

• Árvores binárias

* Estruturas

```
1. typedef struct NoArvore *PtrArvore;  
2. typedef struct NoArvore {  
3.     int chave;  
4.     PtrArvore direita;  
5.     PtrArvore esquerda;  
6. } NoArvore;  
// ...  
7. int main(...){  
8.     PtrArvore raiz; } uso de estrutura de árvore na  
9.     ... main  
10. }
```

estrutura básica de
nó



* Inserções (Recursiva)

```
bool Inserção (ptrArvore *arvore, int x){
```

a) Se $*\text{arvore} == \text{NULL}$:

$(*arvore) = \text{malloc}(\text{sizeof}(\text{NoArvore}))$;

$(*arvore) \rightarrow \text{direita} = (*arvore) \rightarrow \text{esquerda} = \text{NULL}$;

$(*arvore) \rightarrow \text{chave} = x$;

return (true);

b) Se $(*\text{arvore}) \rightarrow \text{chave} == x$

return (false)

c) Se $(*\text{arvore}) \rightarrow \text{chave} > x$

return (Inserção (&(*arvore) \rightarrow esquerda, x));

Serão

return (Inserção (&(*arvore) \rightarrow direita, x));

fin

Em a): sempre que cair em uma folha (primeira ou N-ésima inserção), adicionamos o novo nó.

Em b): não inserir chaves duplicadas na estrutura

a e b são condições de parada da recursão

Consulta / Procura (Recursiva)

```
bool Procurar (PtrArvore *arvore, int x) {
```

(a) Se $*\text{arvore} == \text{NULL}$, false

(b) se $(*\text{arvore}) \rightarrow \text{chave} == x$, true

(c) Se $(*\text{arvore}) \rightarrow \text{chave} > x$:

return (Procurar (&(*arvore) \rightarrow esquerda, x));

Senão

him return (Procurar (&(*arvore) \rightarrow direita, x));

Em a): condições de parada, não achou elemento.
Atingiu uma folha, logo se o nó/elemento existisse,
se, deveria estar ali.

Em b): condição de parada, achou o elemento
a e b são condições de parada da recursão

Em c) temos a recursão para percorrer/se deslocar
pela árvore

Resumo da Inserção:

- a) encontrar uma folha, e inserir o nó (true)
- b) encontrar uma chave já existente, obter false
- c) chamadas recursivas percorrendo a árvore
 - se no.chave > x
chamar recursivo para esquerda
 - se no.chave < x
chamar recursivo para direita

Resumo da Consulta:

- a) encontrar uma folha com filhos nulos (null), o nó/elemento não existe
- b) encontrar o valor desejado : sucesso
- c) chamadas recursivas percorrendo a árvore , como na inserção
 - Se no.chave > x : chamar recursão p/ esquerda
 - Se no.chave < x: chamar recursão p/ direita

obs: Se forem registros dentro da árvore, modificar função para retornar struct por referência

Percursos em Árvores

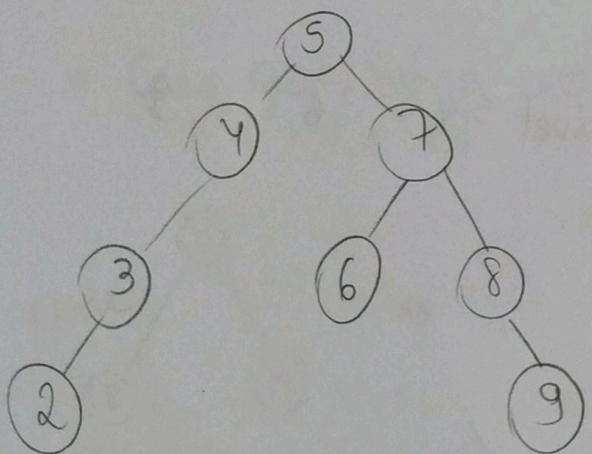
01 Pré-Ordem

```
// if (*node == NULL) return;
```

- a) Imprimir valor do nó corrente
- b) Pré-Ordem (esquerda)
- c) Pré-Ordem direita

árvore

Pré-Ordem: 5, 4, 3, 2, 7, 6, 8, 9



Pos-Ordem: 2, 3, 4, 6, 9, 8, 7, 5

02 Pós-Ordem

```
// if (*node == NULL) return;
```

- a) Pós-Ordem (esquerda)
- b) Pós-Ordem (direita)
- c) Imprime valor do nó corrente

