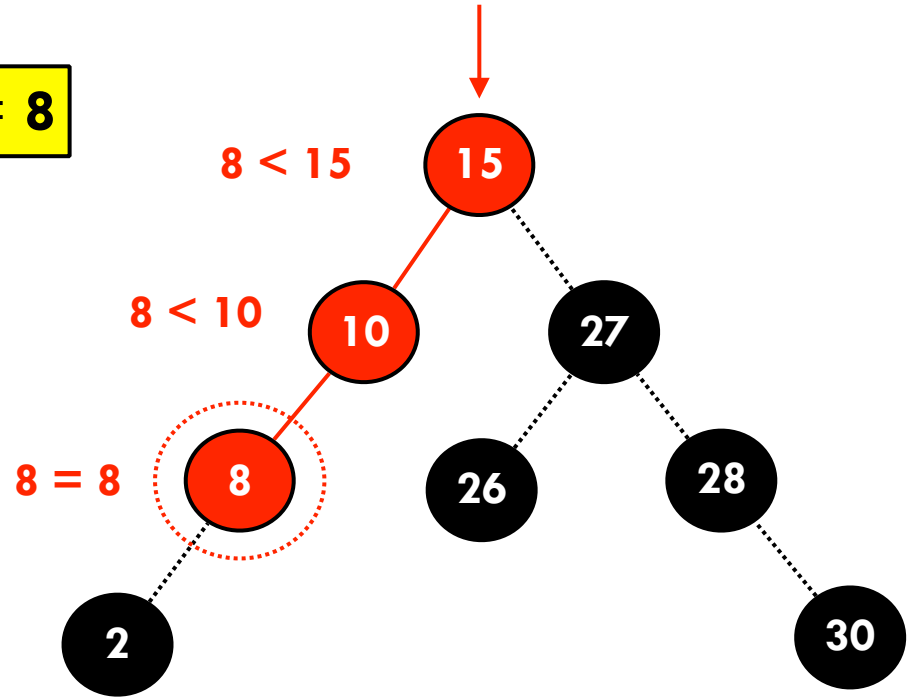
Consultas / Pesquisa

Procurar x = **8**



Consultas / Pesquisa

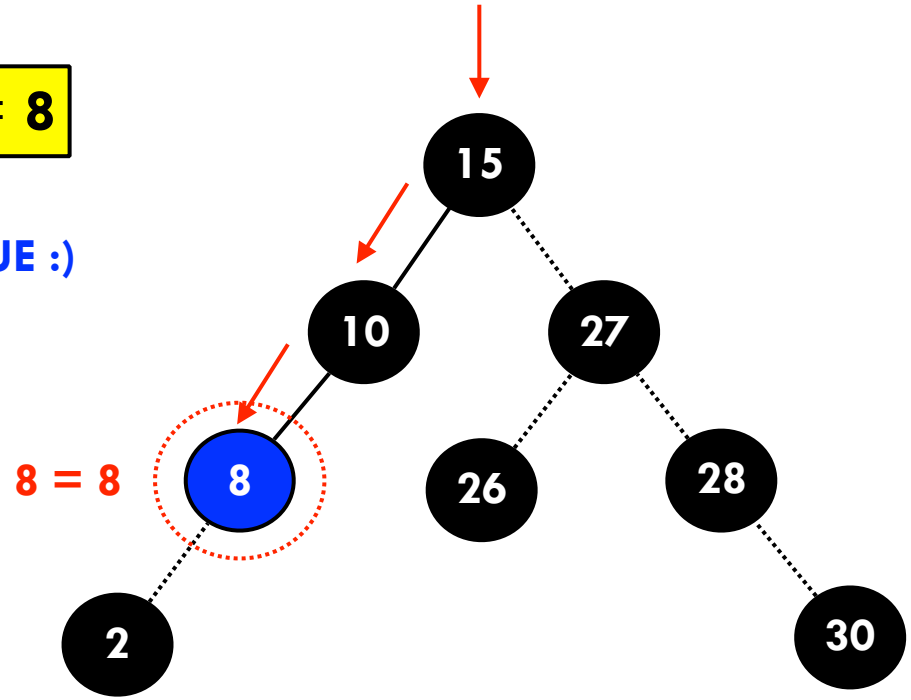
Procurar x = **8**



Consultas / Pesquisa

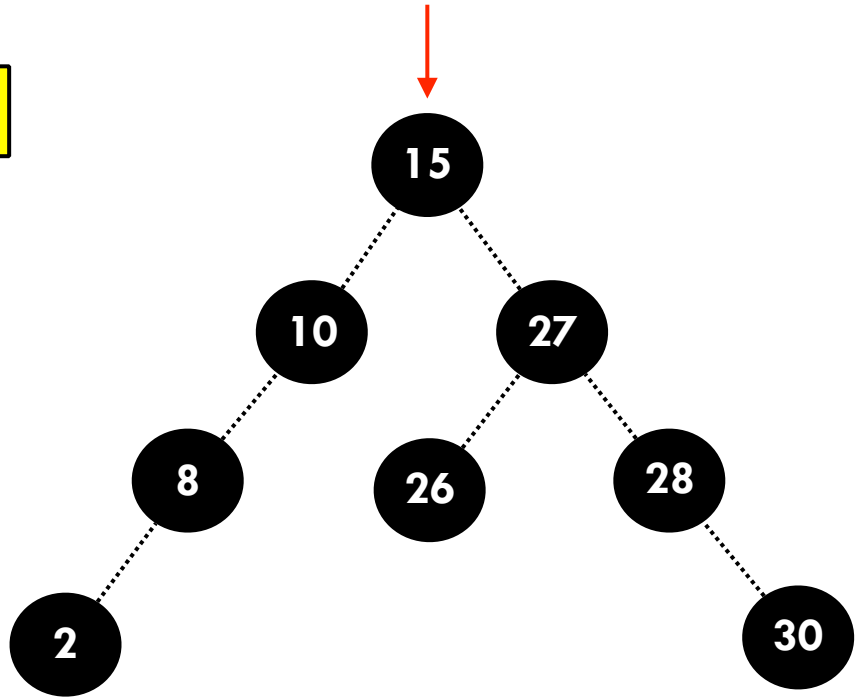
