Estruturas de Dados

Árvores

Aula 06

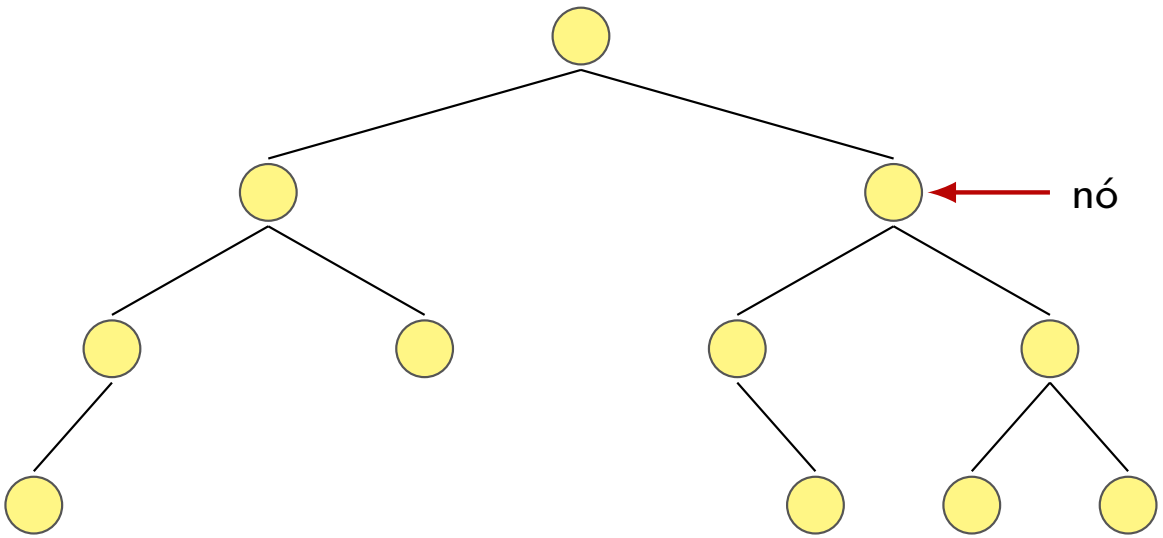
Prof. Felipe A. Louza

Roteiro

Roteiro

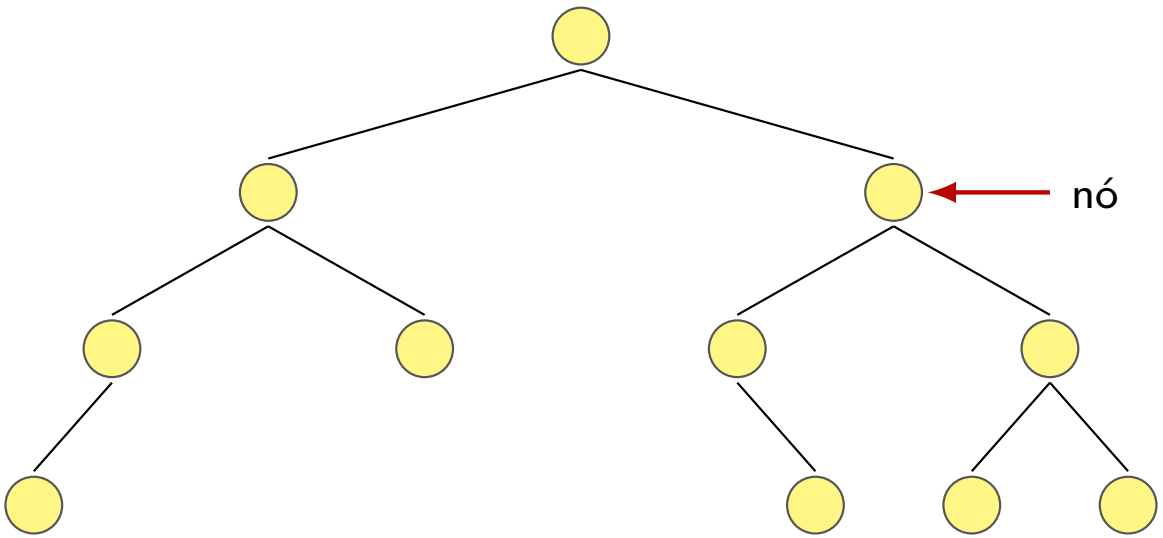
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.

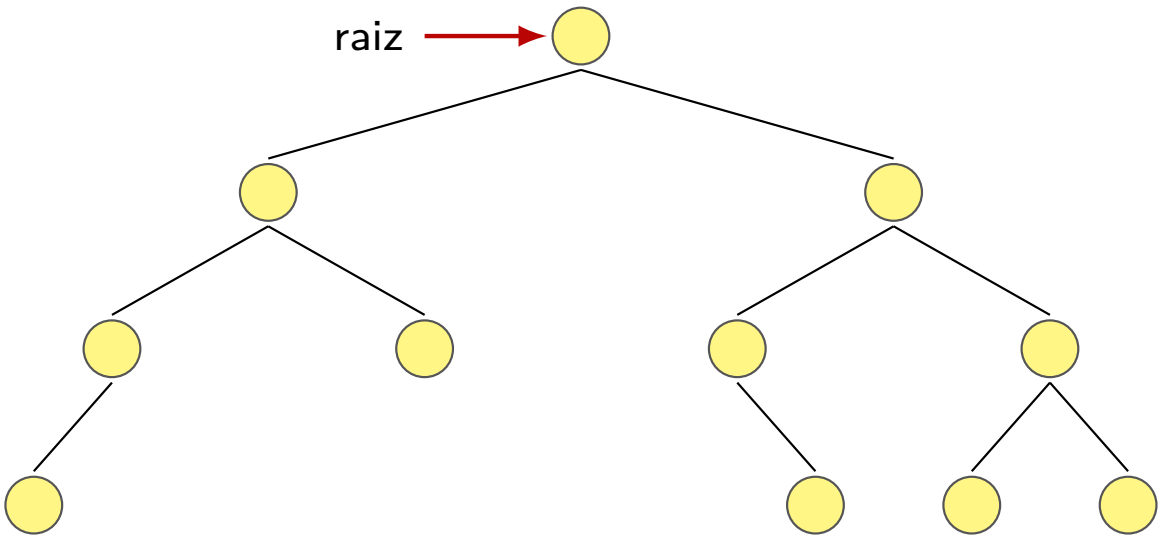


Uma árvore binária é:

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

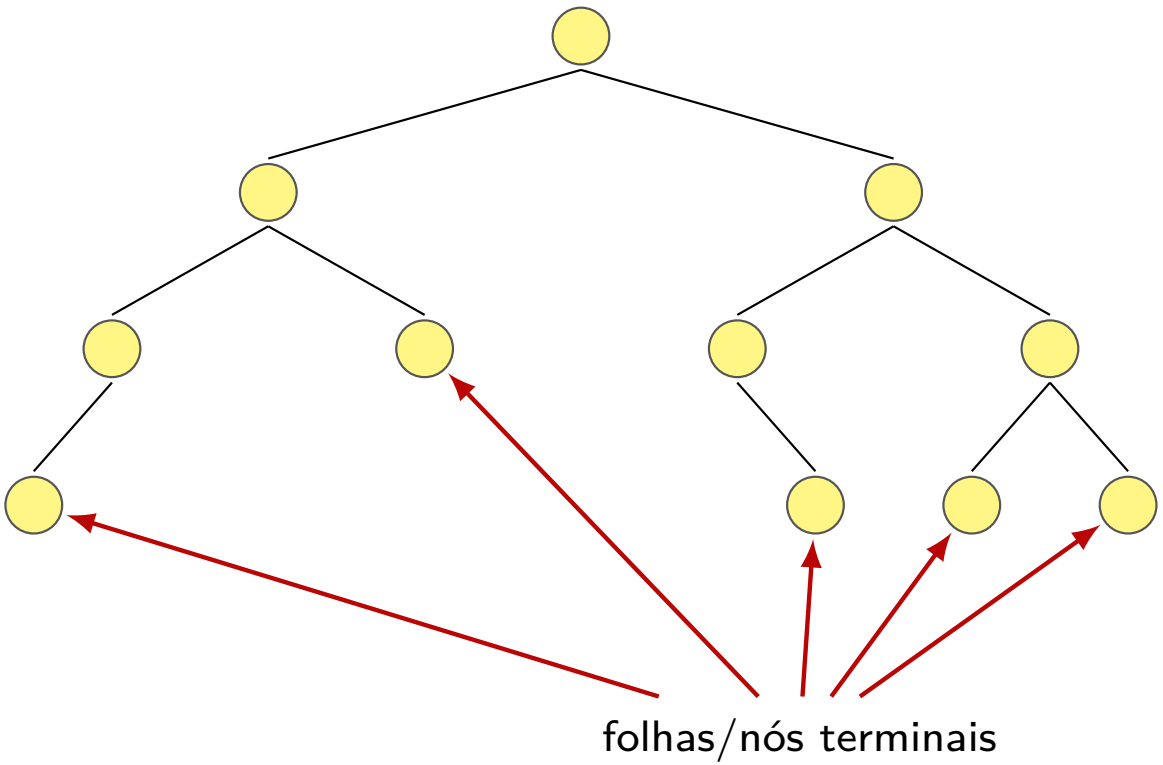
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



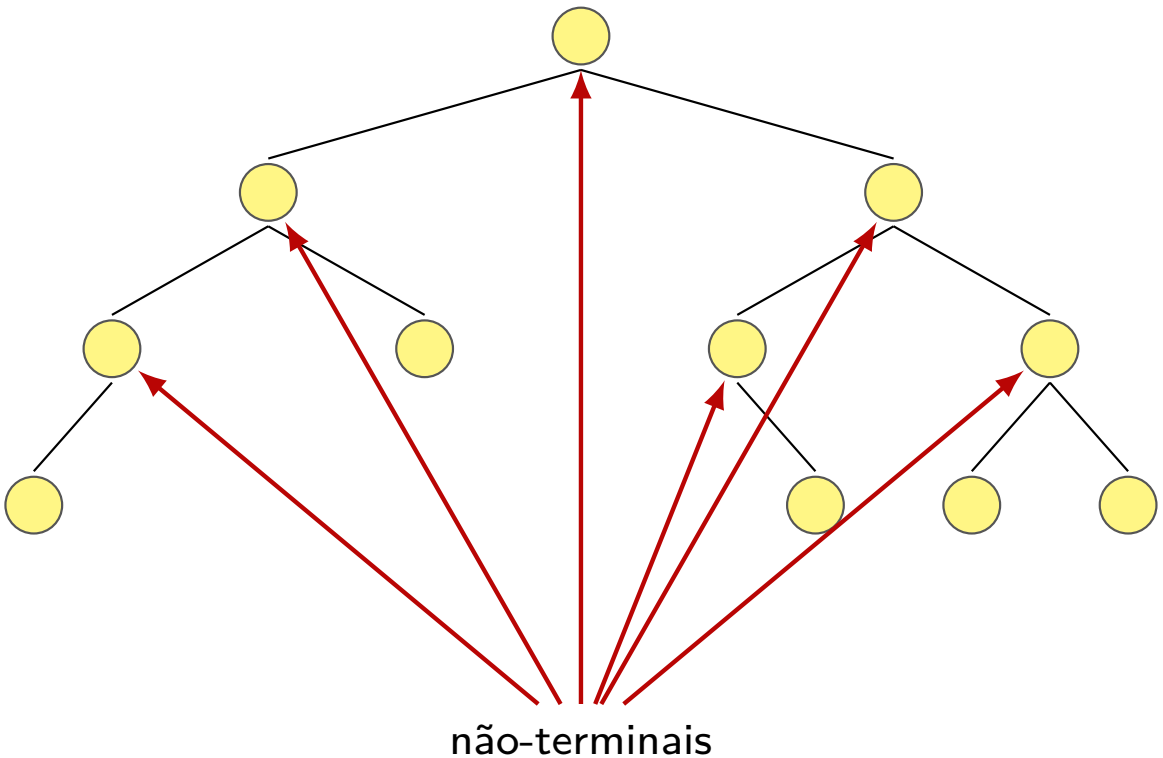
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



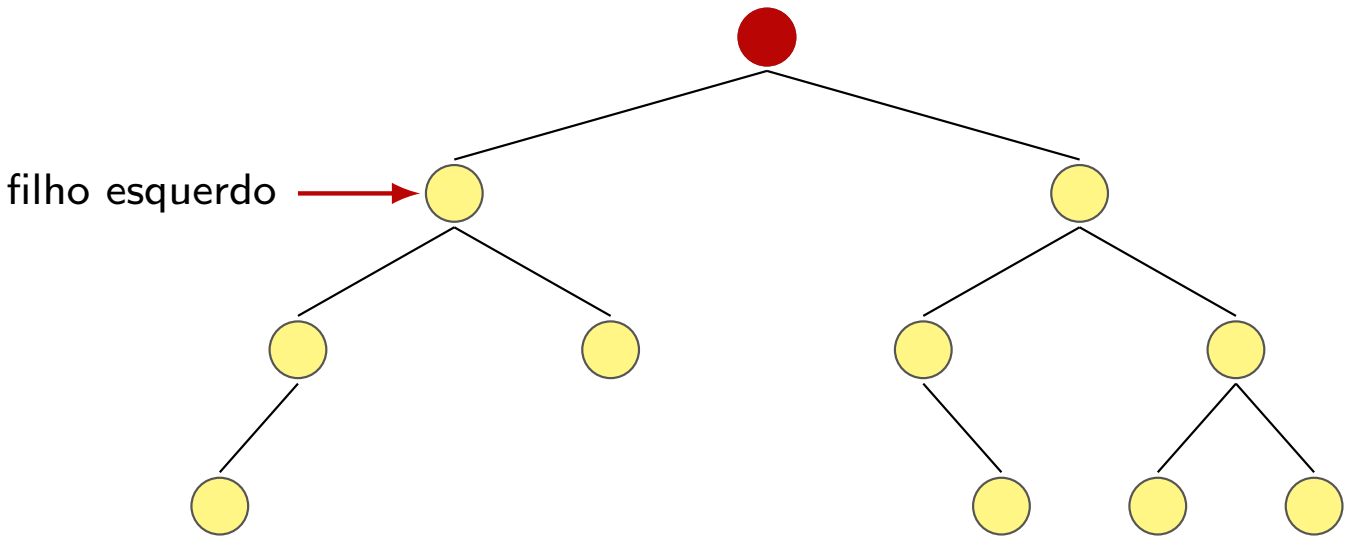
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



