

MC-202  
Árvores Binárias

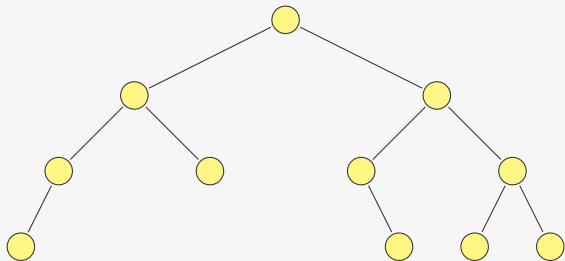
Lehilton Pedrosa  
lehilton@ic.unicamp.br

Universidade Estadual de Campinas

Segundo semestre de 2024

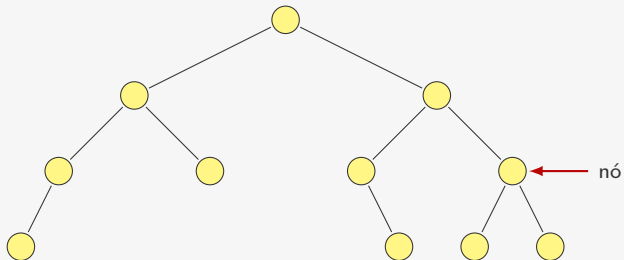
# Árvores Binárias

Exemplo de uma árvore binária:



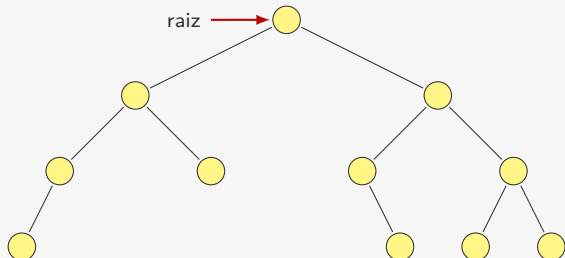
# Árvores Binárias

Exemplo de uma árvore binária:



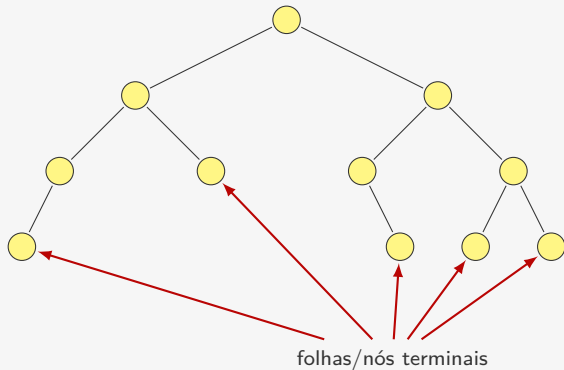
# Árvores Binárias

Exemplo de uma árvore binária:



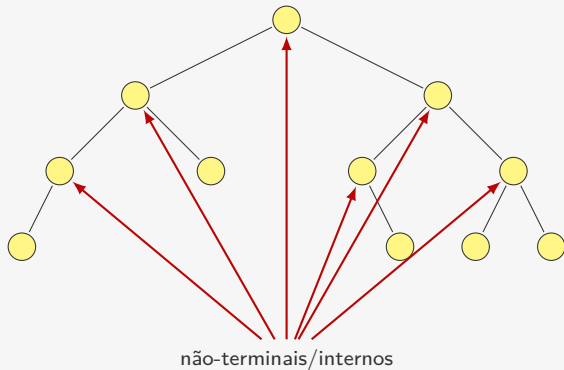
# Árvores Binárias

Exemplo de uma árvore binária:



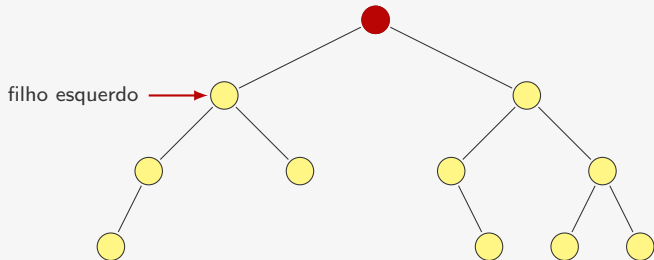
# Árvores Binárias

Exemplo de uma árvore binária:



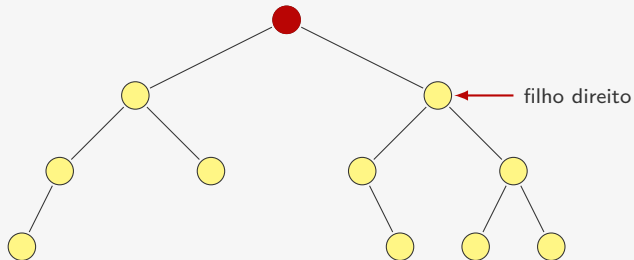
# Árvores Binárias

Exemplo de uma árvore binária:



# Árvores Binárias

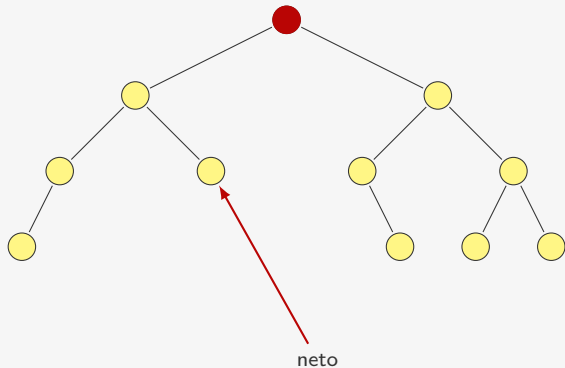
Exemplo de uma árvore binária:





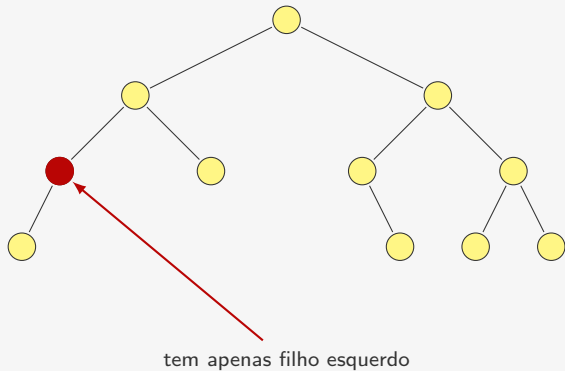
# Árvores Binárias

Exemplo de uma árvore binária:



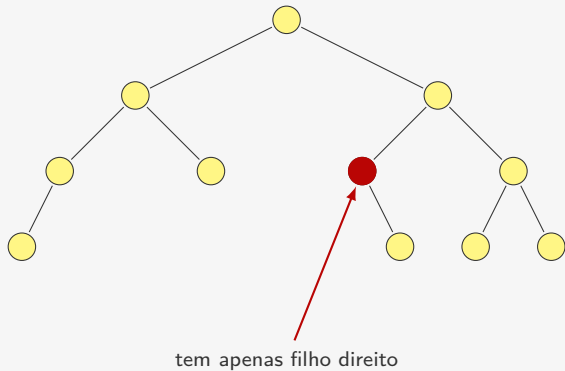
# Árvores Binárias

Exemplo de uma árvore binária:



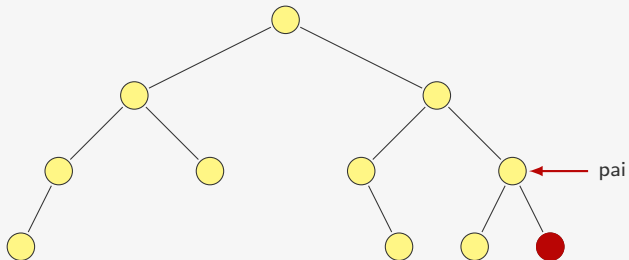
# Árvores Binárias

Exemplo de uma árvore binária:



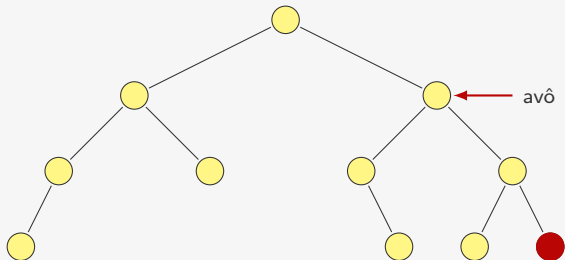
# Árvores Binárias

Exemplo de uma árvore binária:



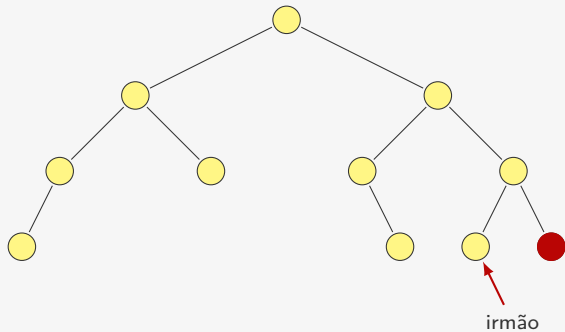
# Árvores Binárias

Exemplo de uma árvore binária:



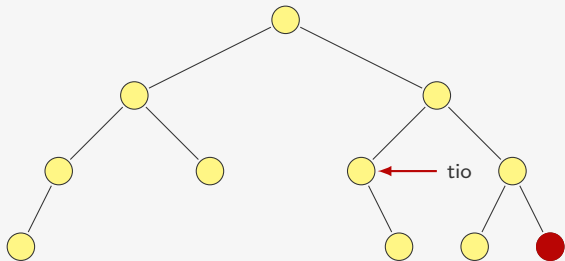
# Árvores Binárias

Exemplo de uma árvore binária:



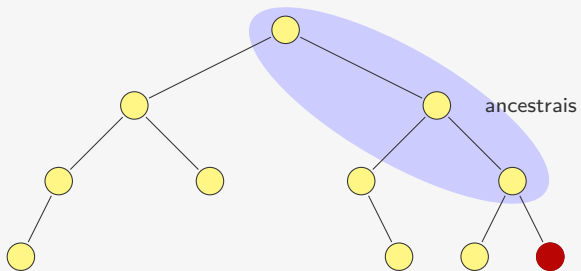
# Árvores Binárias

Exemplo de uma árvore binária:



# Árvores Binárias

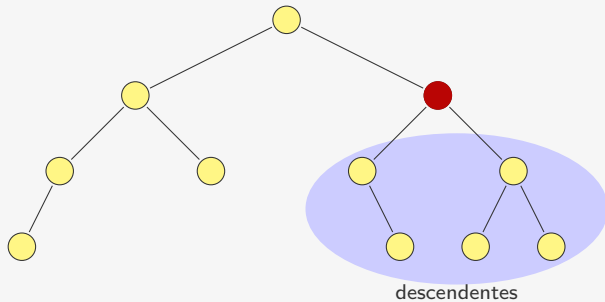
Exemplo de uma árvore binária:





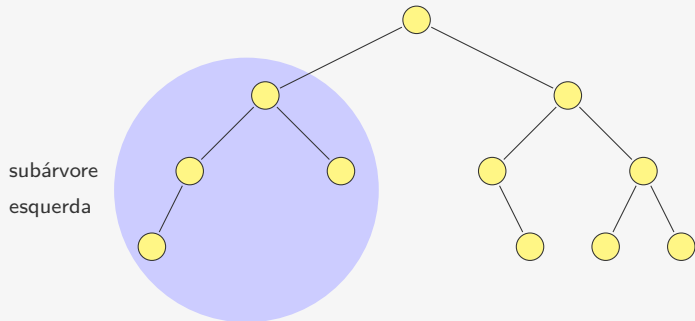
# Árvores Binárias

Exemplo de uma árvore binária:



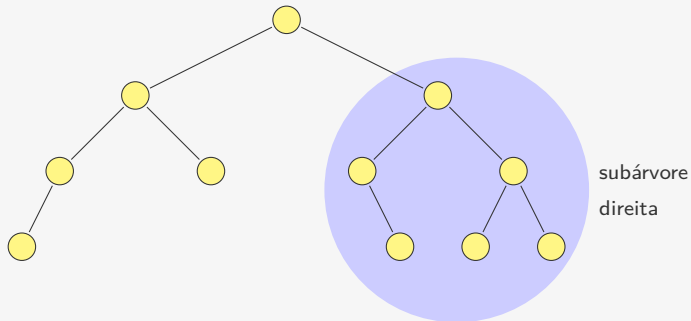
# Árvores Binárias

Exemplo de uma árvore binária:



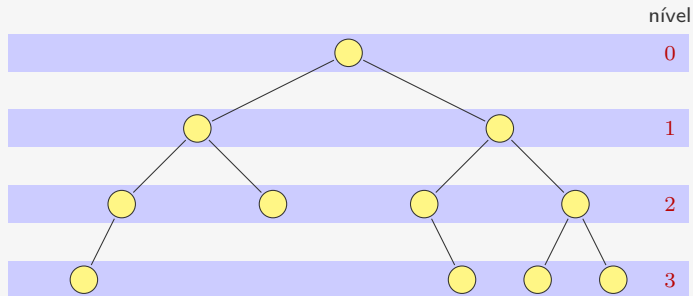
# Árvores Binárias

Exemplo de uma árvore binária:



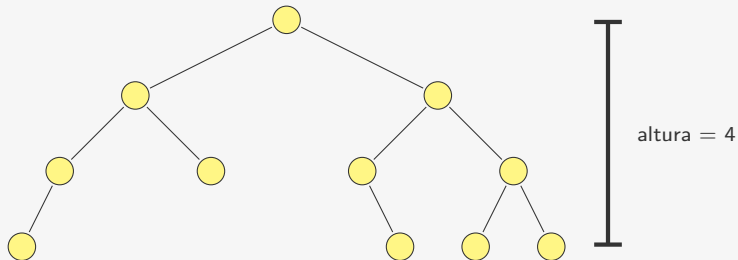
# Árvores Binárias

Exemplo de uma árvore binária:



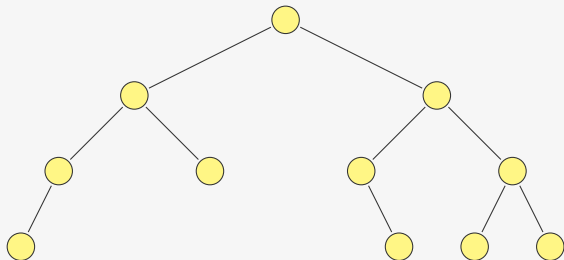
# Árvores Binárias

Exemplo de uma árvore binária:



# Árvores Binárias

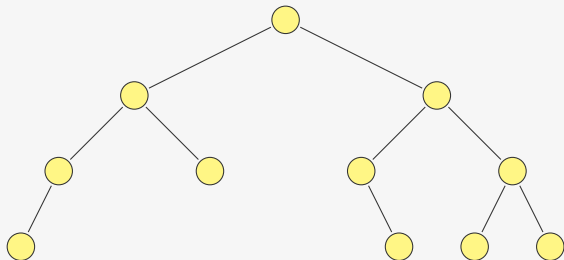
Exemplo de uma árvore binária:



Uma árvore binária é:

# Árvores Binárias

Exemplo de uma árvore binária:

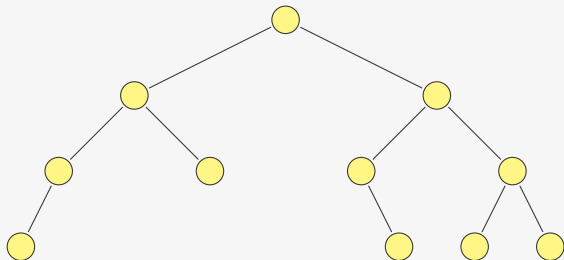


Uma árvore binária é:

- Ou o conjunto vazio

# Árvores Binárias

Exemplo de uma árvore binária:

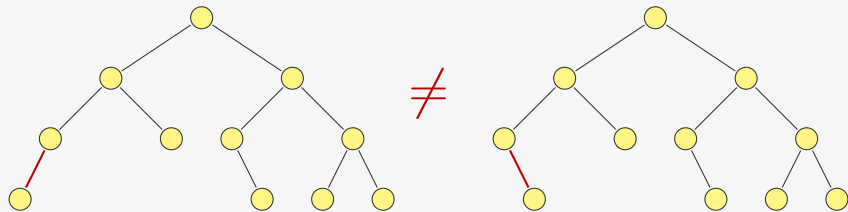


Uma árvore binária é:

- Ou o conjunto vazio
- Ou um nó conectado a duas árvores binárias



## Comparando com atenção



Ordem dos filhos é relevante!

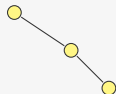
## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

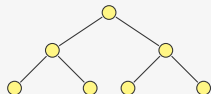
- tem no mínimo  $h$  nós



## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

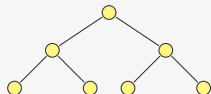
- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós



## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós

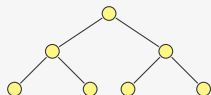


Se a árvore tem  $n \geq 1$  nós, então:

## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós



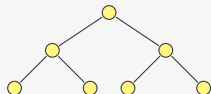
Se a árvore tem  $n \geq 1$  nós, então:

- a altura é no mínimo  $\lceil \lg(n + 1) \rceil$

## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós



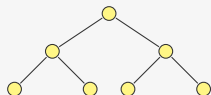
Se a árvore tem  $n \geq 1$  nós, então:

- a altura é no mínimo  $\lceil \lg(n + 1) \rceil$ 
  - quando a árvore é completa

# Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós



Se a árvore tem  $n \geq 1$  nós, então:

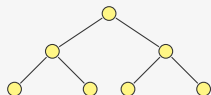
- a altura é no mínimo  $\lceil \lg(n + 1) \rceil$ 
  - quando a árvore é completa
- a altura é no máximo  $n$



## Relação entre altura e número de nós

Se a altura é  $h$ , então a árvore:

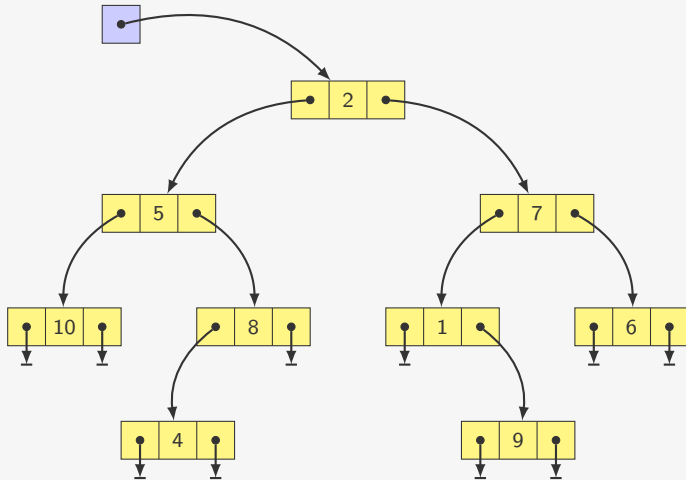
- tem no mínimo  $h$  nós
- tem no máximo  $2^h - 1$  nós