Procurar x = **8**

Resultado: **TRUE :)**



Consultas / Pesquisa

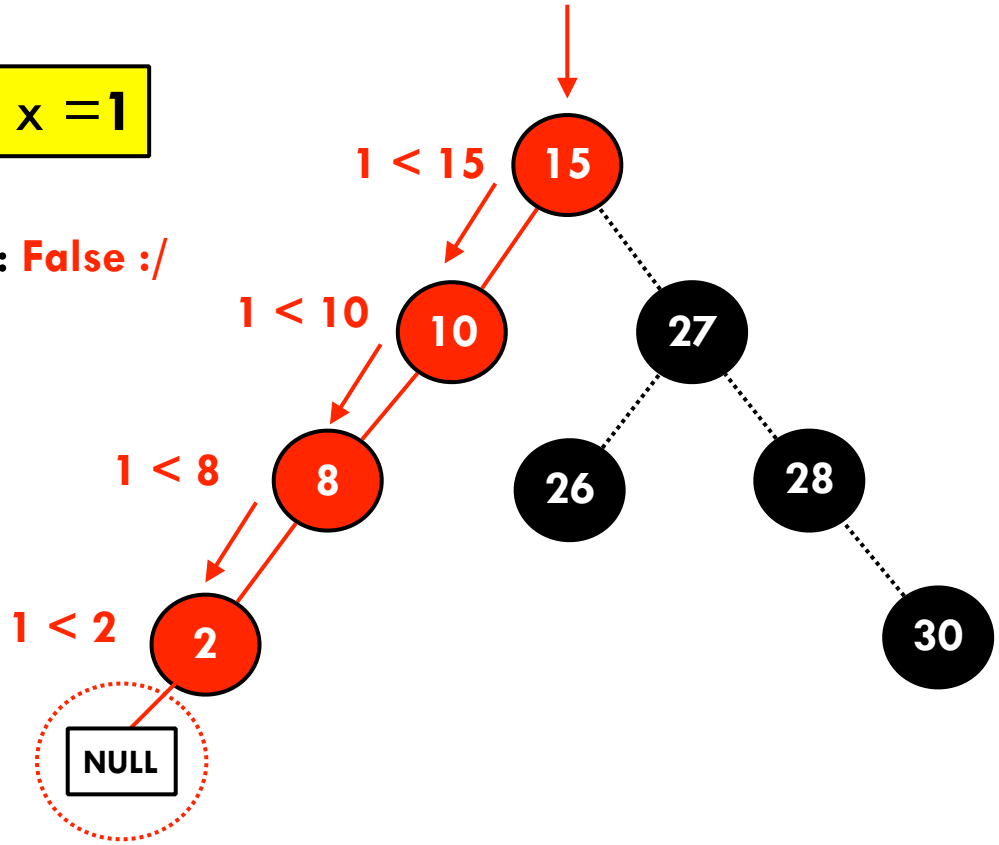
Procurar x =1



Consultas / Pesquisa

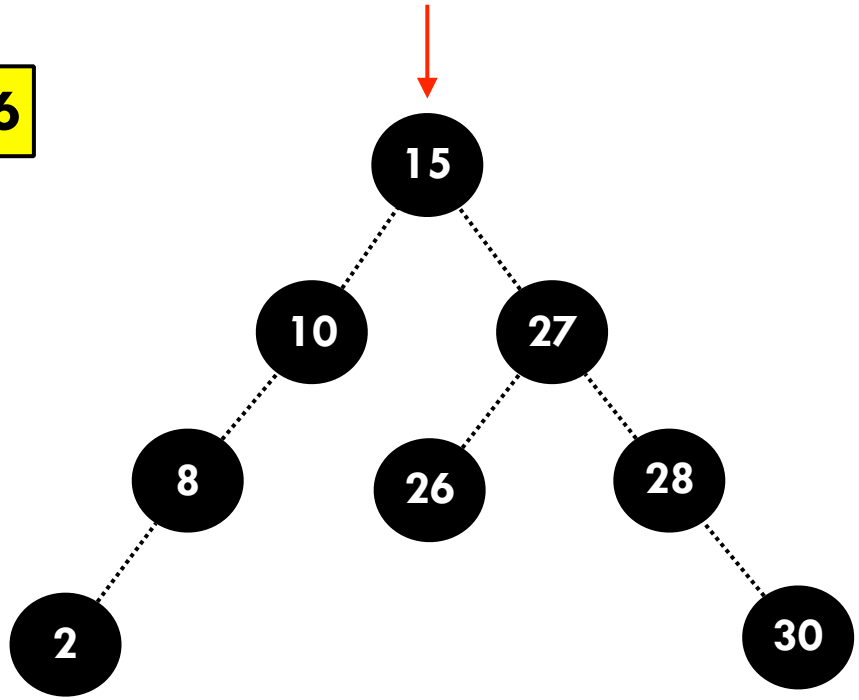
Procurar x =1