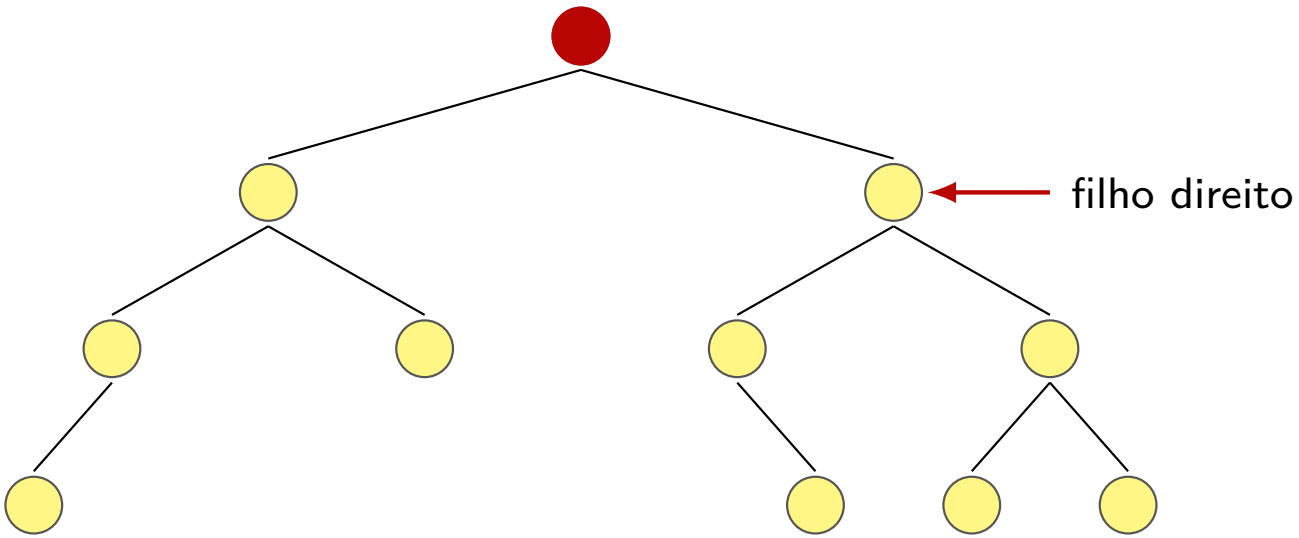
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



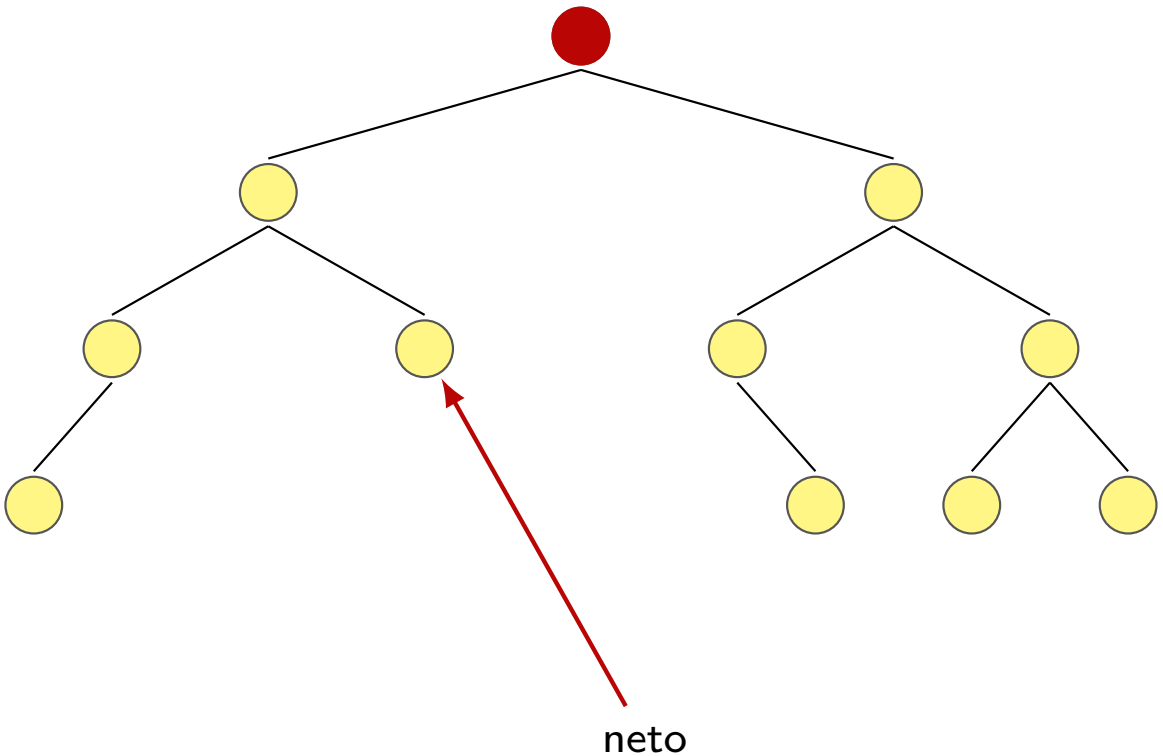
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



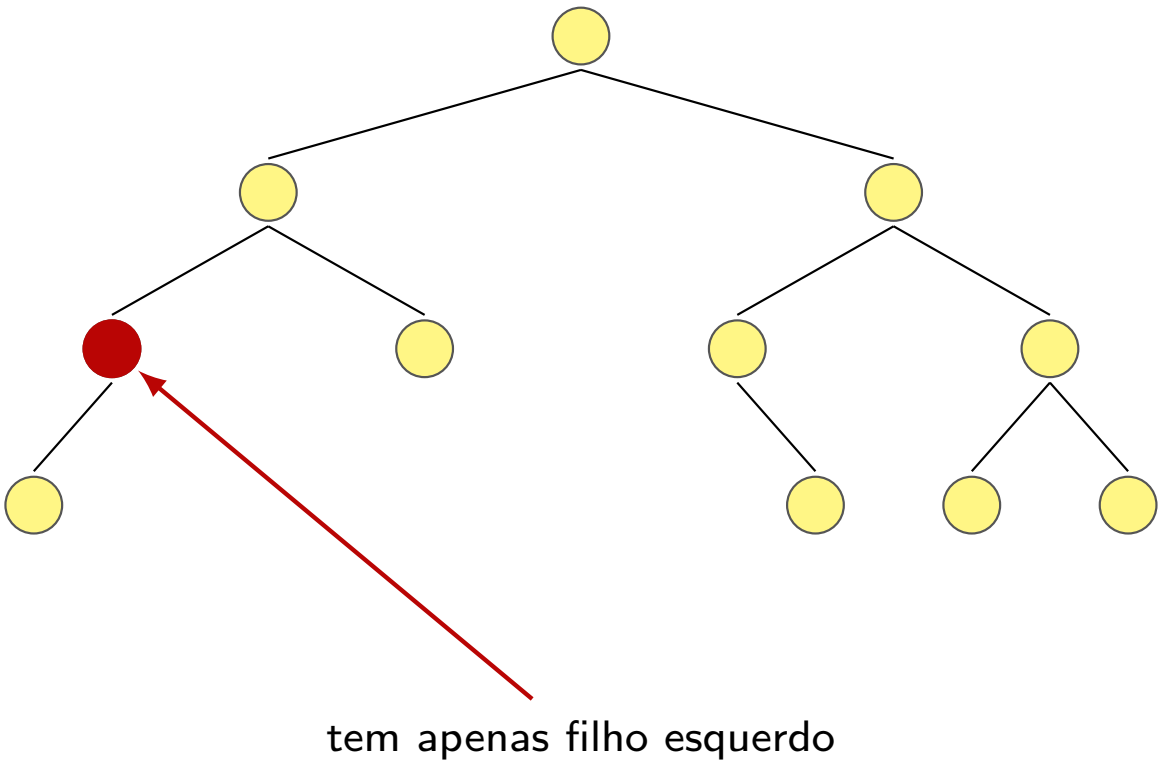
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



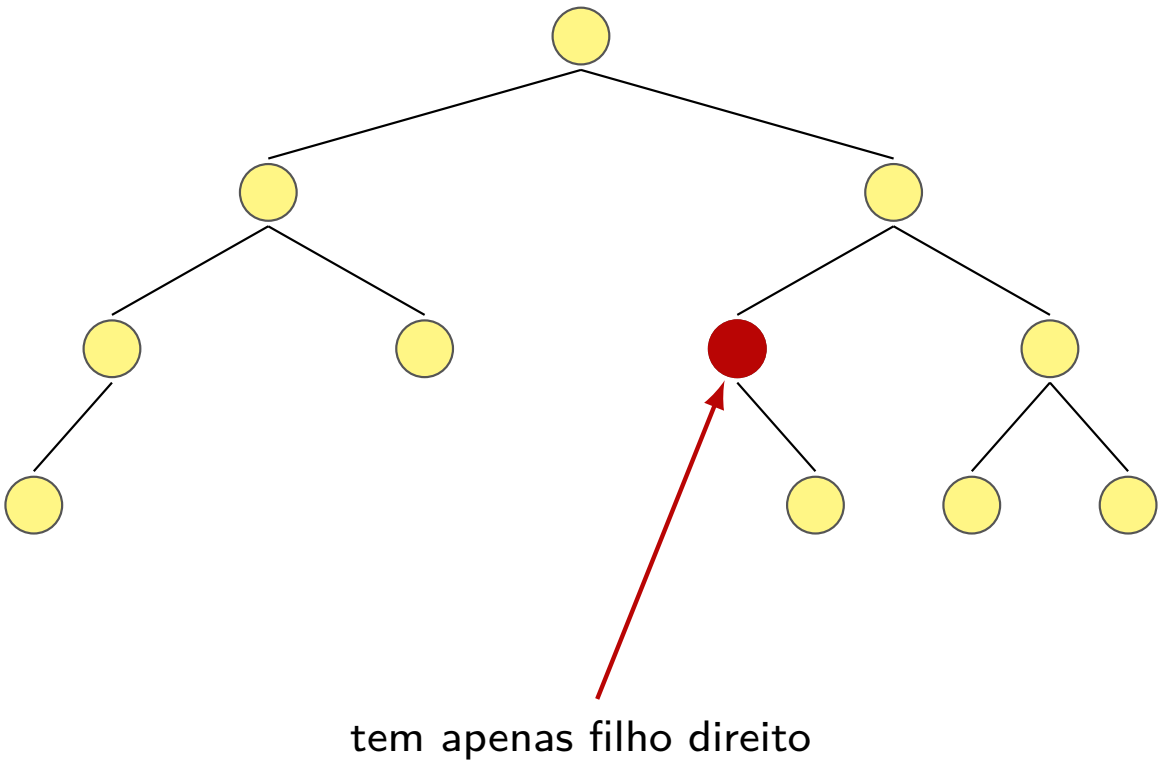
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



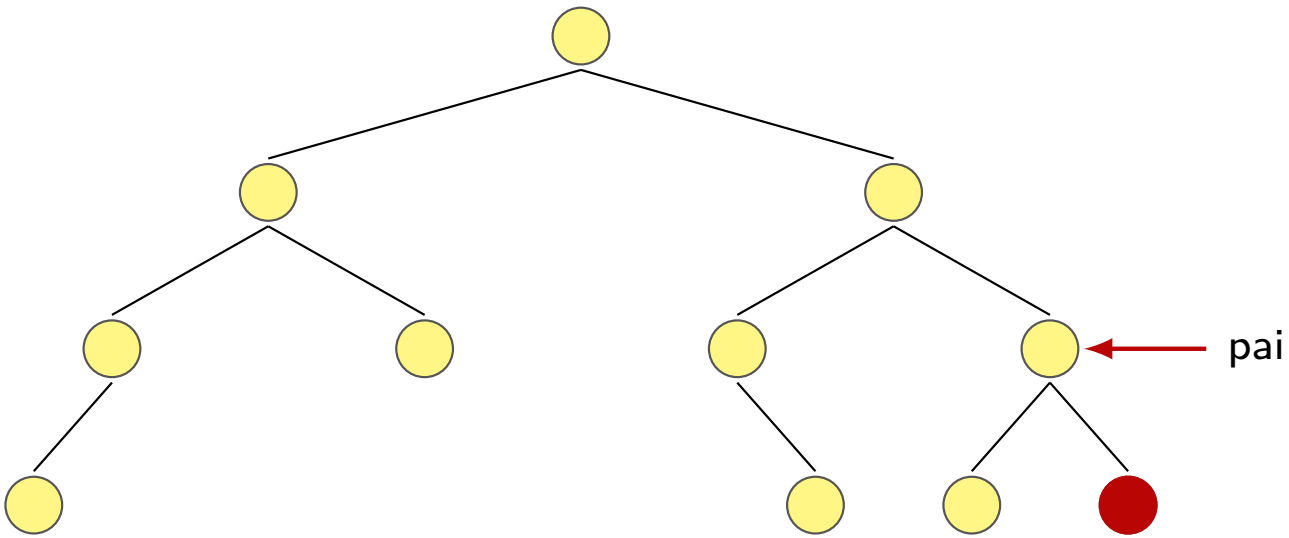
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



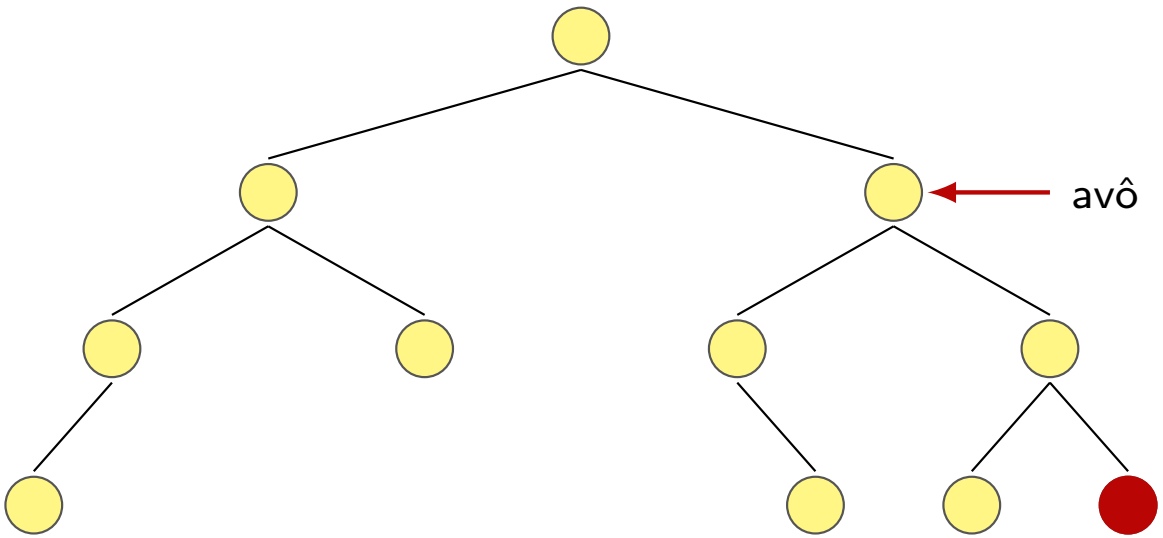
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