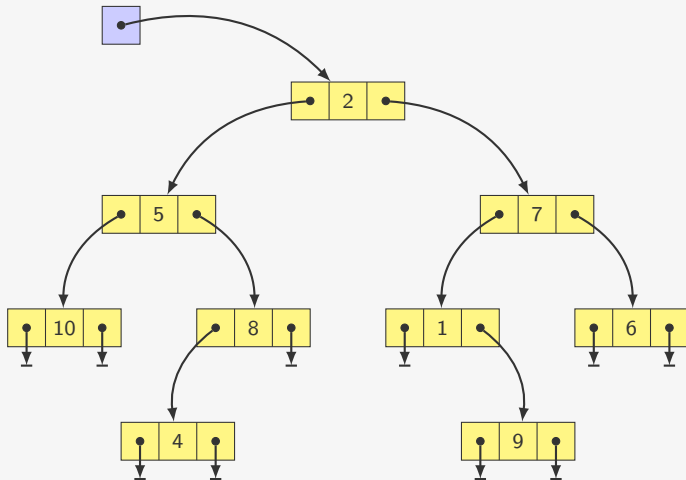
Se a árvore tem  $n \geq 1$  nós, então:

- a altura é no mínimo  $\lceil \lg(n + 1) \rceil$ 
  - quando a árvore é completa
- a altura é no máximo  $n$ 
  - quando cada nó não-terminal tem apenas um filho

# Implementação

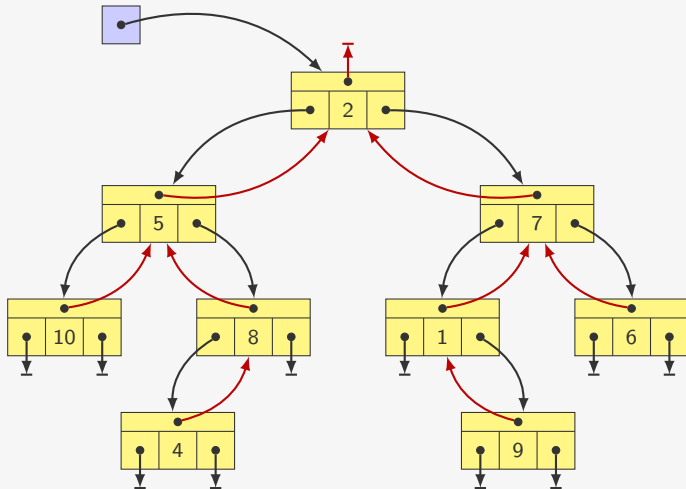


# Implementação



E se quisermos saber o pai de um nó?

# Implementação com ponteiro para pai



# Implementação em C

```
1 typedef struct no *p_no;
2
3 struct no {
4     int dado;
5     p_no esq, dir; /* pai */
6 };
7
8 p_no criar_arvore(int x, p_no esq, p_no dir);
9
10 p_no procurar_no(p_no raiz, int x);
11
12 int numero_nos(p_no raiz);
13
14 int altura(p_no raiz);
```

## Criando uma árvore e buscando

```
1 p_no criar_arvore(int x, p_no esq, p_no dir) {
2     p_no r = malloc(sizeof(struct no));
3     r->dado = x;
4     r->esq = esq;
5     r->dir = dir;
6     return r;
7 }
```

## Criando uma árvore e buscando

```
1 p_no criar_arvore(int x, p_no esq, p_no dir) {  
2     p_no r = malloc(sizeof(struct no));  
3     r->dado = x;  
4     r->esq = esq;  
5     r->dir = dir;  
6     return r;  
7 }
```

Árvores são estruturas definidas recursivamente

## Criando uma árvore e buscando

```
1 p_no criar_arvore(int x, p_no esq, p_no dir) {
2   p_no r = malloc(sizeof(struct no));
3   r->dado = x;
4   r->esq = esq;
5   r->dir = dir;
6   return r;
7 }
```

Árvores são estruturas definidas recursivamente

- basta observar a função `criar_arvore`



## Criando uma árvore e buscando

```
1 p_no criar_arvore(int x, p_no esq, p_no dir) {
2     p_no r = malloc(sizeof(struct no));
3     r->dado = x;
4     r->esq = esq;
5     r->dir = dir;
6     return r;
7 }
```

Árvores são estruturas definidas recursivamente

- basta observar a função `criar_arvore`
- faremos muitos algoritmos recursivos

## Criando uma árvore e buscando

```
1 p_no criar_arvore(int x, p_no esq, p_no dir) {
2     p_no r = malloc(sizeof(struct no));
3     r->dado = x;
4     r->esq = esq;
5     r->dir = dir;
6     return r;
7 }
```

Árvores são estruturas definidas recursivamente

- basta observar a função `criar_arvore`
- faremos muitos algoritmos recursivos

```
1 p_no procurar_no(p_no raiz, int x) {
2     p_no esq;
3     if (raiz == NULL || raiz->dado == x)
4         return raiz;
5     esq = procurar_no(raiz->esq, x);
6     if (esq != NULL)
7         return esq;
8     return procurar_no(raiz->dir, x);
9 }
```

# Número de nós e altura

Como calcular o número de nós da árvore?

# Número de nós e altura

Como calcular o número de nós da árvore?

```
1 int numero_nos(p_no raiz) {  
2     if (raiz == NULL)  
3         return 0;  
4     return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;  
5 }
```

# Número de nós e altura

Como calcular o número de nós da árvore?

```
1 int numero_nos(p_no raiz) {  
2     if (raiz == NULL)  
3         return 0;  
4     return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;  
5 }
```

Como calcular a altura da árvore?

# Número de nós e altura

Como calcular o número de nós da árvore?

```
1 int numero_nos(p_no raiz) {
2     if (raiz == NULL)
3         return 0;
4     return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;
5 }
```

Como calcular a altura da árvore?

```
1 int altura(p_no raiz) {
2     int h_esq, h_dir;
3     if (raiz == NULL)
4         return 0;
5     h_esq = altura(raiz->esq);
6     h_dir = altura(raiz->dir);
7     return 1 + (h_esq > h_dir ? h_esq : h_dir);
8 }
```

# Número de nós e altura

Como calcular o número de nós da árvore?

```
1 int numero_nos(p_no raiz) {
2     if (raiz == NULL)
3         return 0;
4     return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;
5 }
```

Como calcular a altura da árvore?

```
1 int altura(p_no raiz) {
2     int h_esq, h_dir;
3     if (raiz == NULL)
4         return 0;
5     h_esq = altura(raiz->esq);
6     h_dir = altura(raiz->dir);
7     return 1 + (h_esq > h_dir ? h_esq : h_dir);
8 }
```

**Exercício:** faça versões sem recursão dos algoritmos acima

# Número de nós e altura

Como calcular o número de nós da árvore?

```
1 int numero_nos(p_no raiz) {
2     if (raiz == NULL)
3         return 0;
4     return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;
5 }
```

Como calcular a altura da árvore?

```
1 int altura(p_no raiz) {
2     int h_esq, h_dir;
3     if (raiz == NULL)
4         return 0;
5     h_esq = altura(raiz->esq);
6     h_dir = altura(raiz->dir);
7     return 1 + (h_esq > h_dir ? h_esq : h_dir);
8 }
```

**Exercício:** faça versões sem recursão dos algoritmos acima

- você vai precisar de uma pilha...



## Exemplo: Criando um torneio

Dado um vetor  $v$  com  $n$  números, queremos criar um torneio

## Exemplo: Criando um torneio

Dado um vetor  $v$  com  $n$  números, queremos criar um torneio

- Decidir qual é o maior número em um esquema de chaves

## Exemplo: Criando um torneio

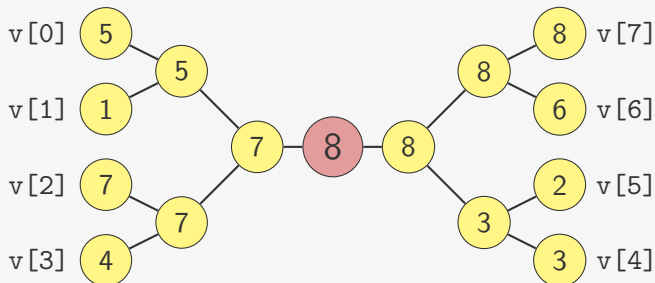
Dado um vetor  $v$  com  $n$  números, queremos criar um torneio

- Decidir qual é o maior número em um esquema de chaves
  - Ex.: para  $n = 8$ , temos quartas de final, semifinal e final

## Exemplo: Criando um torneio

Dado um vetor  $v$  com  $n$  números, queremos criar um torneio

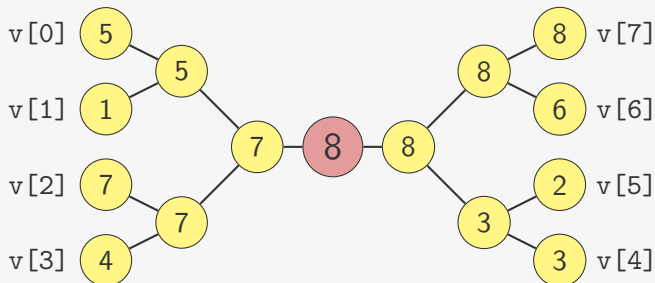
- Decidir qual é o maior número em um esquema de chaves
  - Ex.: para  $n = 8$ , temos quartas de final, semifinal e final



## Exemplo: Criando um torneio

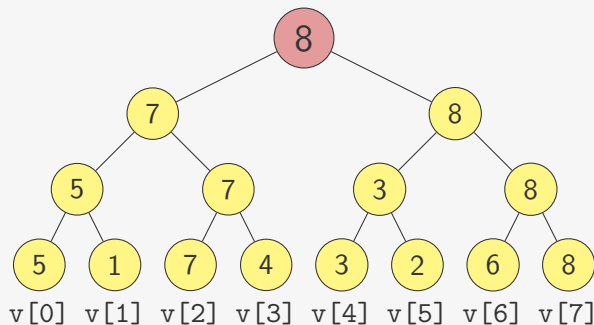
Dado um vetor  $v$  com  $n$  números, queremos criar um torneio

- Decidir qual é o maior número em um esquema de chaves
  - Ex.: para  $n = 8$ , temos quartas de final, semifinal e final