Resultado: **False :/**



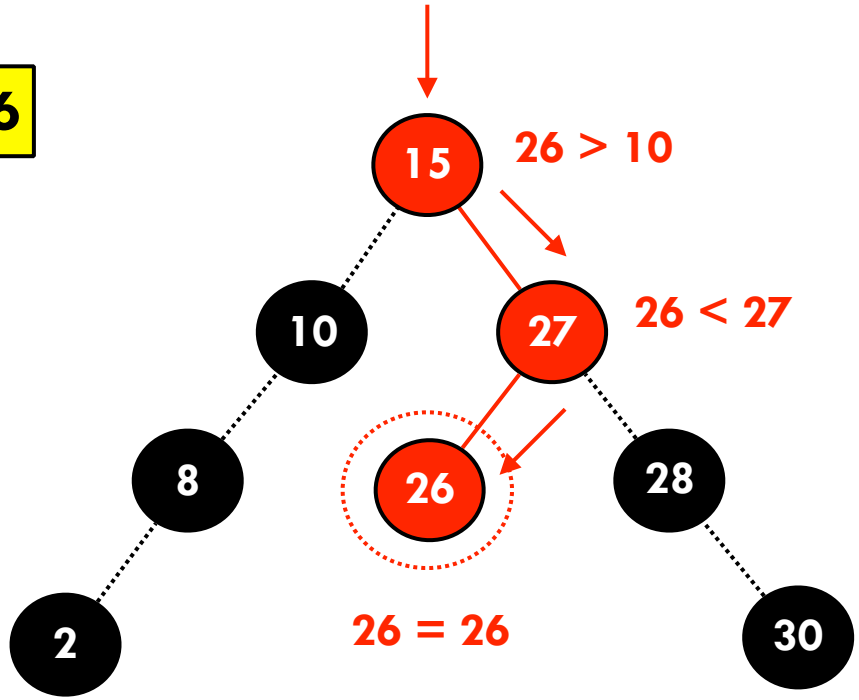
Consultas / Pesquisa

Procurar x = **26**



Consultas / Pesquisa

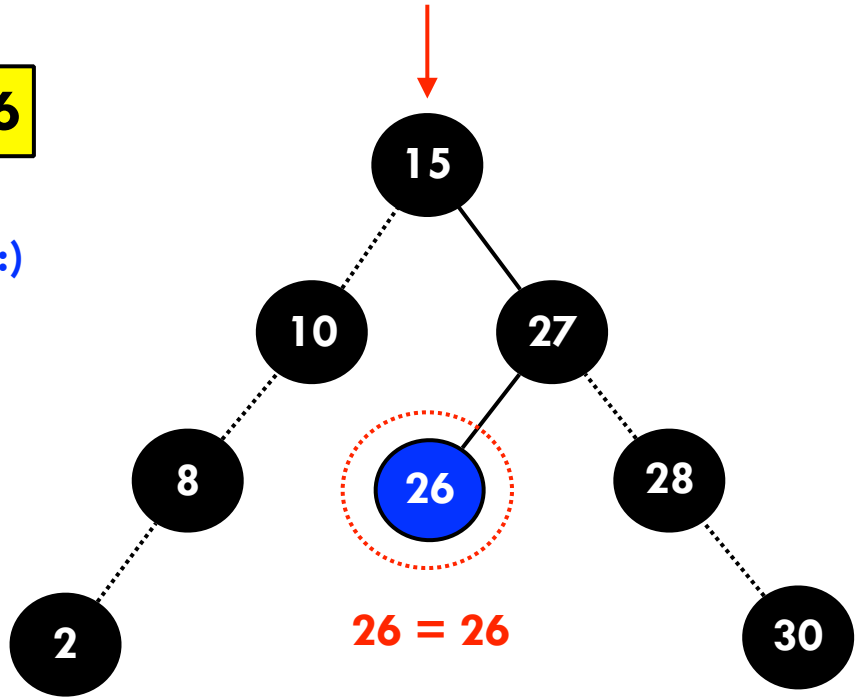
Procurar x = **26**



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 procuraItem(Ponteiro *arvore, int chave, Item *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 procuraItem(Ponteiro *arvore, int chave, Item *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