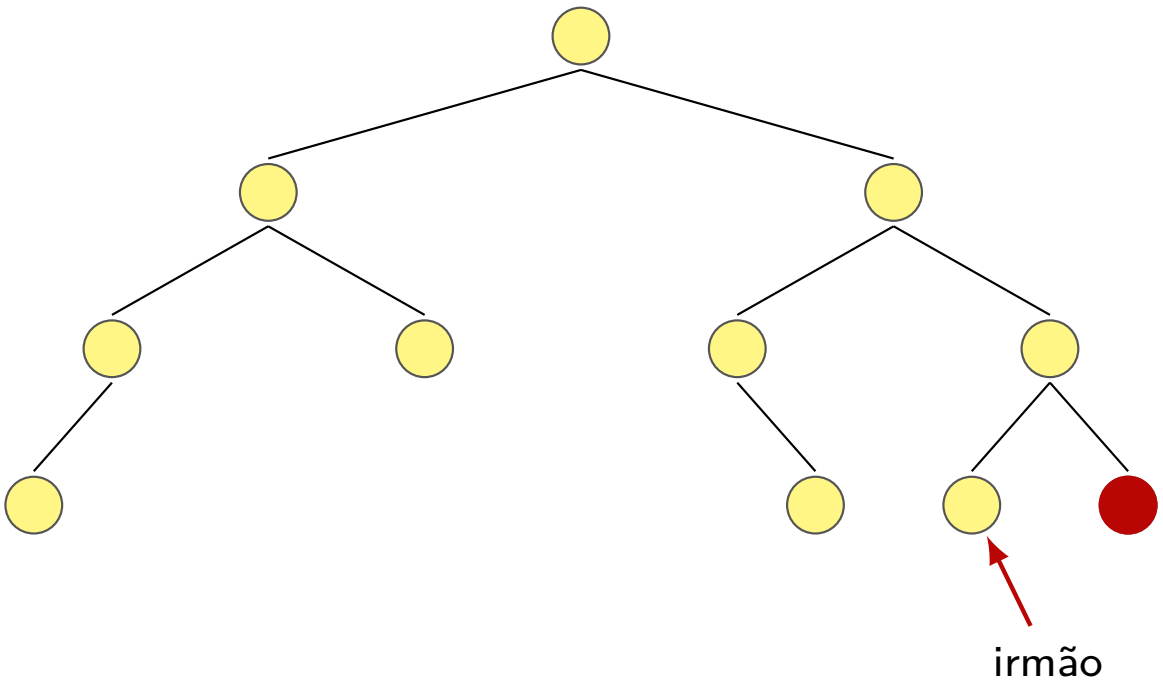
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



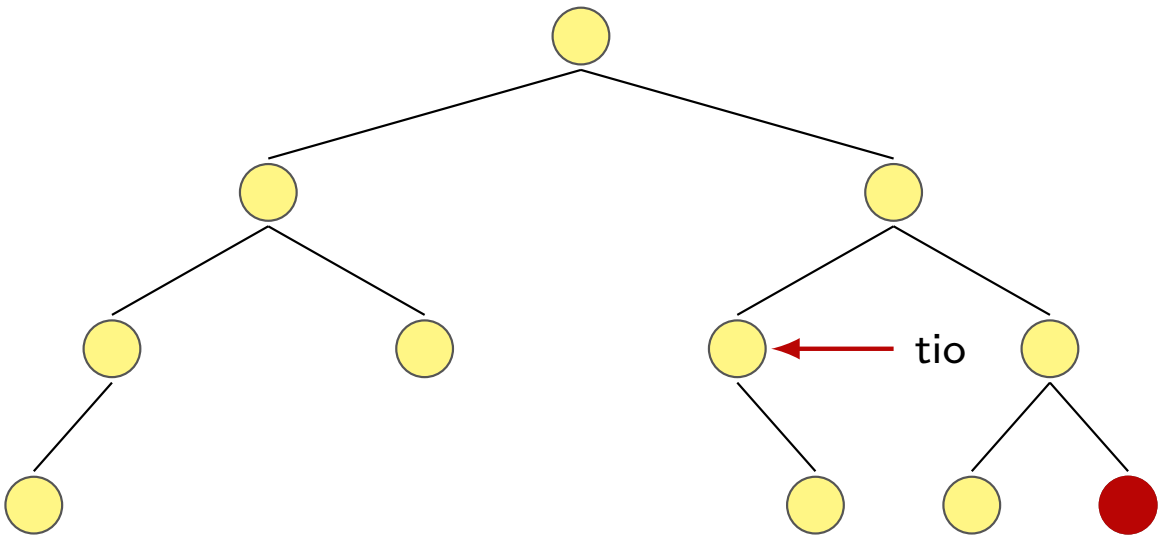
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



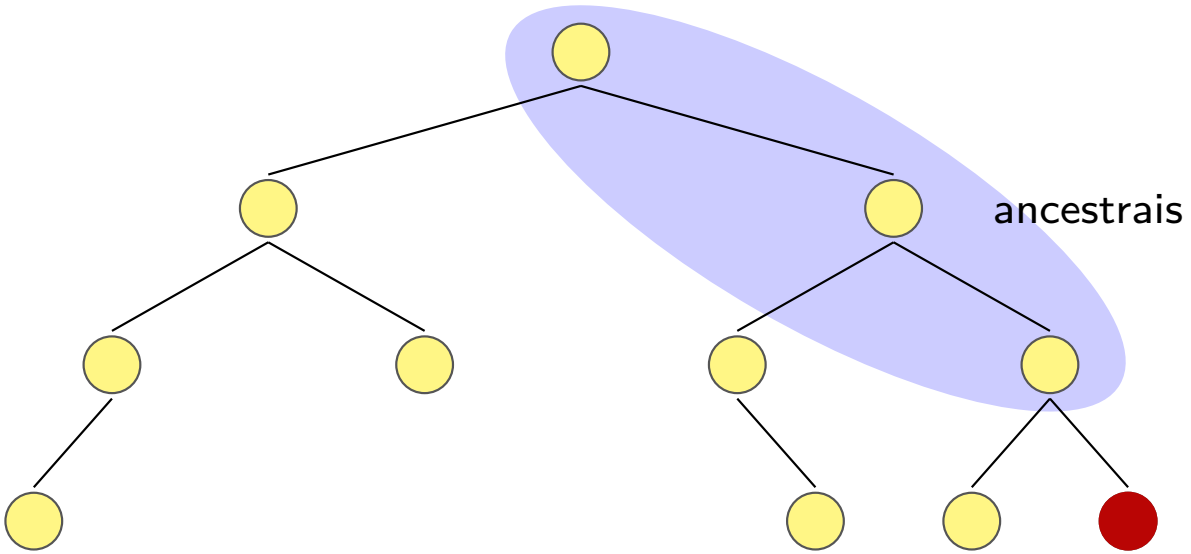
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



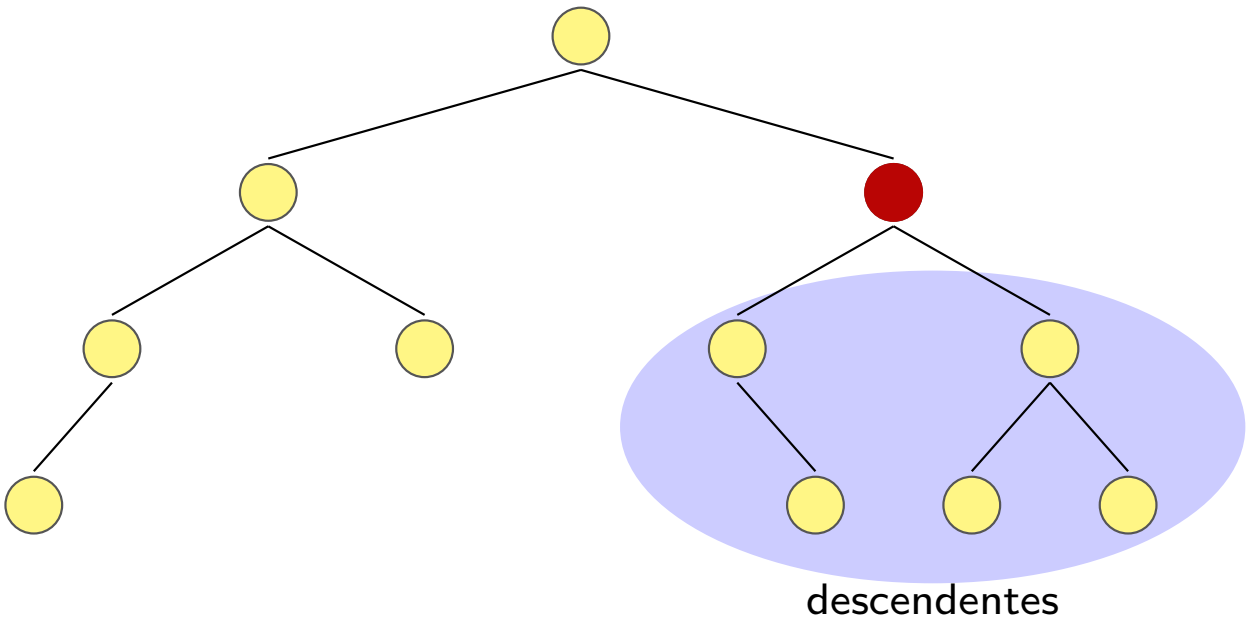
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



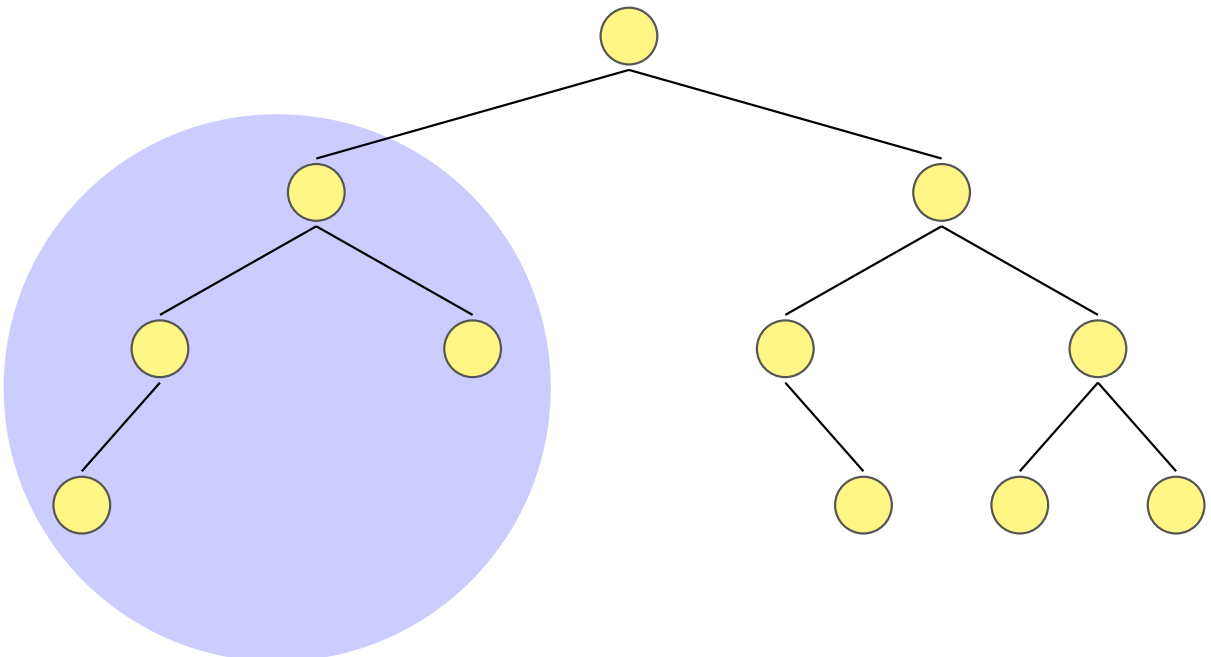
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



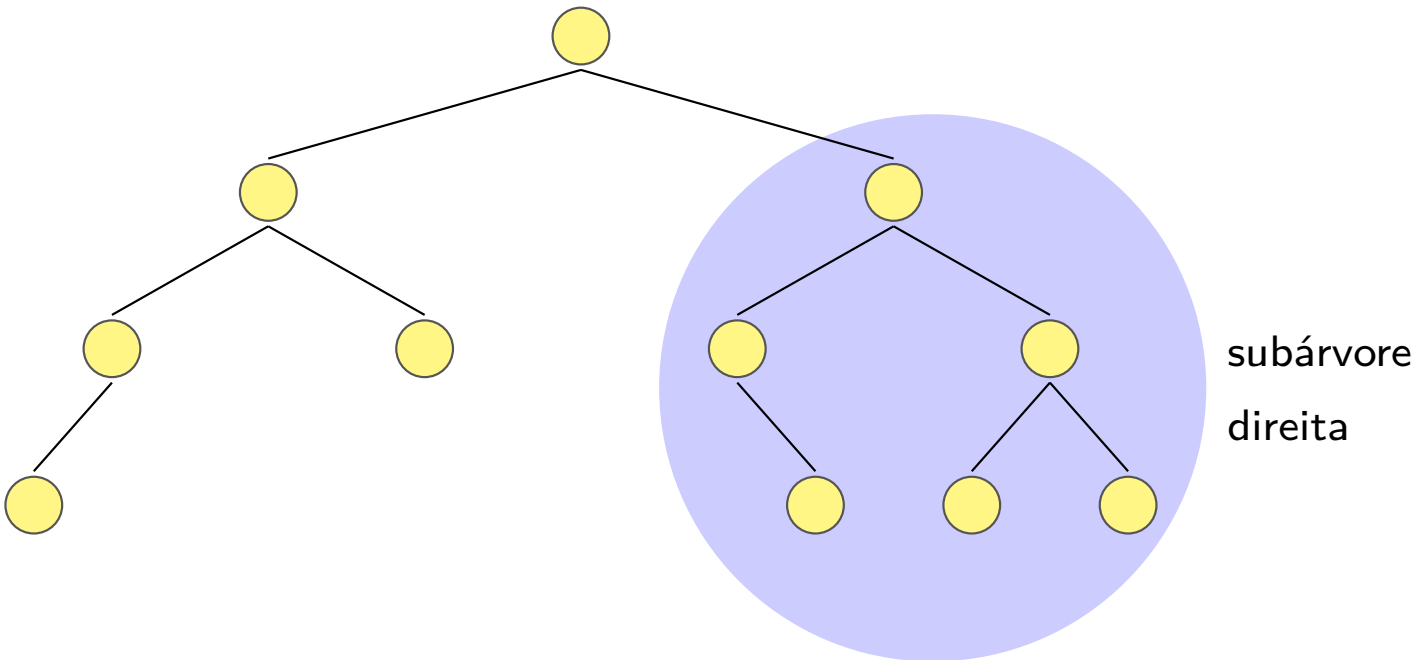
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