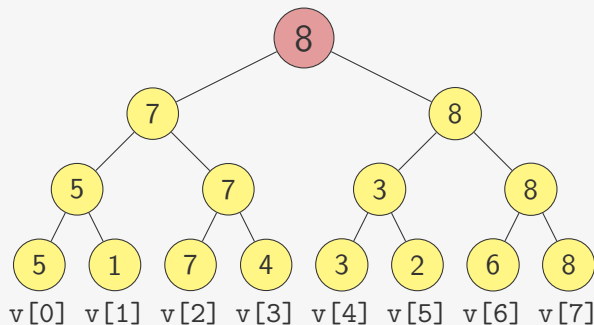


É uma **árvore binária**, onde o valor do pai é o maior valor dos seus filhos

## Exemplo: Criando um torneio

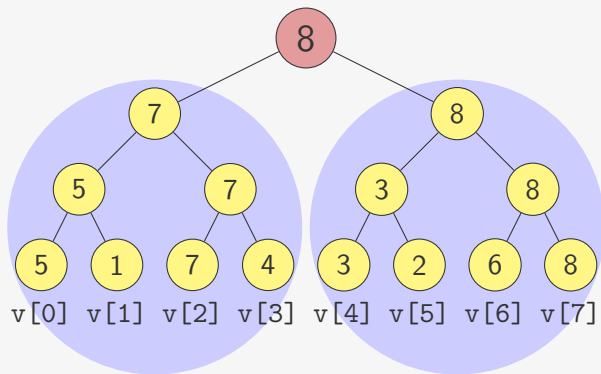


## Exemplo: Criando um torneio



Para resolver o torneio:

## Exemplo: Criando um torneio

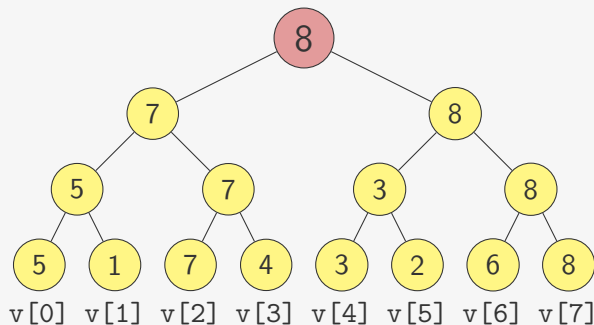


Para resolver o torneio:

- resolva o torneio das duas subárvores recursivamente



## Exemplo: Criando um torneio



Para resolver o torneio:

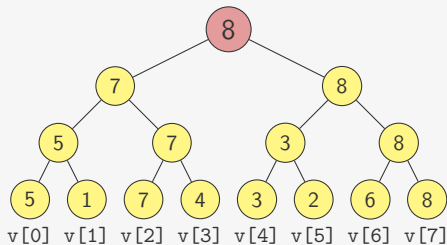
- resolva o torneio das duas subárvores recursivamente
- decida o vencedor

## Exemplo: Criando um torneio

```
1 p_no torneio(int *v, int l, int r) {
2     p_no esq, dir;
3     int valor, m = (l + r) / 2;
4     if (l == r)
5         return criar_arvore(v[l], NULL, NULL);
6     esq = torneio(v, l, m);
7     dir = torneio(v, m + 1, r);
8     valor = esq->dado > dir->dado ? esq->dado : dir->dado;
9     return criar_arvore(valor, esq, dir);
10 }
```

## Exemplo: Criando um torneio

```
1 p_no torneio(int *v, int l, int r) {
2   p_no esq, dir;
3   int valor, m = (l + r) / 2;
4   if (l == r)
5     return criar_arvore(v[l], NULL, NULL);
6   esq = torneio(v, l, m);
7   dir = torneio(v, m + 1, r);
8   valor = esq->dado > dir->dado ? esq->dado : dir->dado;
9   return criar_arvore(valor, esq, dir);
10 }
```



# Percorrendo os nós - Pré-ordem

A pré-ordem

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda

## Percorrendo os nós - Pré-ordem

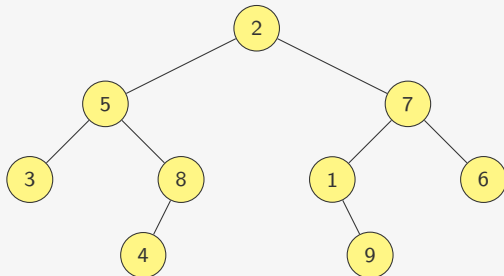
A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita



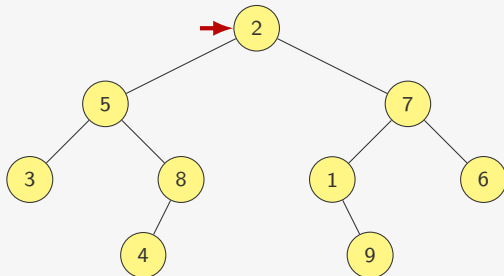
Ex:



## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

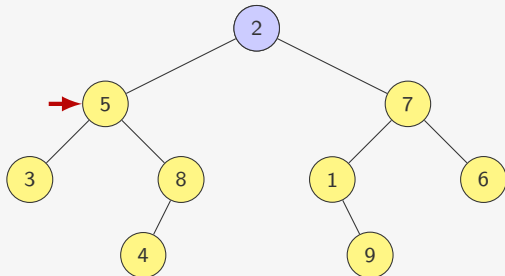


Ex:

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

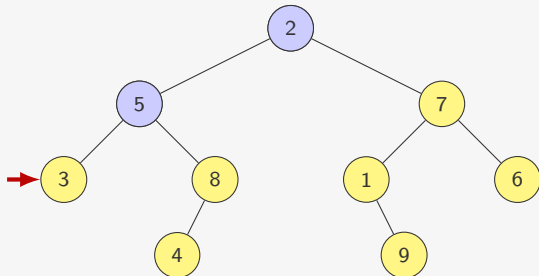


Ex: 2,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

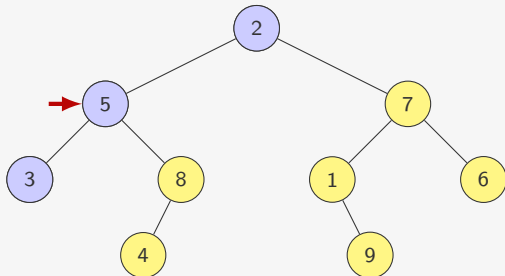


Ex: 2, 5,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

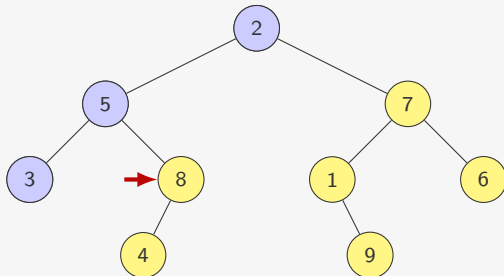


Ex: 2, 5, 3,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

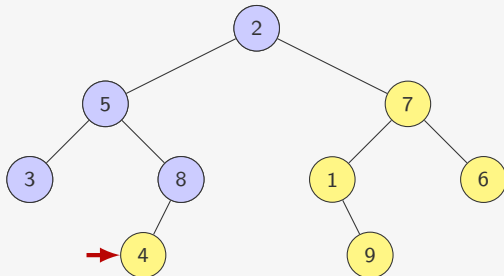


Ex: 2, 5, 3,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

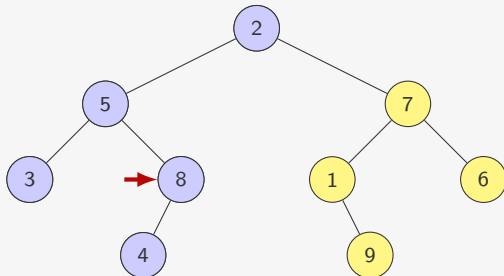


Ex: 2, 5, 3, 8,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

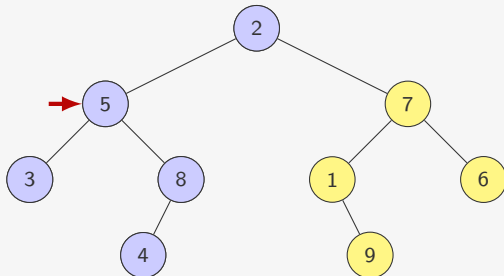


Ex: 2, 5, 3, 8, 4,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita



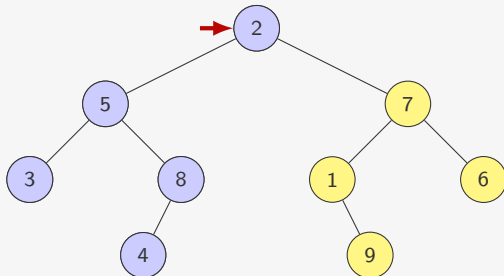
Ex: 2, 5, 3, 8, 4,



## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

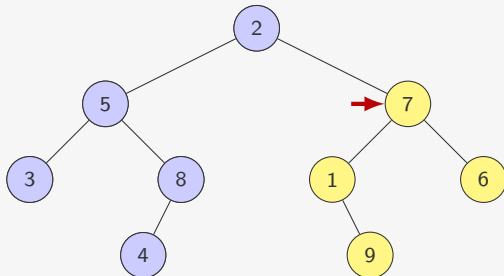


Ex: 2, 5, 3, 8, 4,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

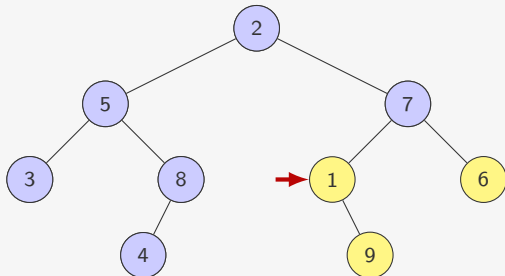


Ex: 2, 5, 3, 8, 4,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

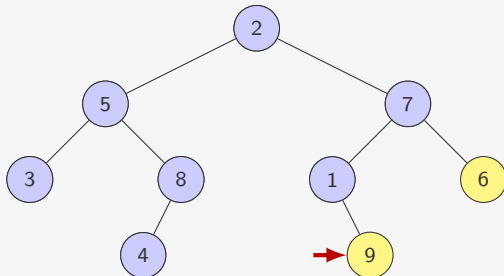


Ex: 2, 5, 3, 8, 4, 7,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

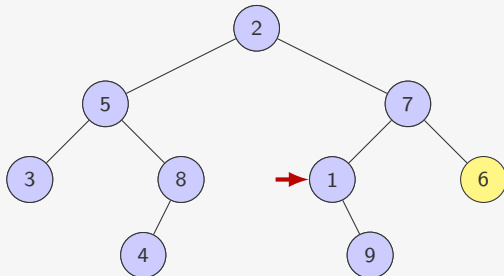


Ex: 2, 5, 3, 8, 4, 7, 1,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

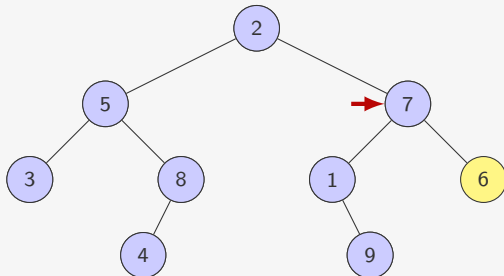


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

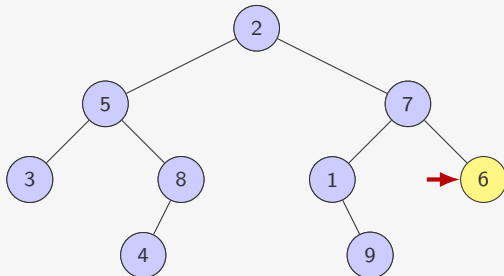


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

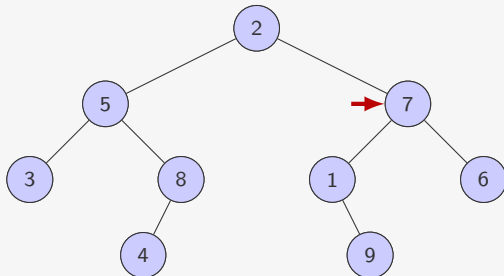


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita



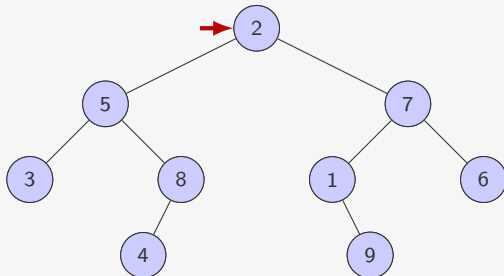
Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6



## Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita



Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

# Percorrendo os nós - Pós-ordem

A pós-ordem

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita

## Percorrendo os nós - Pós-ordem

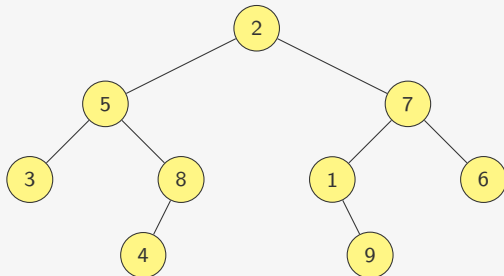
A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

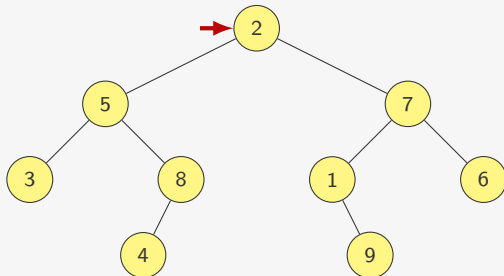


Ex:

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

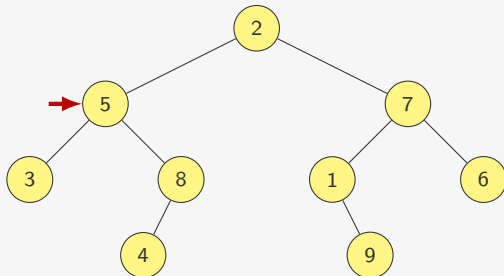


Ex:

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz



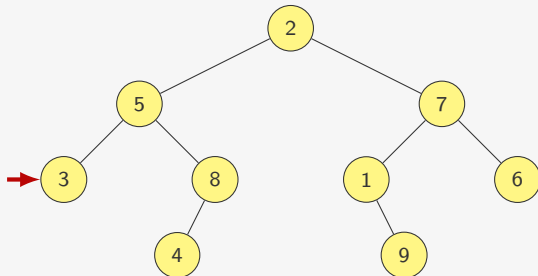
Ex:



## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

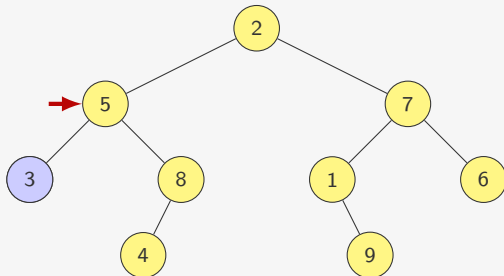


Ex:

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

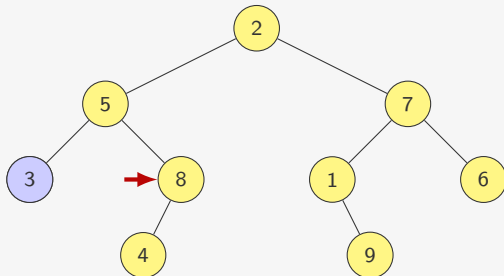


Ex: 3,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

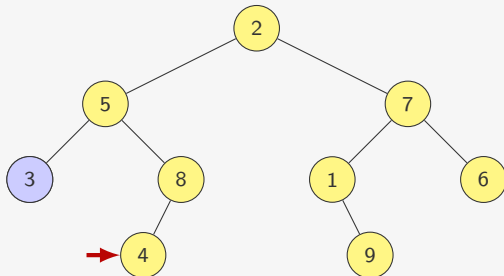


Ex: 3,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

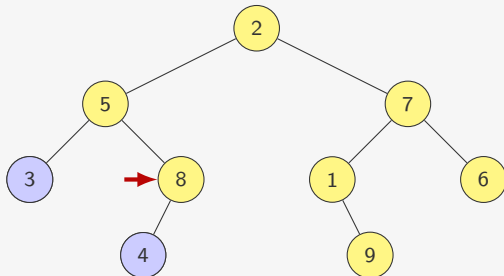


Ex: 3,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

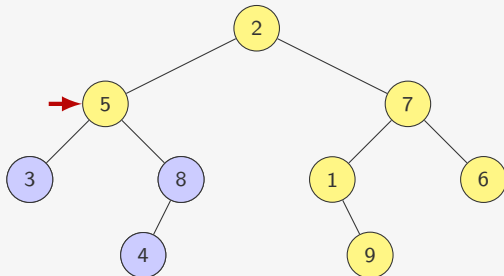


Ex: 3, 4,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

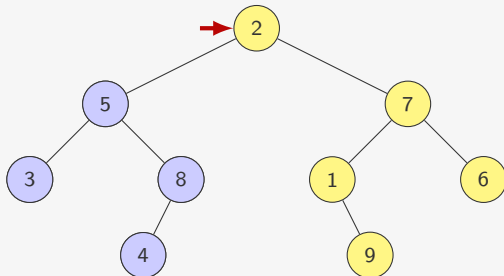


Ex: 3, 4, 8,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

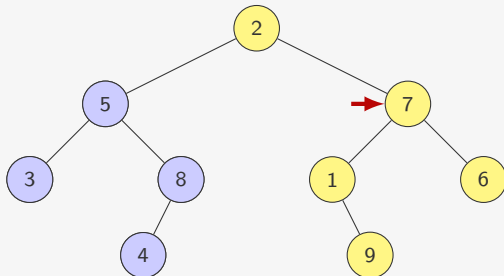


Ex: 3, 4, 8, 5,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz



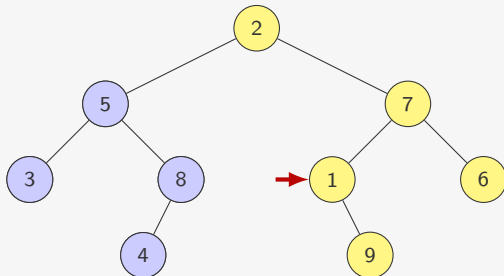
Ex: 3, 4, 8, 5,



## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

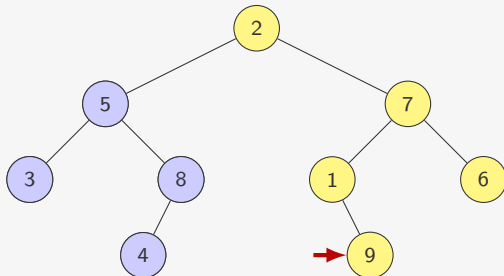


Ex: 3, 4, 8, 5,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

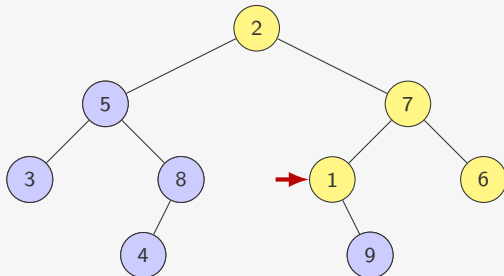


Ex: 3, 4, 8, 5,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

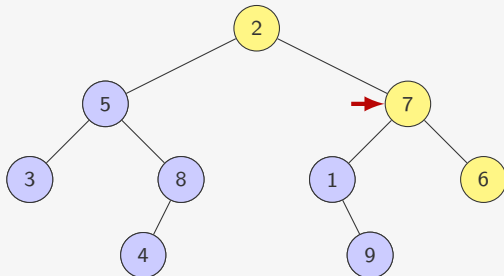


Ex: 3, 4, 8, 5, 9,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

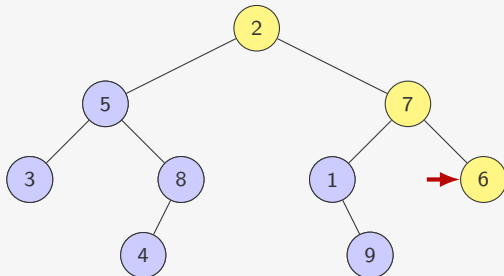


Ex: 3, 4, 8, 5, 9, 1,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

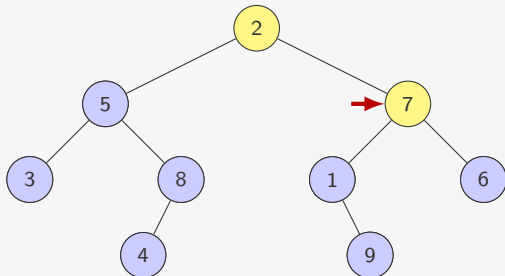


Ex: 3, 4, 8, 5, 9, 1,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

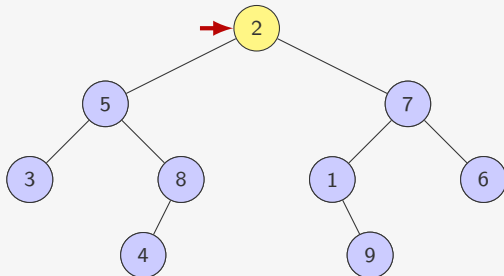


Ex: 3, 4, 8, 5, 9, 1, 6,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

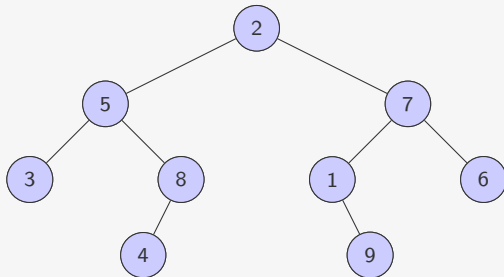


Ex: 3, 4, 8, 5, 9, 1, 6, 7,

## Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz



Ex: 3, 4, 8, 5, 9, 1, 6, 7, 2



# Percorrendo os nós - Inordem

A inordem

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz

## Percorrendo os nós - Inordem

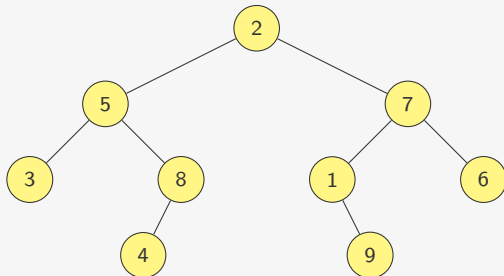
A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

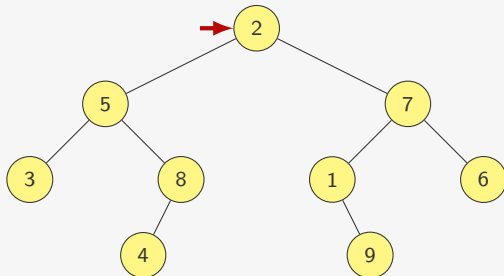


Ex:

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

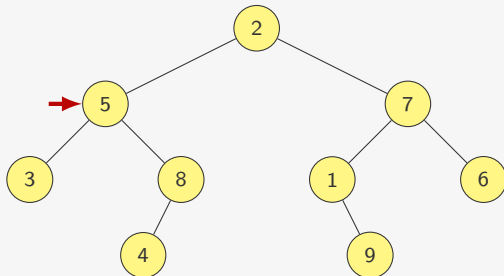


Ex:

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

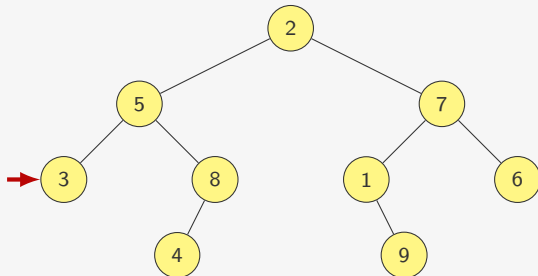


Ex:

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita



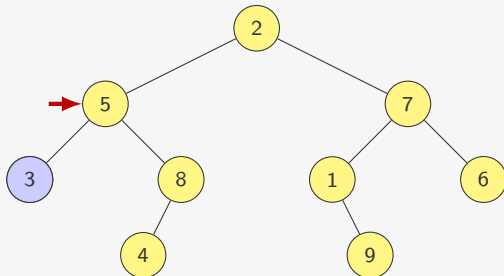
Ex:



## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

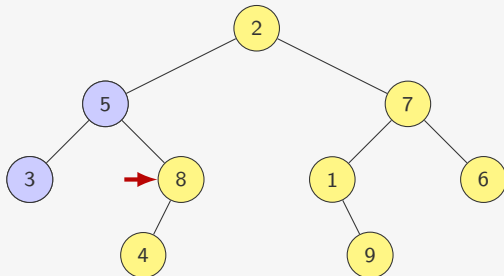


Ex: 3,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

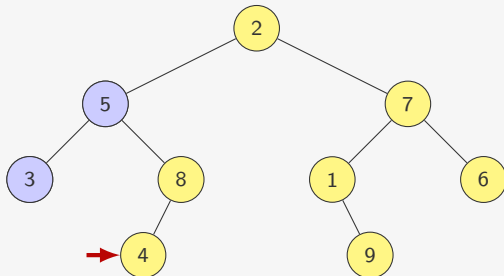


Ex: 3, 5,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

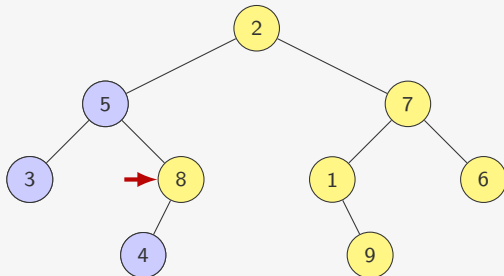


Ex: 3, 5,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

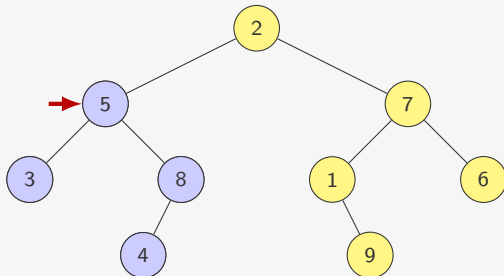


Ex: 3, 5, 4,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

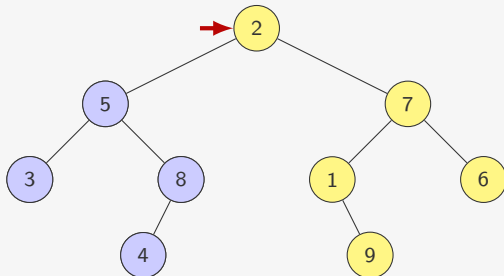


Ex: 3, 5, 4, 8,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

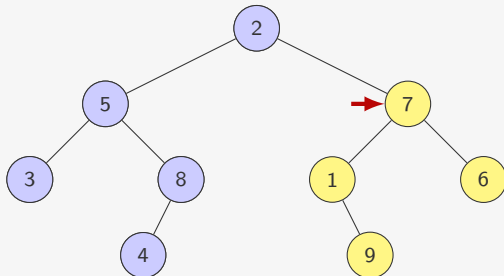


Ex: 3, 5, 4, 8,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

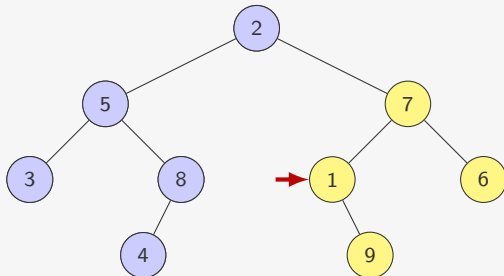


Ex: 3, 5, 4, 8, 2,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita



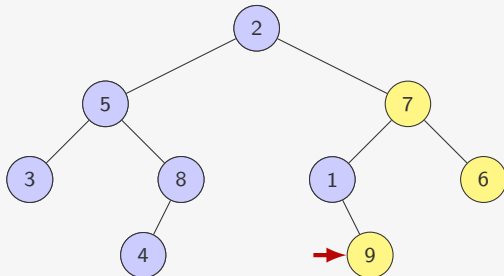
Ex: 3, 5, 4, 8, 2,



## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

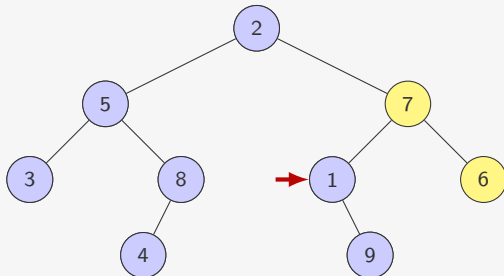


Ex: 3, 5, 4, 8, 2, 1,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

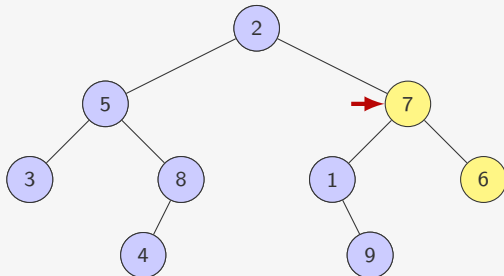


Ex: 3, 5, 4, 8, 2, 1, 9,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

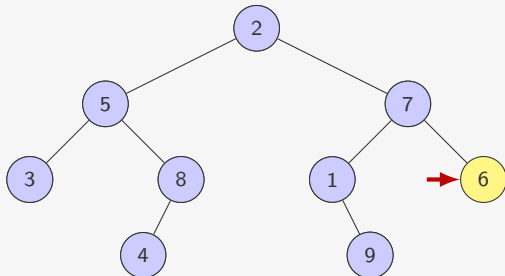


Ex: 3, 5, 4, 8, 2, 1, 9,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

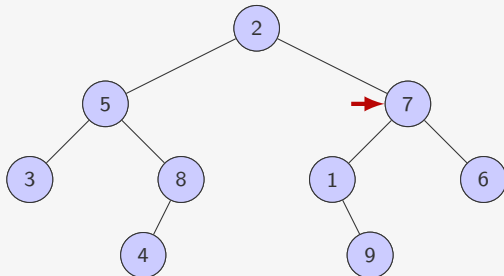


Ex: 3, 5, 4, 8, 2, 1, 9, 7,

## Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

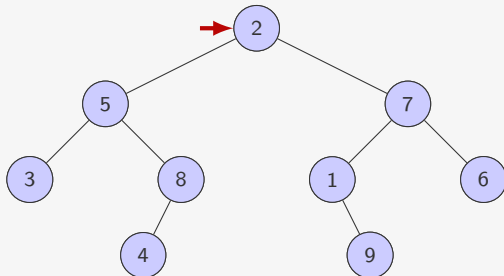


Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

## Percorrendo os nós - Inordem

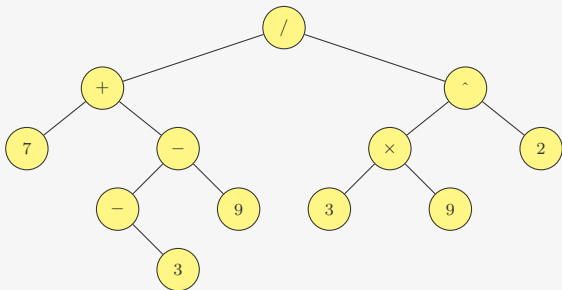
A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita



Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

## Percurso em profundidade e expressões



### Notação

- **Pré-fixa:**  $/ + 7 - - 3 9 \wedge \times 3 9 2$
- **Pós-fixa:**  $7 3 - 9 - + 3 9 \times 2 \wedge /$
- **Infixa:**  $7 + - 3 - 9 / 3 \times 9 \wedge 2$

## Implementação de percurso em profundidade

```
1 void pre_ordem(p_no raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita raiz */
4         pre_ordem(raiz->esq);
5         pre_ordem(raiz->dir);
6     }
7 }
```



# Implementação de percurso em profundidade

```
1 void pre_ordem(p_no raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita raiz */
4         pre_ordem(raiz->esq);
5         pre_ordem(raiz->dir);
6     }
7 }
```

```
1 void pos_ordem(p_no raiz) {
2     if (raiz != NULL) {
3         pos_ordem(raiz->esq);
4         pos_ordem(raiz->dir);
5         printf("%d ", raiz->dado); /* visita raiz */
6     }
7 }
```

# Implementação de percurso em profundidade

```
1 void pre_ordem(p_no raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita raiz */
4         pre_ordem(raiz->esq);
5         pre_ordem(raiz->dir);
6     }
7 }
```

```
1 void pos_ordem(p_no raiz) {
2     if (raiz != NULL) {
3         pos_ordem(raiz->esq);
4         pos_ordem(raiz->dir);
5         printf("%d ", raiz->dado); /* visita raiz */
6     }
7 }
```

```
1 void inordem(p_no raiz) {
2     if (raiz != NULL) {
3         inordem(raiz->esq);
4         printf("%d ", raiz->dado); /* visita raiz */
5         inordem(raiz->dir);
6     }
7 }
```

# Percurso em profundidade com pilha

Como implementar sem usar recursão?

# Percurso em profundidade com pilha

Como implementar sem usar recursão?

```
1 void pre_ordem(p_no raiz) {
2     p_pilha p; /* pilha de p_no */
3     p = criar_pilha();
4     empilhar(p, raiz);
5     while(!pilha_vazia(p)) {
6         raiz = desempilhar(p);
7         if (raiz != NULL) {
8             empilhar(p, raiz->dir);
9             empilhar(p, raiz->esq);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_pilha(p);
14 }
```

# Percurso em profundidade com pilha

Como implementar sem usar recursão?

```
1 void pre_ordem(p_no raiz) {
2     p_pilha p; /* pilha de p_no */
3     p = criar_pilha();
4     empilhar(p, raiz);
5     while(!pilha_vazia(p)) {
6         raiz = desempilhar(p);
7         if (raiz != NULL) {
8             empilhar(p, raiz->dir);
9             empilhar(p, raiz->esq);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_pilha(p);
14 }
```

Por que empilhamos `raiz->dir` primeiro?

# Percurso em profundidade com pilha

Como implementar sem usar recursão?

```
1 void pre_ordem(p_no raiz) {
2     p_pilha p; /* pilha de p_no */
3     p = criar_pilha();
4     empilhar(p, raiz);
5     while(!pilha_vazia(p)) {
6         raiz = desempilhar(p);
7         if (raiz != NULL) {
8             empilhar(p, raiz->dir);
9             empilhar(p, raiz->esq);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_pilha(p);
14 }
```

Por que empilhamos `raiz->dir` primeiro?

- E se fosse o contrário?

# Percorrendo os nós — em largura

O percurso em largura

# Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis



## Percorrendo os nós — em largura

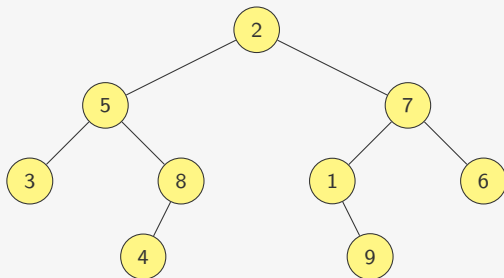
O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

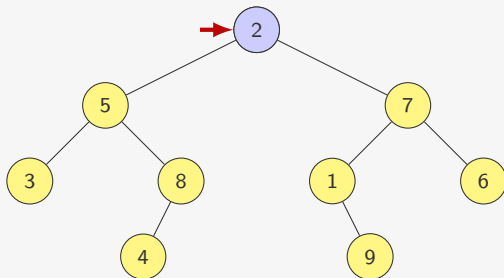


Ex:

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

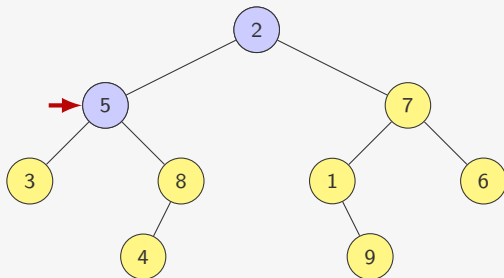


Ex: 2,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

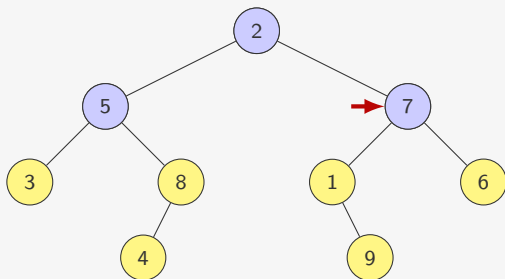


Ex: 2, 5,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

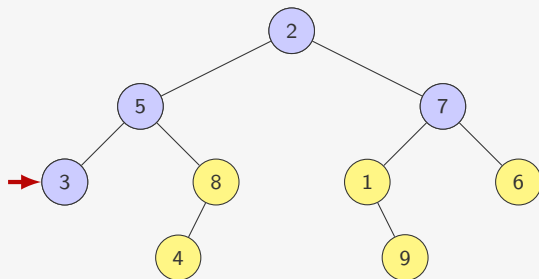


Ex: 2, 5, 7,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

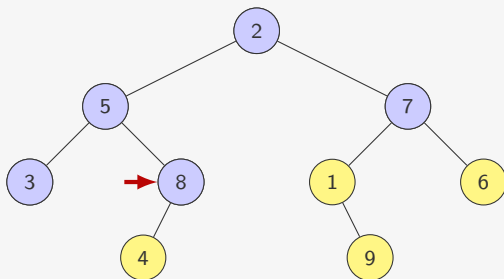


Ex: 2, 5, 7, 3,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

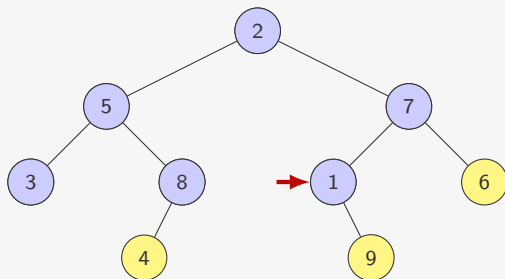


Ex: 2, 5, 7, 3, 8,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita



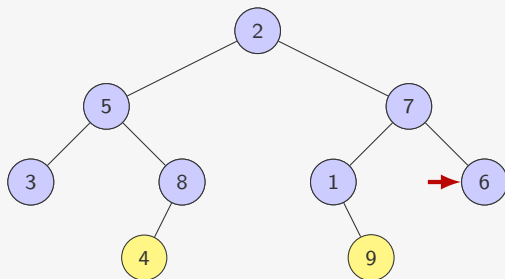
Ex: 2, 5, 7, 3, 8, 1,



## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

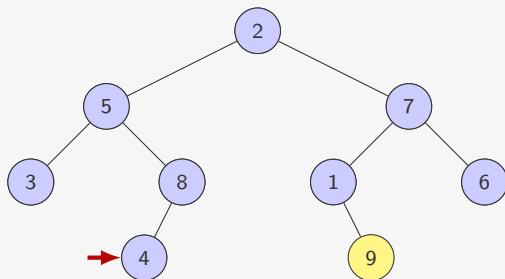


Ex: 2, 5, 7, 3, 8, 1, 6,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita

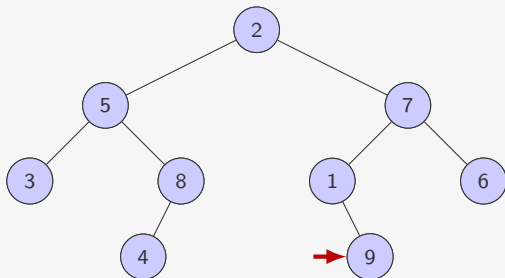


Ex: 2, 5, 7, 3, 8, 1, 6, 4,

## Percorrendo os nós — em largura

O percurso em largura

- visita os nós por níveis
- da esquerda para a direita



Ex: 2, 5, 7, 3, 8, 1, 6, 4, 9

# Implementação do percurso em largura

Como implementar o percurso em largura?

# Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila

# Implementação do percurso em largura

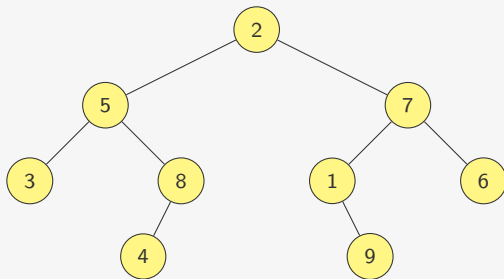
Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois

## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos

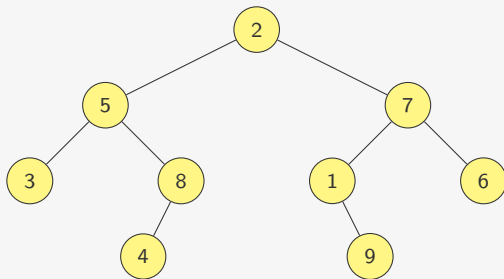


Fila

## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Fila

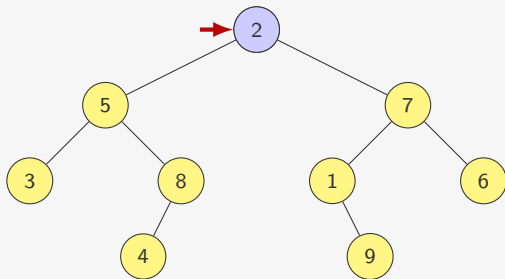




## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



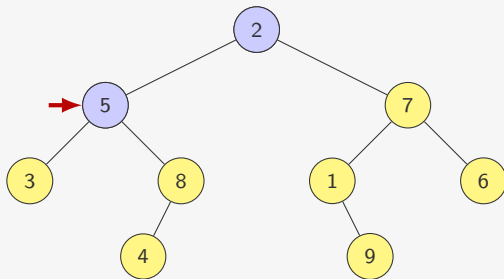
Fila 

2	5	7
---	---	---

## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



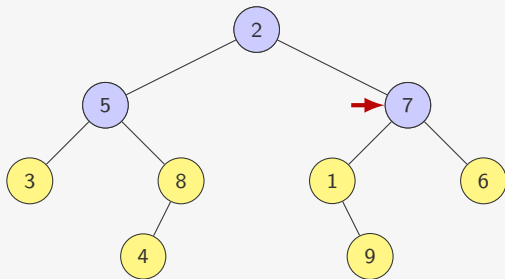
Fila



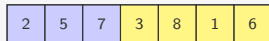
## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



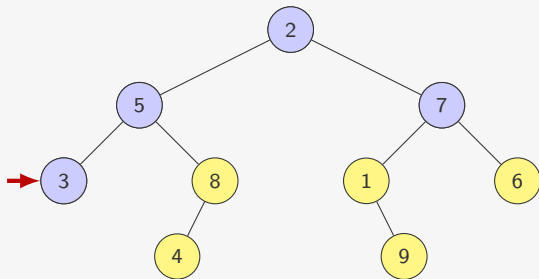
Fila



## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



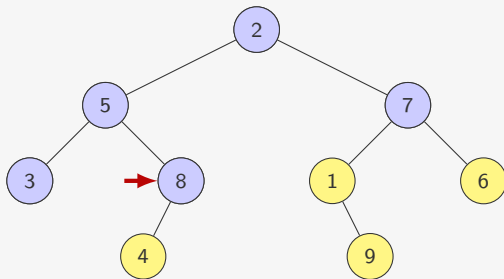
Fila



# Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



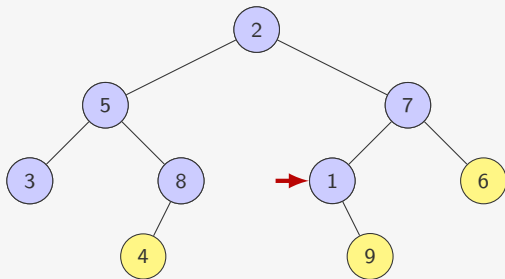
Fila



## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



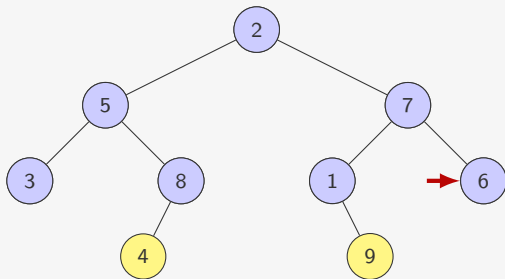
Fila

2	5	7	3	8	1	6	4	9
---	---	---	---	---	---	---	---	---

## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



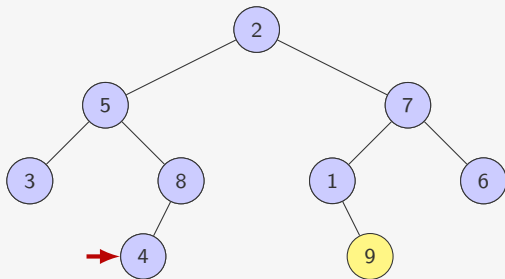
Fila



## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Fila

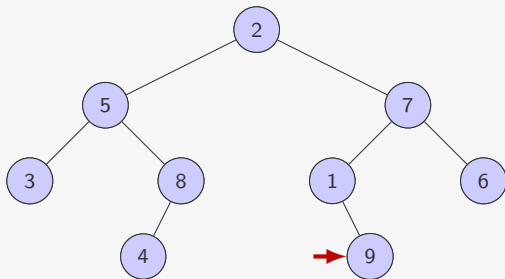
2	5	7	3	8	1	6	4	9
---	---	---	---	---	---	---	---	---



## Implementação do percurso em largura

Como implementar o percurso em largura?

- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Fila

2	5	7	3	8	1	6	4	9
---	---	---	---	---	---	---	---	---

## Percurso em largura

```
1 void percurso_em_largura(p_no raiz) {
2     p_fila f;
3     f = criar_fila();
4     enfileirar(f, raiz);
5     while(!fila_vazia(f)) {
6         raiz = desenfileirar(f);
7         if (raiz != NULL) {
8             enfileirar(f, raiz->esq);
9             enfileirar(f, raiz->dir);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_fila(f);
14 }
```

## Percurso em largura

```
1 void percurso_em_largura(p_no raiz) {
2     p_fila f;
3     f = criar_fila();
4     enfileirar(f, raiz);
5     while(!fila_vazia(f)) {
6         raiz = desenfileirar(f);
7         if (raiz != NULL) {
8             enfileirar(f, raiz->esq);
9             enfileirar(f, raiz->dir);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_fila(f);
14 }
```

Agora enfileiramos `raiz->esq` primeiro

## Percurso em largura

```
1 void percurso_em_largura(p_no raiz) {
2     p_fila f;
3     f = criar_fila();
4     enfileirar(f, raiz);
5     while(!fila_vazia(f)) {
6         raiz = desenfileirar(f);
7         if (raiz != NULL) {
8             enfileirar(f, raiz->esq);
9             enfileirar(f, raiz->dir);
10            printf("%d ", raiz->dado); /* visita raiz */
11        }
12    }
13    destruir_fila(f);
14 }
```

Agora enfileiramos `raiz->esq` primeiro

- E se fosse o contrário?

## Exercício

Escreva uma função que calcula o número de folhas em uma árvore dada.

# Solução

```
1 int folhas(p_no raiz) {
2     if (raiz == NULL)
3         return 0;
4     if (raiz->esq == NULL && raiz->dir == NULL)
5         return 1;
6     return folhas(raiz->esq) + folhas(raiz->dir);
7 }
```

## Exercício

Escreva uma função recursiva que apaga todas as folhas de uma árvore que tenham a chave igual a um valor dado.

# Solução

```
1 p_no apaga_folhas(p_no raiz, int x) {
2     if (raiz == NULL)
3         return NULL;
4     if (raiz->esq == NULL && raiz->dir == NULL && raiz->dado == x) {
5         free(raiz);
6         return NULL;
7     }
8     raiz->esq = apaga_folhas(raiz->esq, x);
9     raiz->dir = apaga_folhas(raiz->dir, x);
10    return raiz;
11 }
```



## Exercício

Escreva uma função que compara se duas árvores binárias são iguais.

# Solução

```
1 int iguais(p_no arvore1, p_no arvore2) {
2     if (arvore1 == NULL && arvore2 == NULL)
3         return 1;
4     if (arvore1 == NULL || arvore2 == NULL)
5         return 0;
6     return arvore1->dado == arvore2->dado &&
7         iguais(arvore1->esq, arvore2->esq) &&
8         iguais(arvore1->dir, arvore2->dir);
9 }
```

Dúvidas?