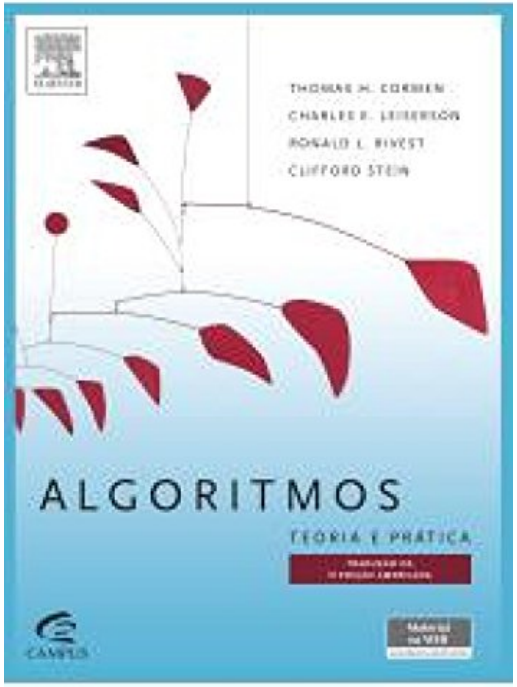
Exercício 06

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

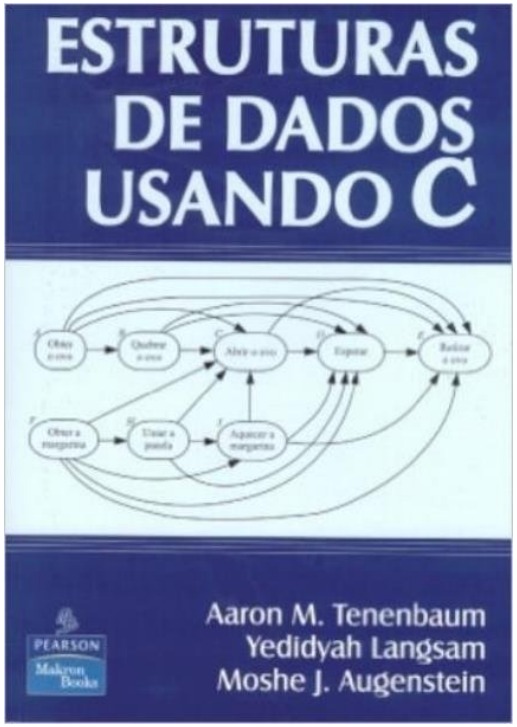
Roteiro

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

Referências sugeridas

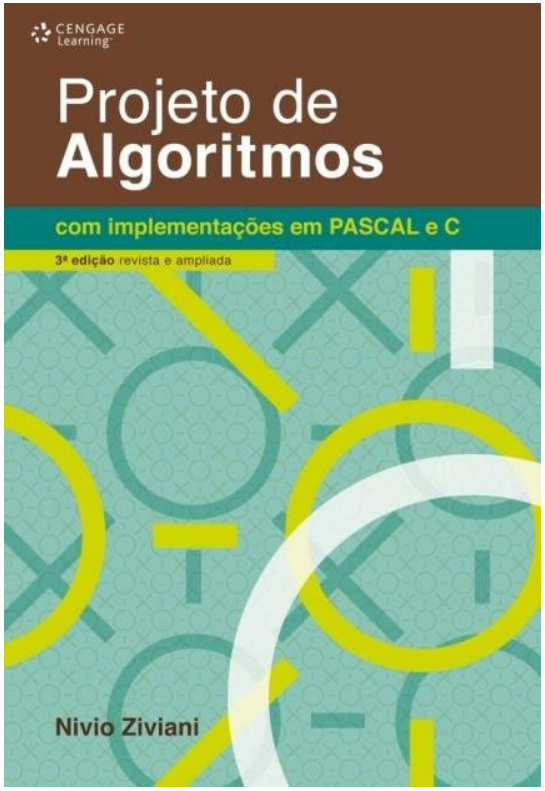


[Cormen et al, 2018]



[Tenenbaum et al, 1995]

Referências sugeridas



[Ziviani, 2010]

[Drozdek, 2017]

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br

Minimo

- Pseudocodigo

- Tree-Minimum(x) —> x é a raíz
- while x.esqueda != NULL
  - x = x.esquerda;
  - return x

  - O(h)

Maximo

- Pseudocodigo

- Tree-Maximum(x) —> x é a raíz
- while x.direita != NULL
  - x = x.direita;
- return x

- O(h)

Exercicio extra

- Escreva versões recursivas de maiorElemento(), e menorElemento()

Inserção - Iterativo

- Inserção / Pseudocógido
- Insert(T, z)
  - y = NULL
  - x = T.raiz
  - while x!= NULL
    - y = x
    - if(z.chave < x.chave)
      - x = x.direita
    - else x = x.direita
  - z.p = y
  - if y == NUL, T.raiz = z // a arvore era vazia
  - else if(z.chave < y.chave)
    - y.esquerda = z
  - else y.direita = z