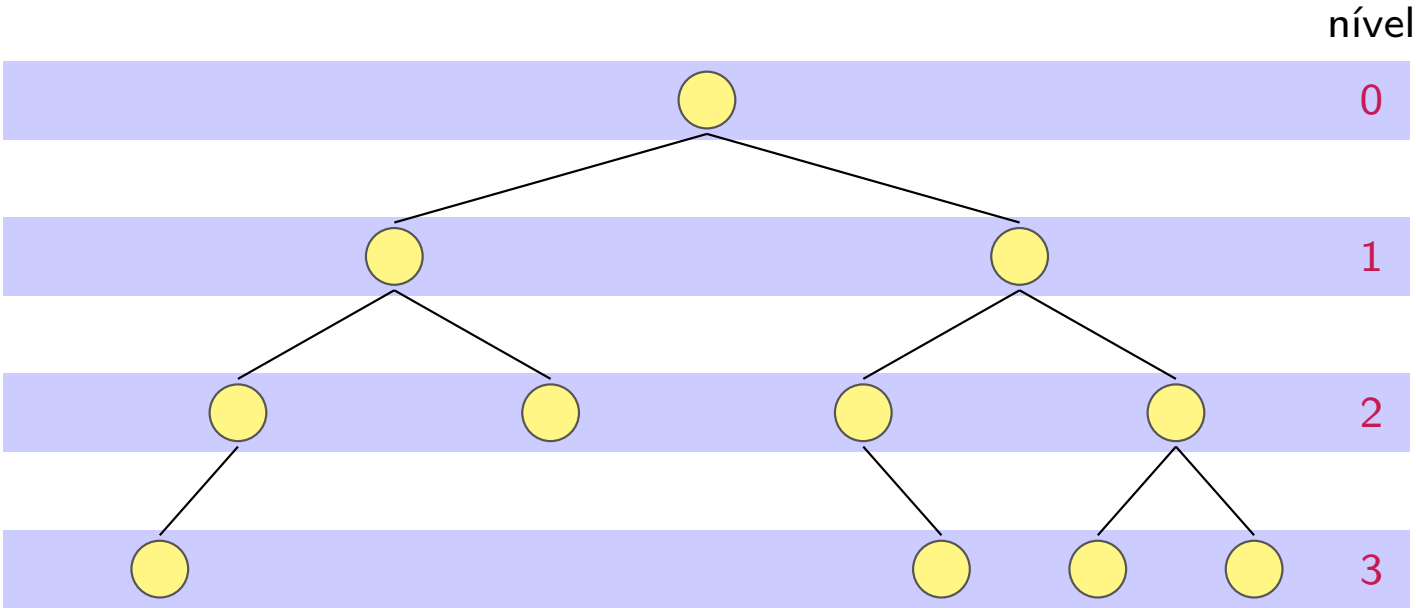
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.



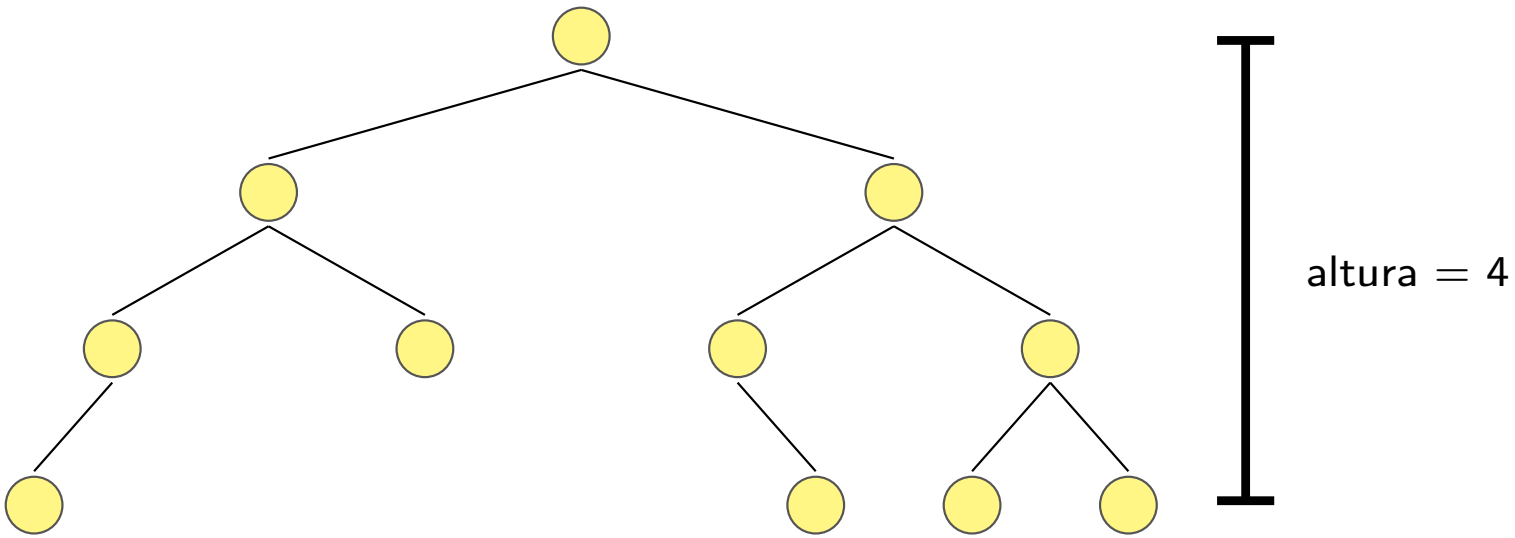
Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.

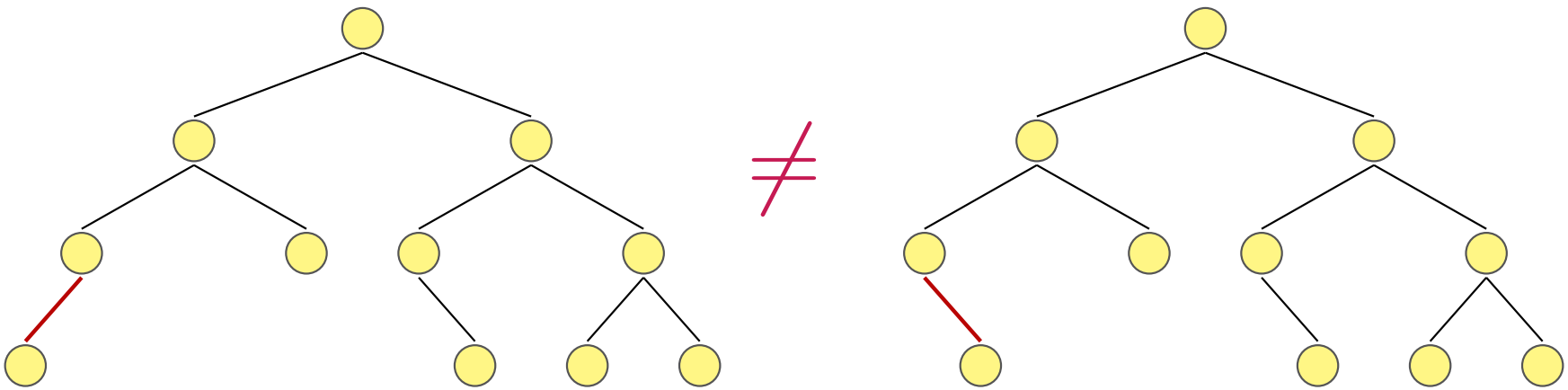


Árvores binárias

Uma árvore é uma forma de organizar os dados na memória: cada nó aponta para zero ou mais filhos.

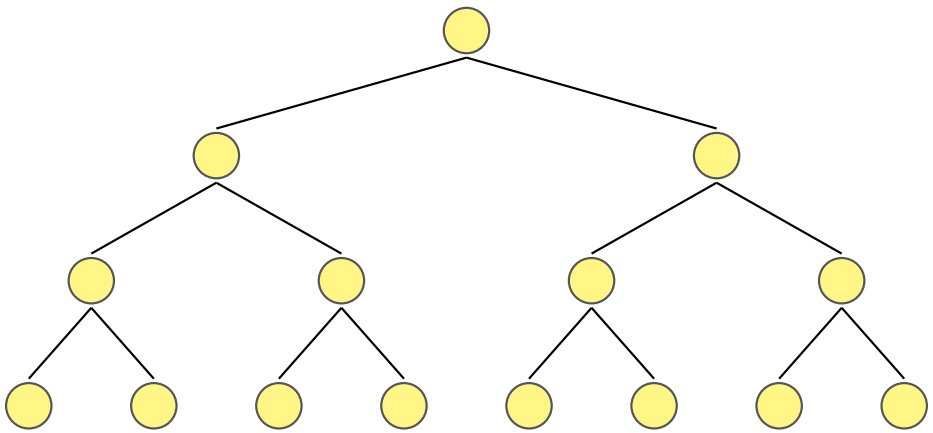


Comparando com atenção



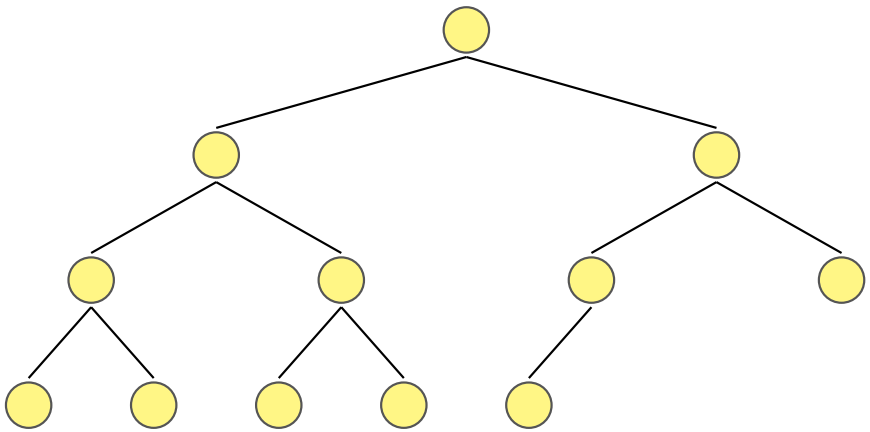
Ordem dos filhos é relevante!

Árvores binárias completas



- Uma árvore binária é **completa** se todos os nós internos têm dois filhos e todas as folhas estão no mesmo nível.

Árvores binárias quase completas



- Uma árvore binária é **quase-completa** se todos os seus níveis estão preenchidos, exceto talvez pelas folhas à direita do último nível.

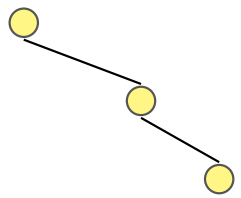
Relação entre altura e número de nós

Se a altura é h, então a árvore:

Relação entre altura e número de nós

Se a altura é h, então a árvore:

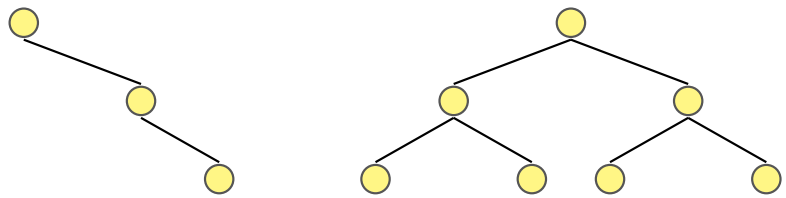
- tem no mínimo h nós



Relação entre altura e número de nós

Se a altura é h, então a árvore:

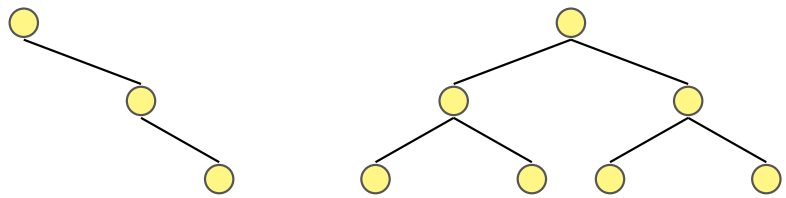
- tem no mínimo h nós
- tem no máximo $2^h - 1$ nós



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

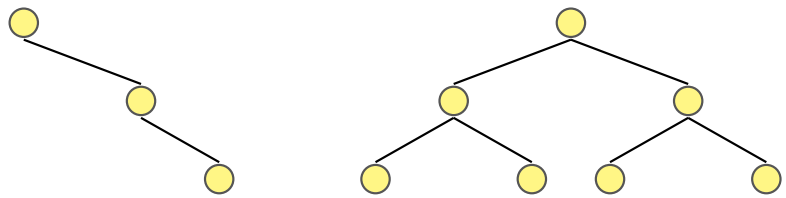


Se a árvore tem $n \geq 1$ nós, então:

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



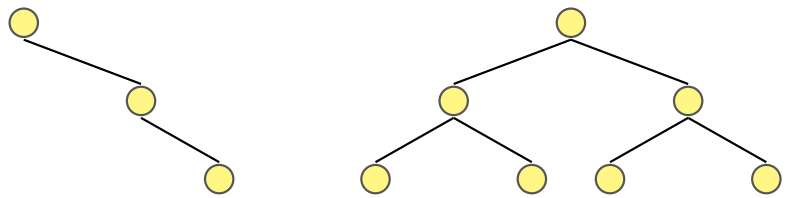
Se a árvore tem $n \geq 1$ nós, então:

- 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



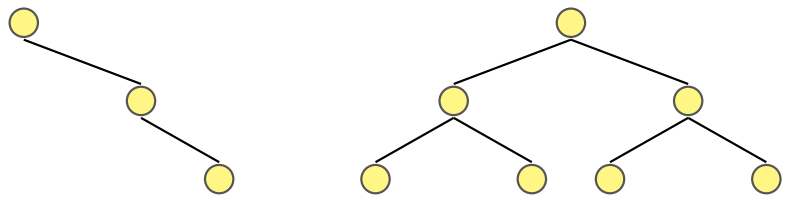
Se a árvore tem $n \geq 1$ nós, então:

- 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



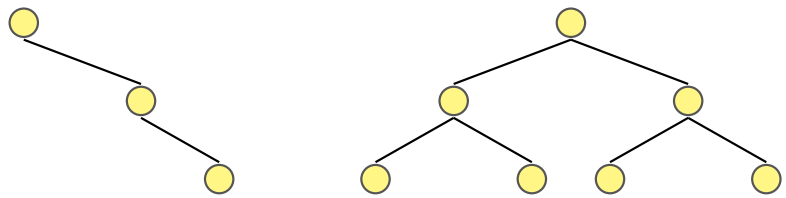
Se a árvore tem $n \geq 1$ nós, então:

- 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

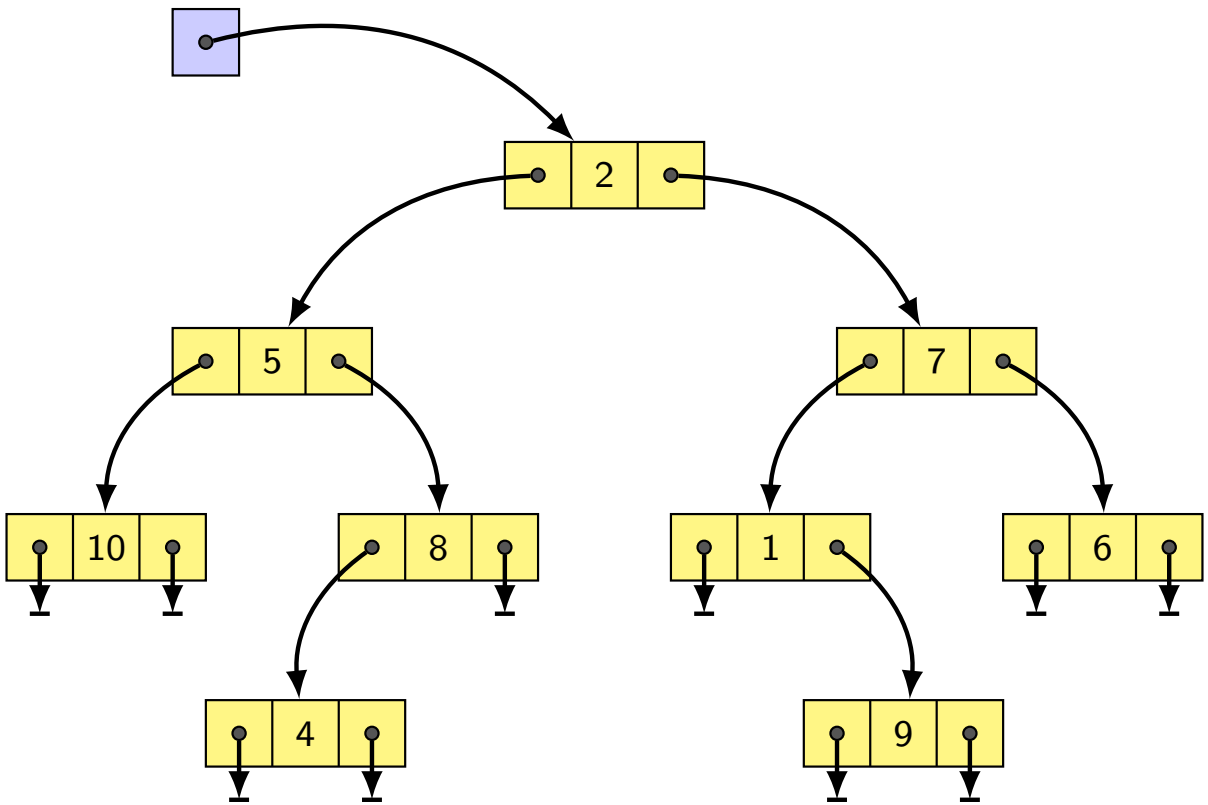


Se a árvore tem $n \geq 1$ nós, então:

- 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 (lista)

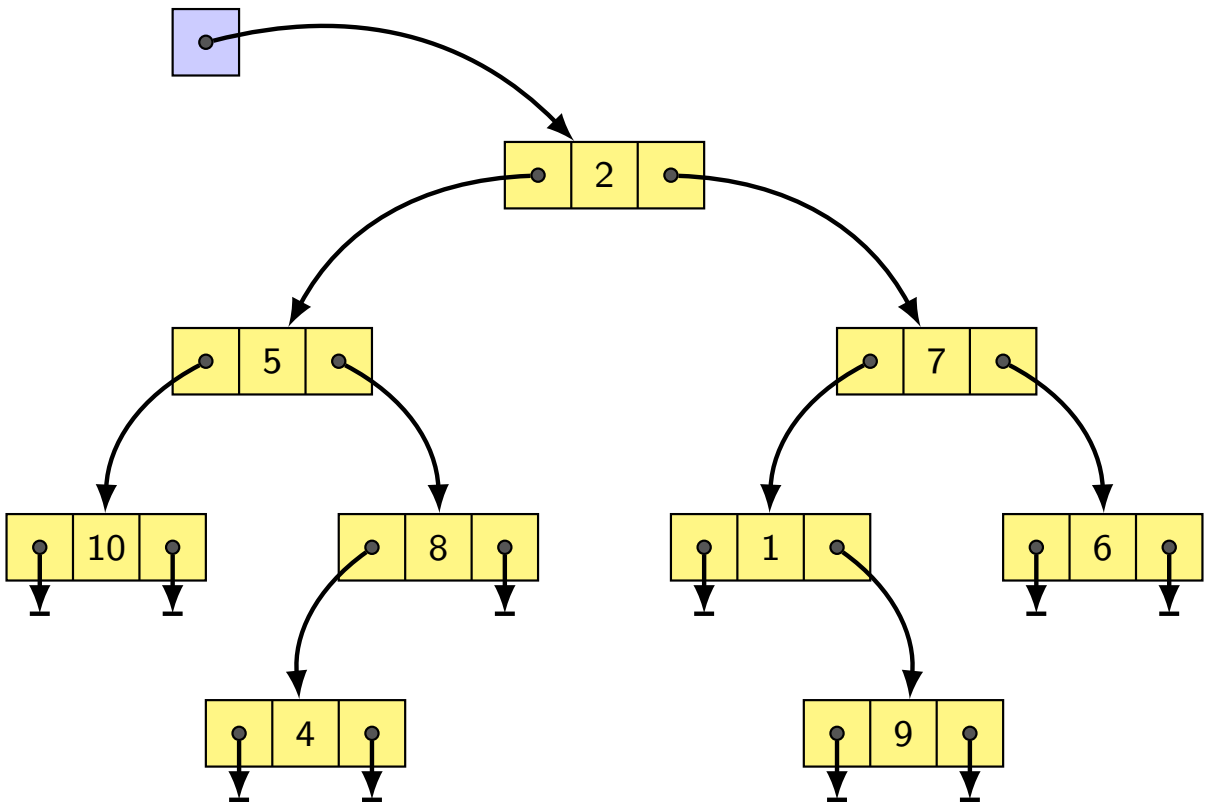
Árvores binárias: Implementação

```
5 typedef struct no {
6     int dado;
7     struct no *esq, *dir; // *pai
8 } No;
```



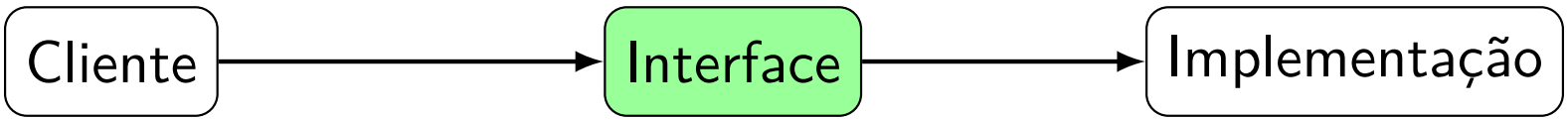
Árvores binárias: Implementação

```
5  typedef struct no {
6      int dado;
7      struct no *esq, *dir; // *pai
8  } No;
```



E se quisermos saber o pai de um nó?

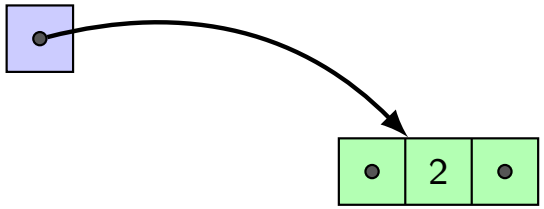
TAD - Interface



arvore.h

```c
#ifndef ARVORE_BINARIA_H
#define ARVORE_BINARIA_H

//Dados
typedef struct no {
    int dado;
    struct no *esq, *dir; // *pai
} No;

//Funções
No* criar_no(int x, No *l, No *r);
void destruir_arvore(No **raiz);
void imprimir_arvore(No *raiz);

No* procurar_no(No *raiz, int x);

int numero_nos(No *raiz);

int altura(No *raiz);

#endif
```

Árvores binárias - Criar Nó

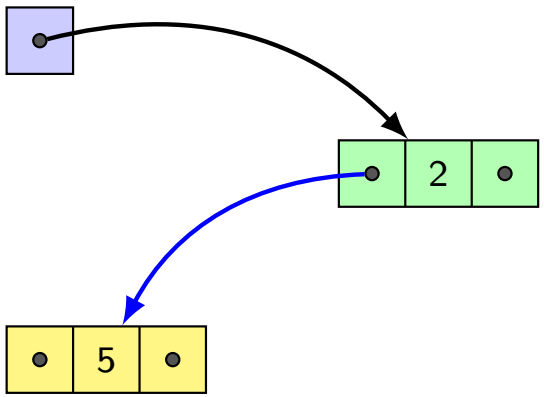


arvore.c

```
5  No* criar_no(int x, No *l, No *r) {
6    No *no = (No*) malloc(sizeof(No));
7    no->dado = x;
8    no->esq = l;
9    no->dir = r;
10   return no;
11 }
```

Árvores são definidas recursivamente

Árvores binárias - Criar Nó

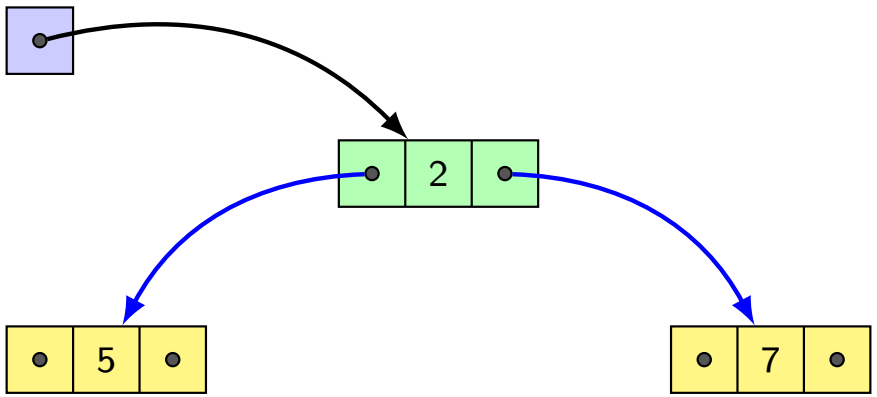


arvore.c

```
5  No* criar_no(int x, No *l, No *r) {
6    No *no = (No*) malloc(sizeof(No));
7    no->dado = x;
8    no->esq = l;
9    no->dir = r;
10   return no;
11 }
```

Árvores são definidas recursivamente

Árvores binárias - Criar Nó

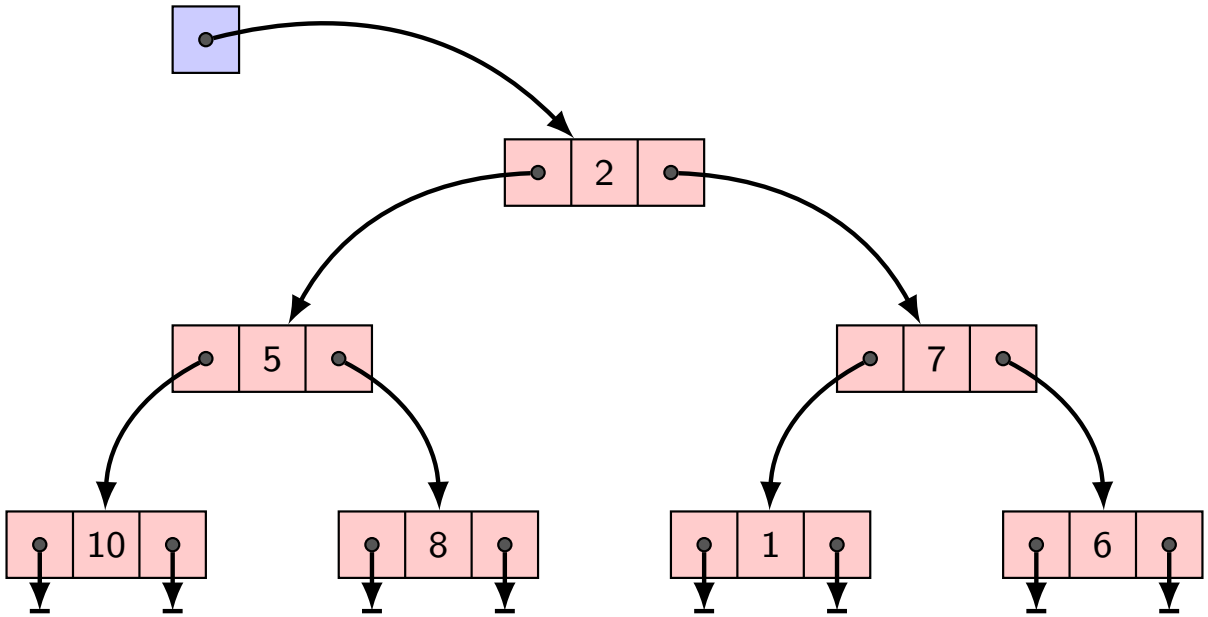


arvore.c

```
5  No* criar_no(int x, No *l, No *r) {
6    No *no = (No*) malloc(sizeof(No));
7    no->dado = x;
8    no->esq = l;
9    no->dir = r;
10   return no;
11 }
```