(05)

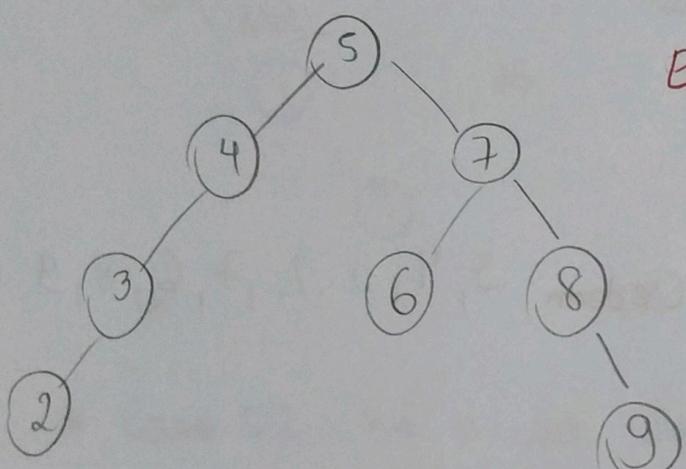
03 Em - Ordem

// if (*node == null) return;

a) Em - Ordem (esquerda)

b) Imprime nó corrente

c) Em - Ordem (direita)



Em - Ordem: 2, 3, 4, 5, 6, 7, 8, 9

Em nível: 5, 4, 7, 3, 6, 8, 2, 9

04 Em nível

a) Enfileira raiz em $Q = \emptyset$

b) Enquanto $Q \neq \emptyset$:

b.1 desenfileira elemento e imprime

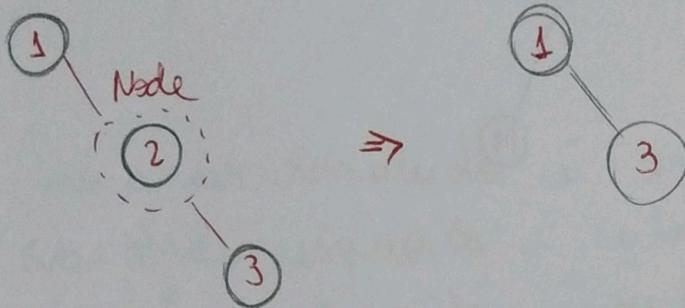
b.2 enfileira filho da esquerda

b.3 enfileira filho da direita

06

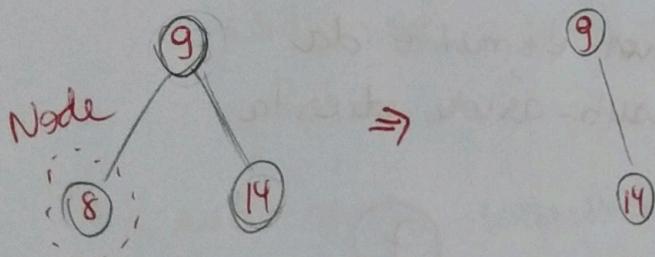
Remoção (Recursiva)

- * Caso 01: sub-árvore esquerda é nula, mas a direita contém dados



1. se $\text{node} \rightarrow \text{esq} == \text{NULL}$
2. $\text{node} = \text{node} \rightarrow \text{dir}$

- * Caso 02: nó a ser removido é uma folha



1. Se é folha:
2. $\text{node} = \text{NULL}$

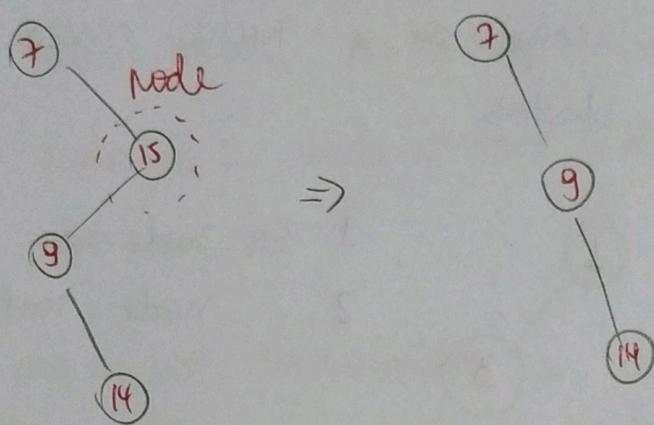
Poderia ser também:

3. $\text{node} = \text{node} \rightarrow \text{dir};$
ou
4. $\text{node} = \text{node} \rightarrow \text{esq};$

} pois tanto:
 $\text{node} \rightarrow \text{dir} = \text{node} \rightarrow \text{esq} = \text{NULL}$

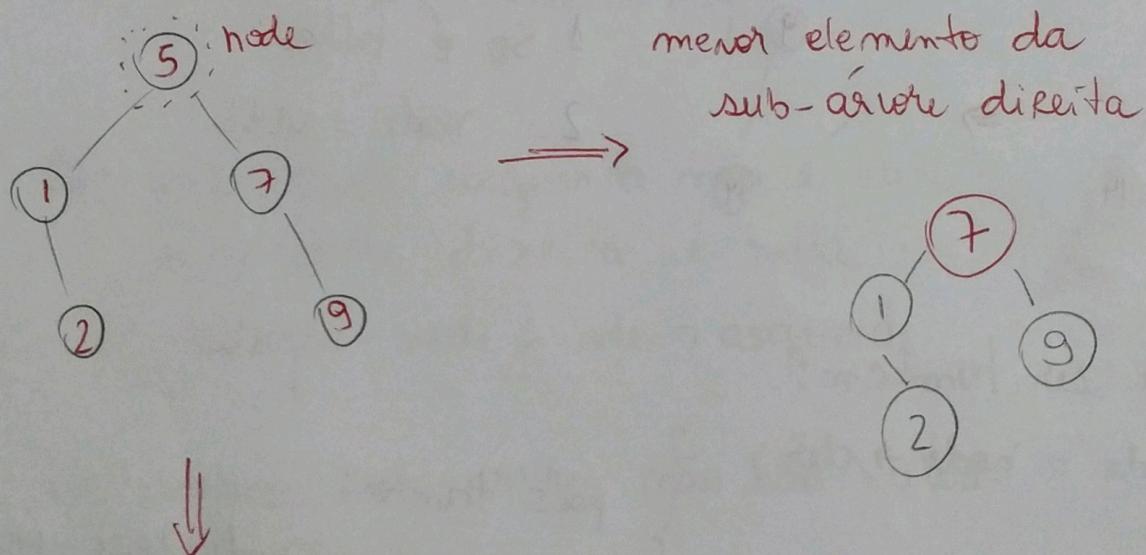
Obs: Se usarmos o comando 3, tanto o caso 01 como o 02 são resolvidos em uma só execução

* Caso 03: sub-árvore direita é nula, mas a esquerda não

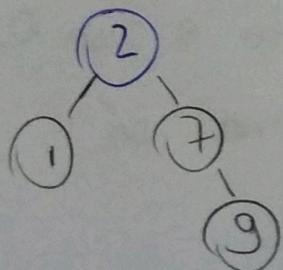


1. Se $\text{node} \rightarrow \text{dir} == \text{NULL}$
2. $\text{node} = \text{node} \rightarrow \text{esq}$

* Caso 04: Nem huma sub-árvore é nula
 $(\text{node} \rightarrow \text{dir} \neq \text{NULL} \text{ e } \text{node} \rightarrow \text{esq} \neq \text{NULL})$



maior elemento da sub-árvore esquerda (folha)



Resumo da Remoção

C1: sub-árvore esquerda é nula
sub-árvore direita != nula



Solução: node = node \rightarrow direita

C2: nó a ser removido é uma folha
(sub-árvore esquerda é nula)
(sub-árvore direita é nula)



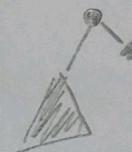
Solução:

S1: node = NULL

S2: node = node \rightarrow dir

S3: node = node \rightarrow esq

C3: sub-árvore esquerda não é nula
sub-árvore direita é nula



Solução: node = node \rightarrow esquerda

C4: ambos sub-árvores não são nulas:

Soluções:

S1: substituir pelo maior valor da sub-árvore esquerda

S2: substituir pelo menor valor da sub-árvore direita

a) encontrar ponteiro nulo, o elemento que se deseja remover não existe

b) se a chave do nó corrente é o elemento que se quer remover:

* tratar os casos c1, c2, c3, c4

c) chamadas recursivas percorrendo a árvore:

se $\text{nó.chave} > x$: chamar recursão para esquerda
se $\text{nó.chave} < x$: chamar recursão para direita