Árvores são definidas recursivamente

Árvores binárias - Destruir árvore



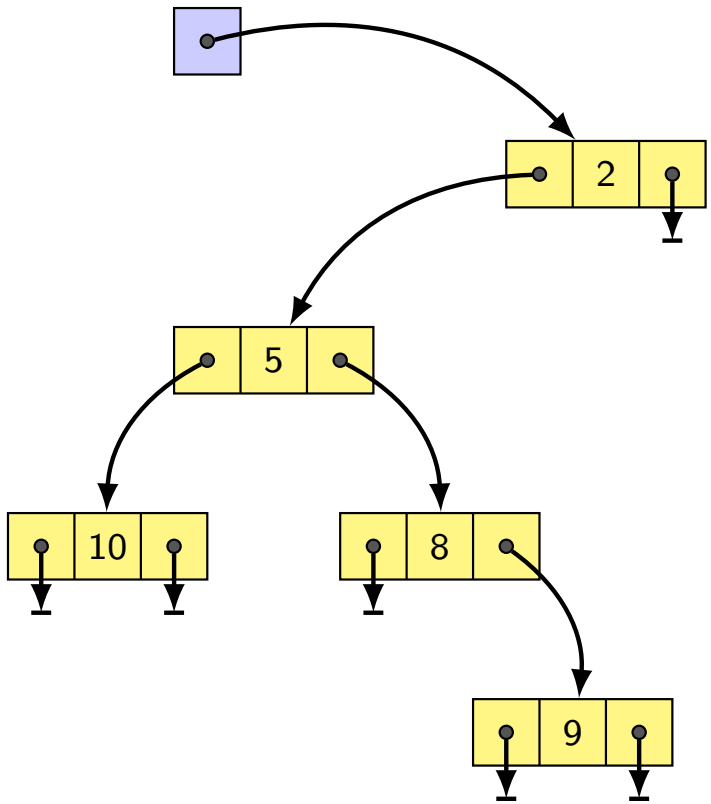
arvore.c

```
13 void destruir_arvore(No **raiz){//função recursiva
14   if (*raiz == NULL) return;
15   destruir_arvore(&((*raiz)->esq));
16   destruir_arvore(&((*raiz)->dir));
17   free(*raiz);
18   *raiz = NULL;
19 }
```

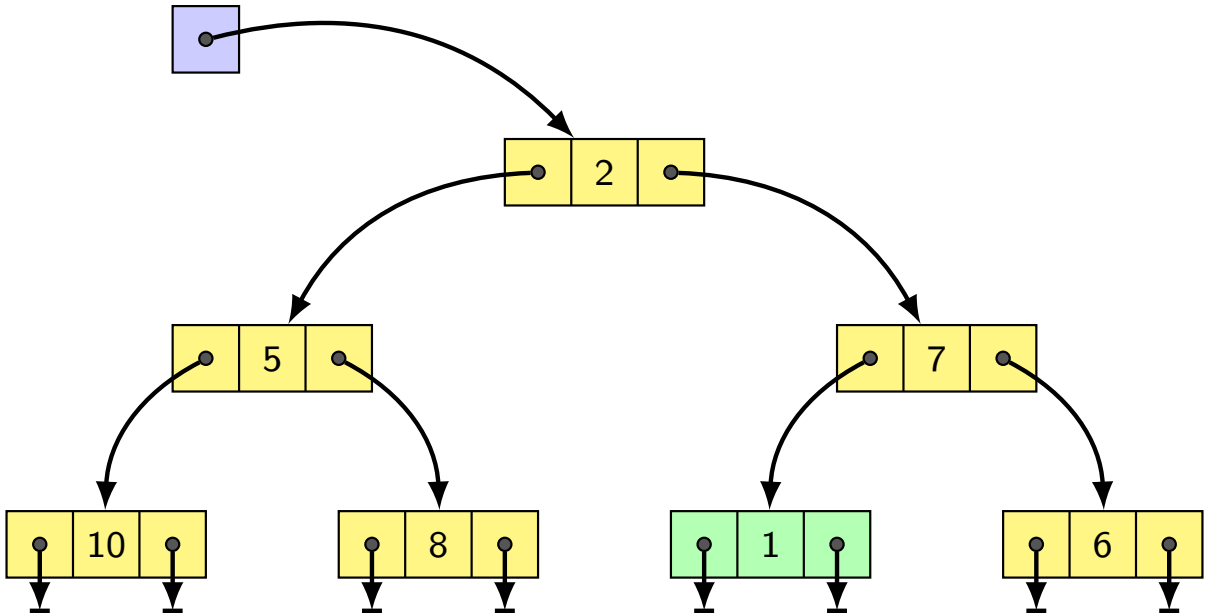
Árvores binárias - Imprimir árvore

arvore.c

```
21 void printnode(int x, int h) {
22   int i;
23   for (i = 0; i < h; i++) printf("-");
24   printf("%2d\n", x);
25 }
26
27 void print(No *p, int h) {
28   if (p == NULL)
29     return;
30   print(p->dir, h+1);
31   printnode(p->dado, h);
32   print(p->esq, h+1);
33 }
34
35 void imprimir_arvore(No *raiz){
36   print(raiz, 0);
37 }
```



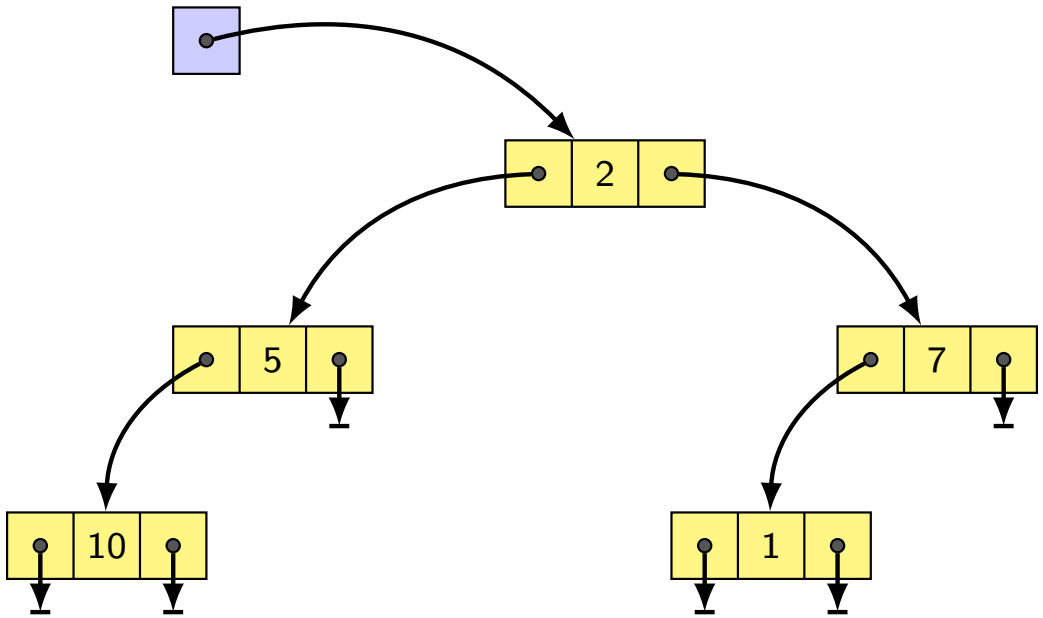
Árvores binárias - Buscar valor



arvore.c

```
39 No* procurar_no(No *raiz, int x) {
40   if (raiz == NULL || raiz->dado == x)
41     return raiz;
42   No *esq = procurar_no(raiz->esq, x);
43   if (esq != NULL)
44     return esq;
45   return procurar_no(raiz->dir, x);
46 }
```

Árvores binárias - Calcular o número de nós



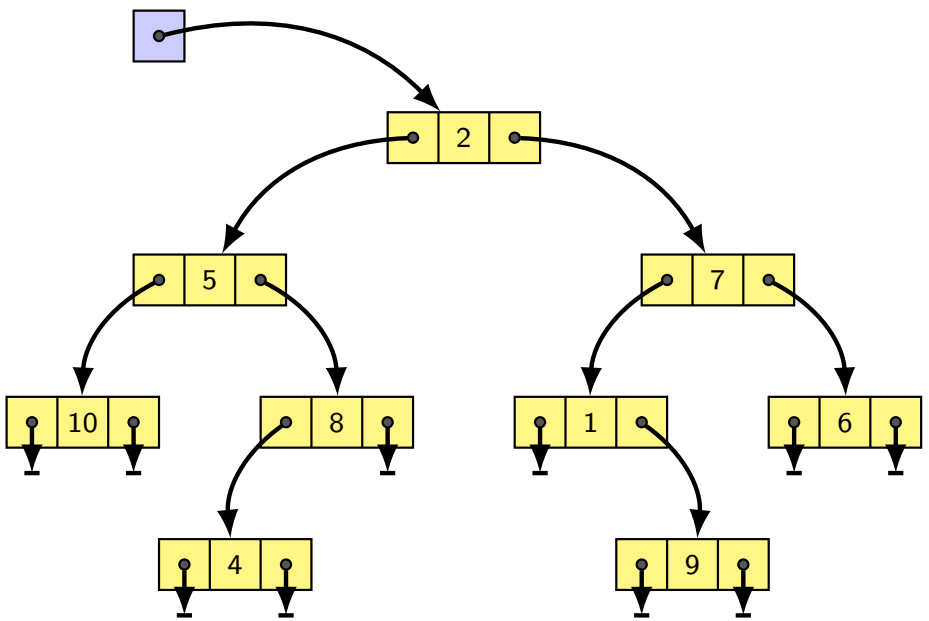
arvore.c

```
48 int numero_nos(No *raiz) {
49   if (raiz == NULL)
50     return 0;
51   return numero_nos(raiz->esq) + numero_nos(raiz->dir) + 1;
52 }
```

Árvores binárias - Altura da árvore

arvore.c

```
54 int altura(No *raiz) {
55   int h_esq, h_dir;
56   if (raiz == NULL)
57     return 0;
58   h_esq = altura(raiz->esq);
59   h_dir = altura(raiz->dir);
60   return (h_esq > h_dir ? h_esq : h_dir) + 1;
61 }
```



Roteiro

Exemplo: Criando um torneio

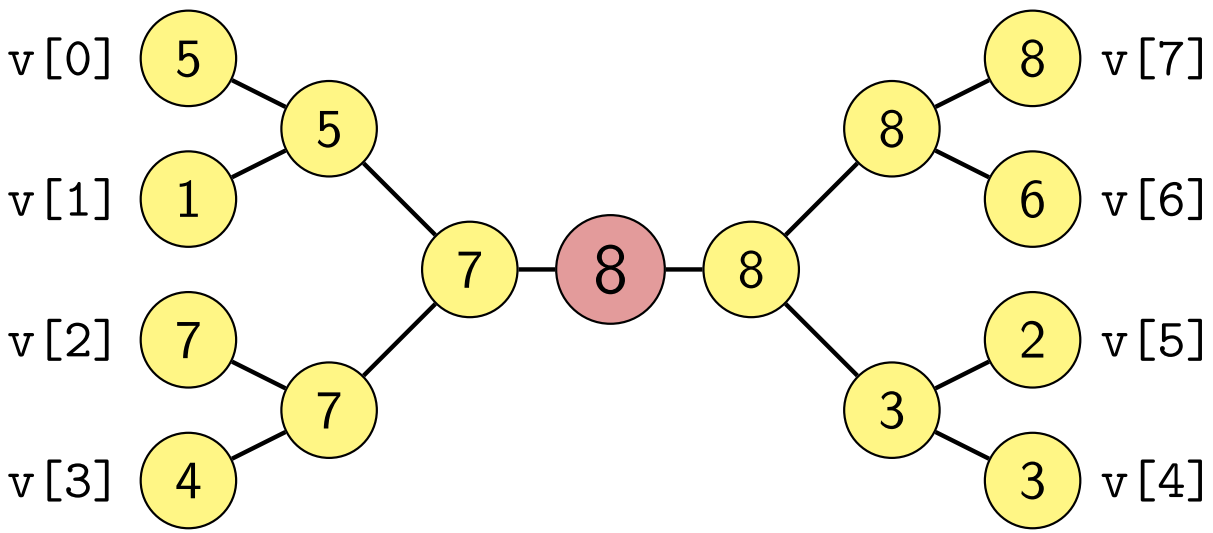
Dado um vetor `v` com `n` números, queremos criar um torneio

- 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

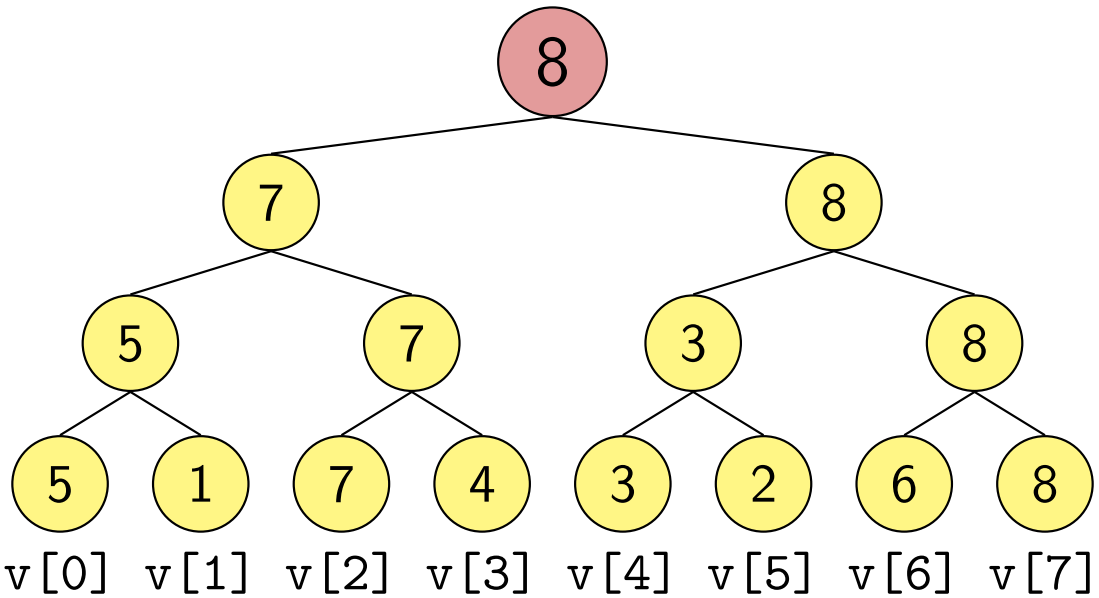
Dado um vetor `v` com `n` números, queremos criar um torneio

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

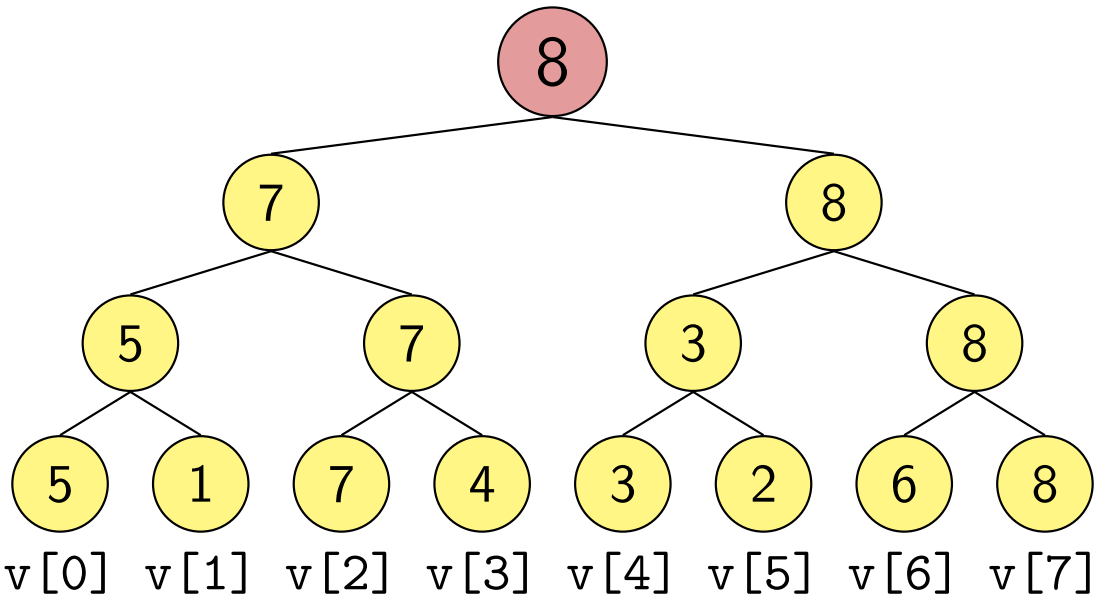


É uma árvore binária, onde o valor do pai é o maior valor dos seus filhos

Exemplo: Criando um torneio

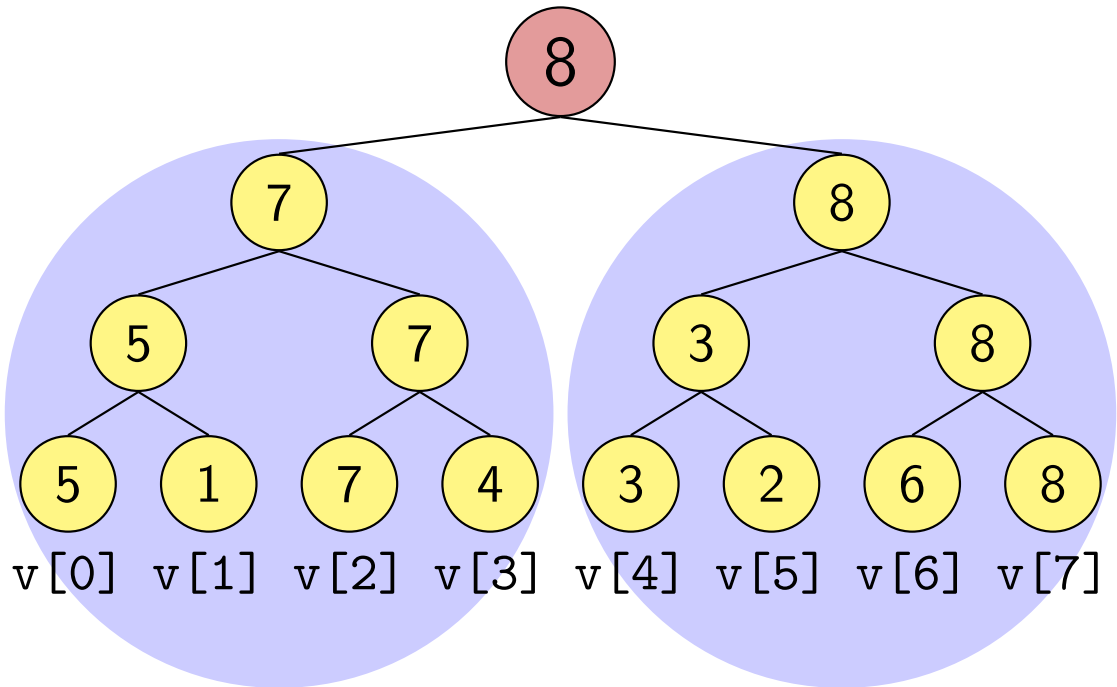


Exemplo: Criando um torneio



Para resolver o torneio:

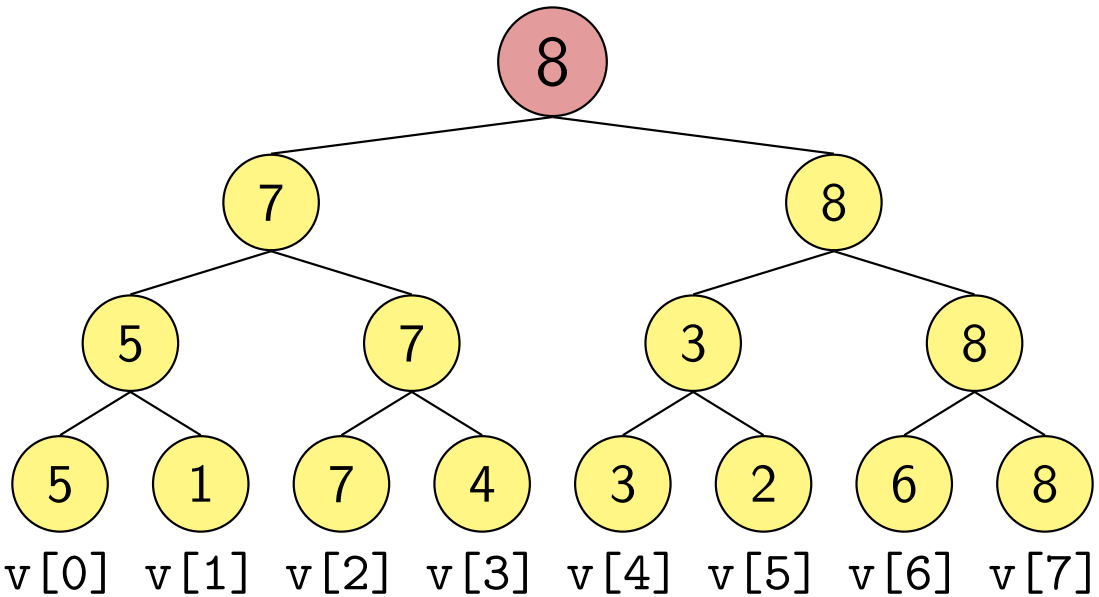
Exemplo: Criando um torneio



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

Cliente

exemplo1.c

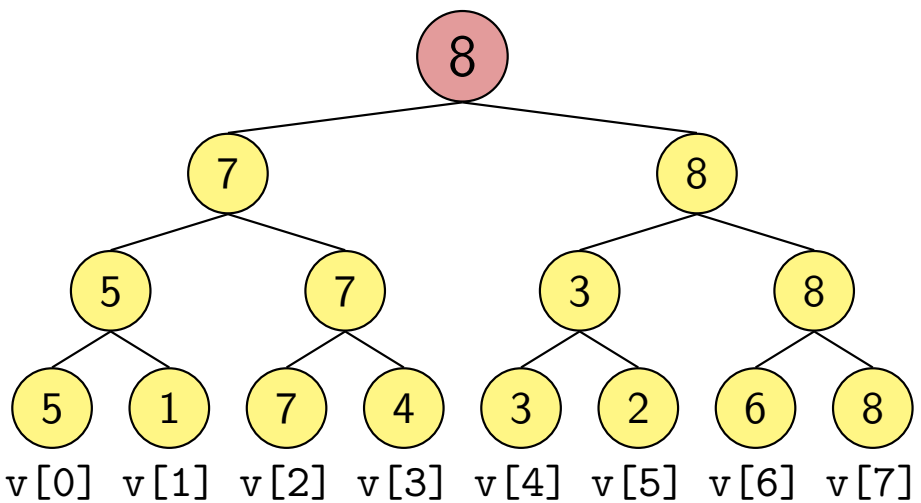
```c
#include <stdio.h>
#include "arvore.h"

No *torneio(int *v, int l, int r);

int main() {
  int teste[8] = {5, 1, 7, 4, 3, 2, 6, 8};
  No *T = torneio(teste, 0, 7);
  imprimir_arvore(T);
  printf("nós: %d\n", numero_nos(T));
  printf("altura: %d\n", altura(T));
  destruir_arvore(&T);
  return 0;
}
```

Exemplo: Criando um torneio

```
16 No *torneio(int *v, int l, int r) {
17   No *esq, *dir;
18   int valor, m = (l + r) / 2;
19   if (l == r)
20     return criar_no(v[l], NULL, NULL);
21   esq = torneio(v, l, m);
22   dir = torneio(v, m+1, r);
23   valor = (esq->dado > dir->dado) ? esq->dado : dir->dado;
24   return criar_no(valor, esq, dir);
25 }
```

Exemplo: Criando um torneio

```
16 No *torneio(int *v, int l, int r) {
17   No *esq, *dir;
18   int valor, m = (l + r) / 2;
19   if (l == r)
20     return criar_no(v[l], NULL, NULL);
21   esq = torneio(v, l, m);
22   dir = torneio(v, m+1, r);
23   valor = (esq->dado > dir->dado) ? esq->dado : dir->dado;
24   return criar_no(valor, esq, dir);
25 }
```



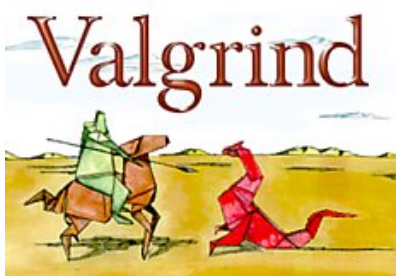
Makefile e Valgrind

Vamos usar o Makefile para compilar:

```
1 exemplo1: exemplo1.c arvore.o
2   gcc $^ -o $@
```

Vamos verificar a memória com o Valgrind:

```
1 $ valgrind --leak-check=yes ./exemplo1
```

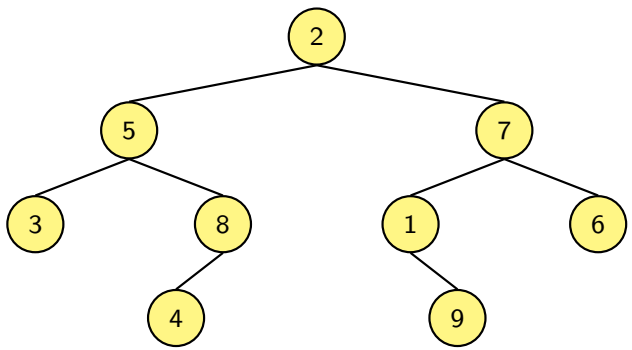


Roteiro

Percursos em uma árvore

Em uma árvore há várias formas de percorrer os nós a partir da raiz.

- Alguns percursos são úteis para obter informações a respeito da árvore ou dos dados armazenados na árvore.
- Há dois principais:
  1. em **profundidade**: "para baixo primeiro".
  2. em **largura**: "para o lado primeiro".



Durante o percurso geralmente realiza-se alguma operação em cada nó, que vamos chamar genericamente de **visitar**.

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 acessa a subárvore esquerda

Percorrendo os nós - Pré-ordem

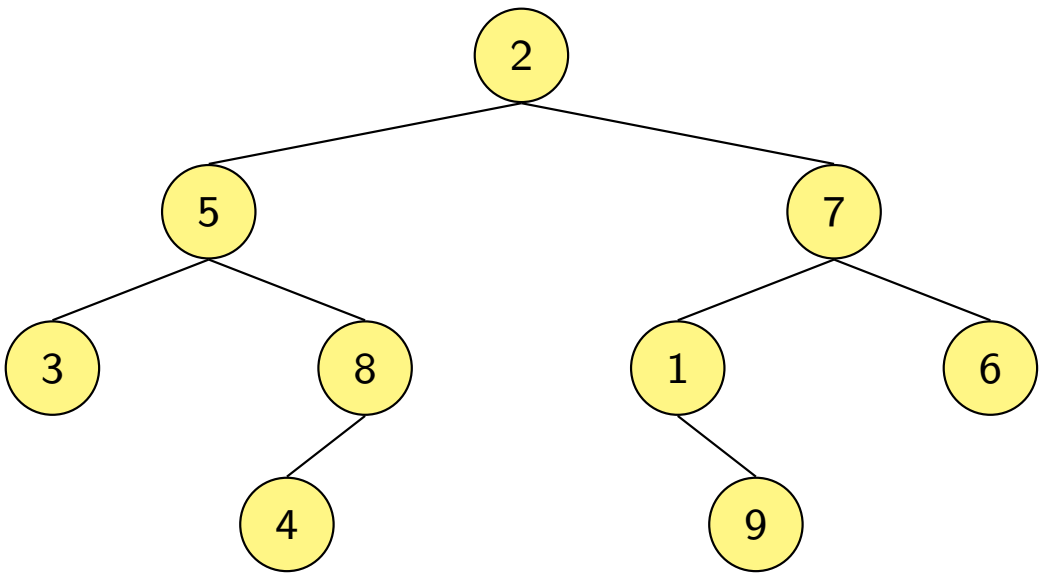
A pré-ordem

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

Percorrendo os nós - Pré-ordem

A pré-ordem

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

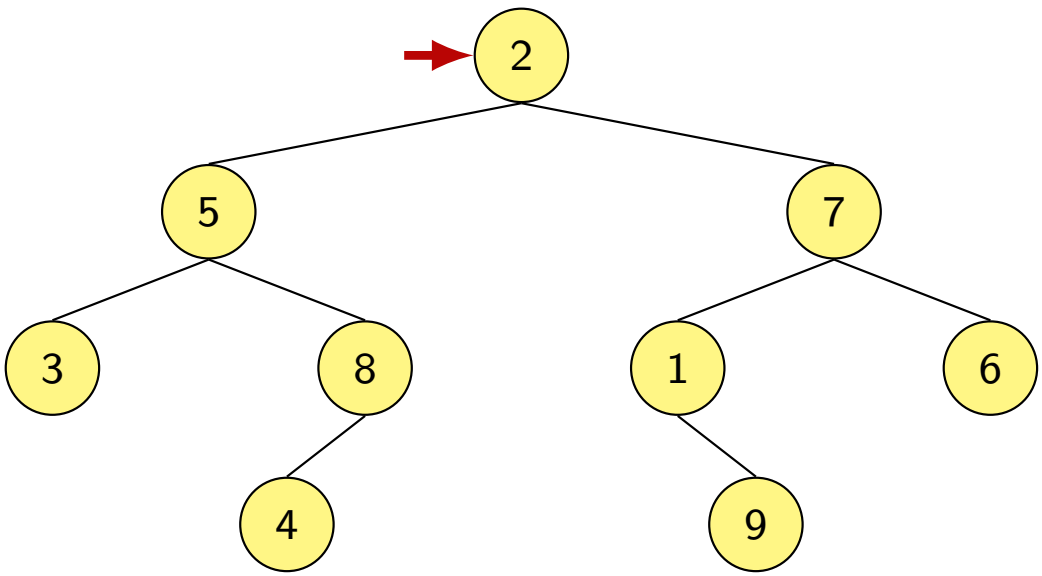


Ex:

Percorrendo os nós - Pré-ordem

A pré-ordem

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

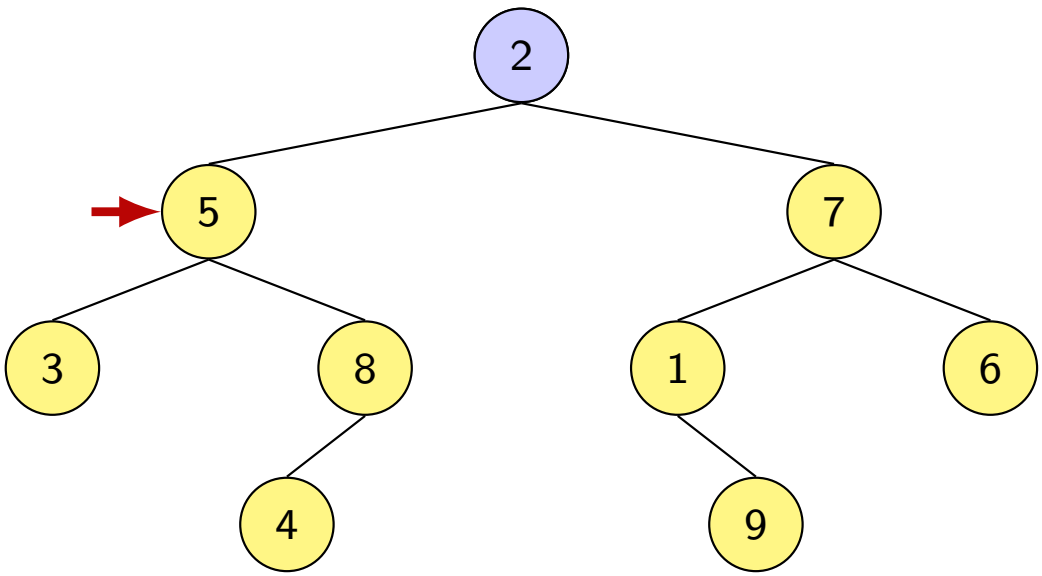


Ex:

Percorrendo os nós - Pré-ordem

A pré-ordem

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

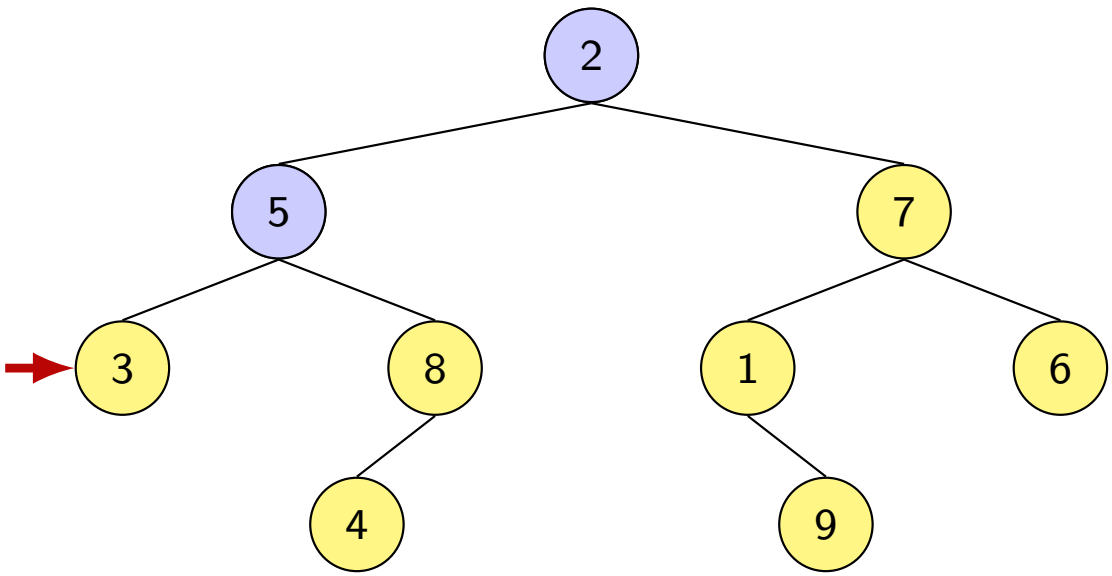


Ex: 2,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

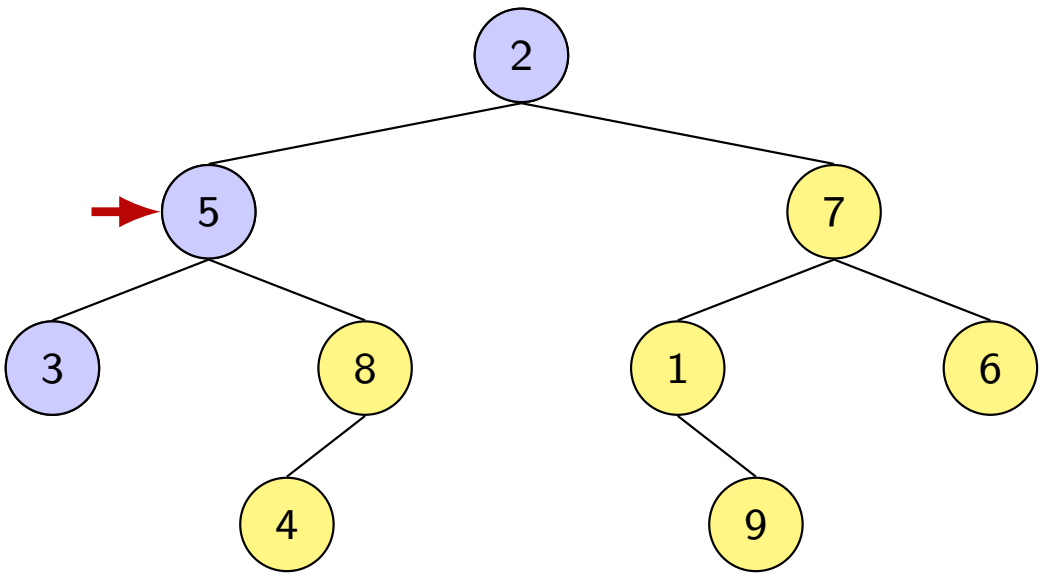


Ex: 2, 5,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

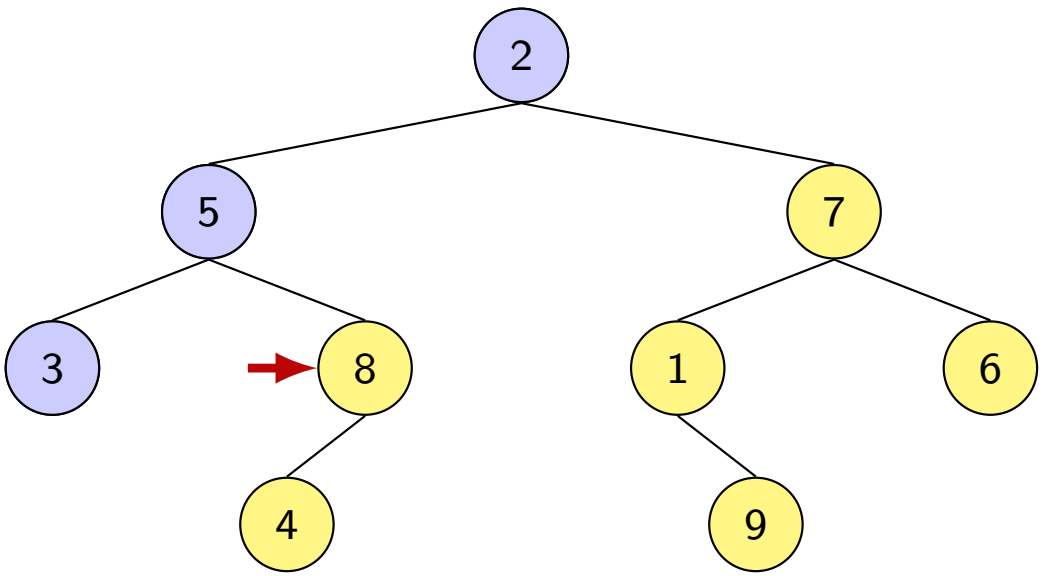


Ex: 2, 5, 3,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

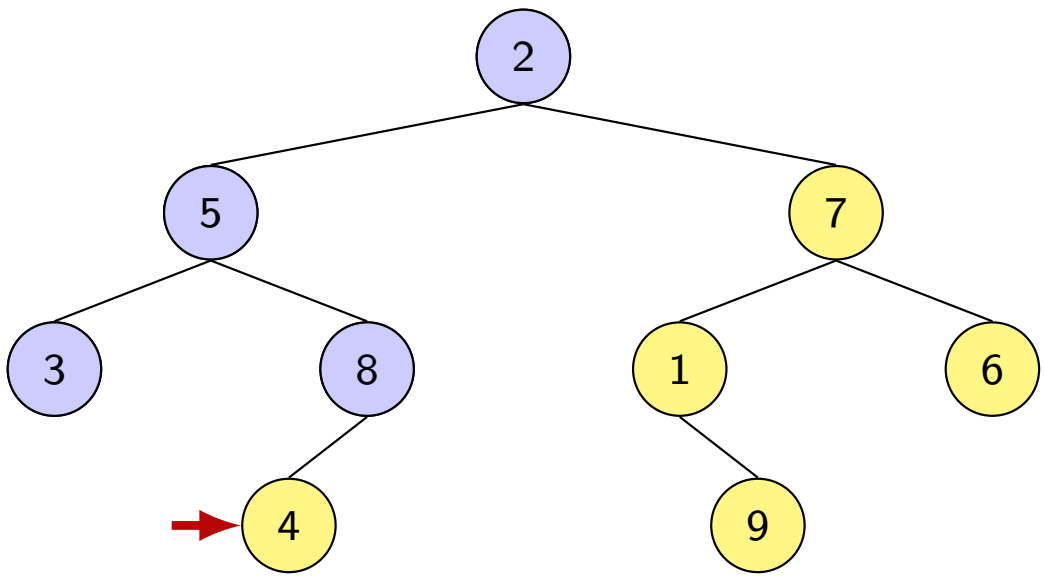


Ex: 2, 5, 3,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

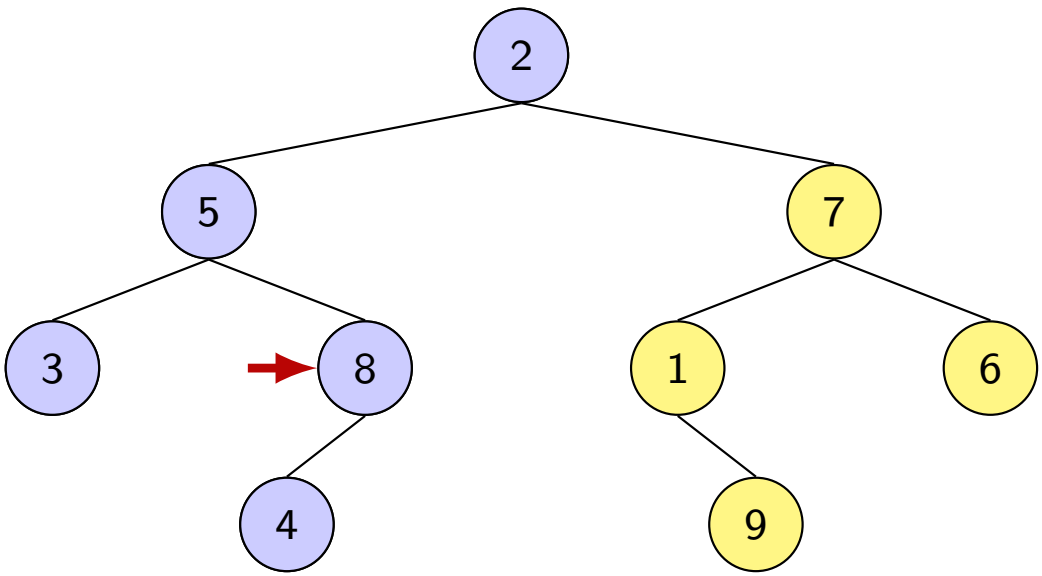


Ex: 2, 5, 3, 8,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

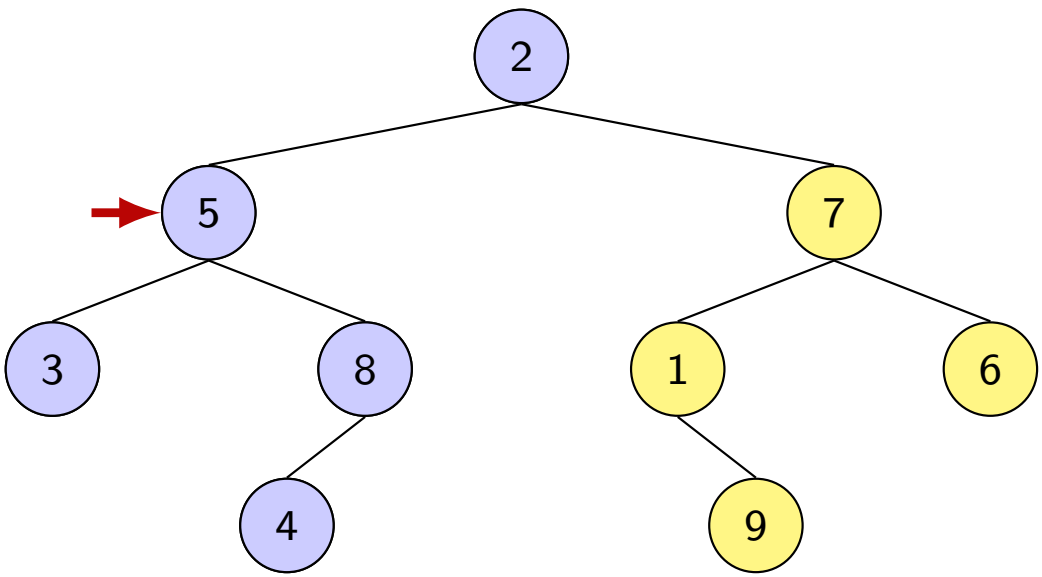


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

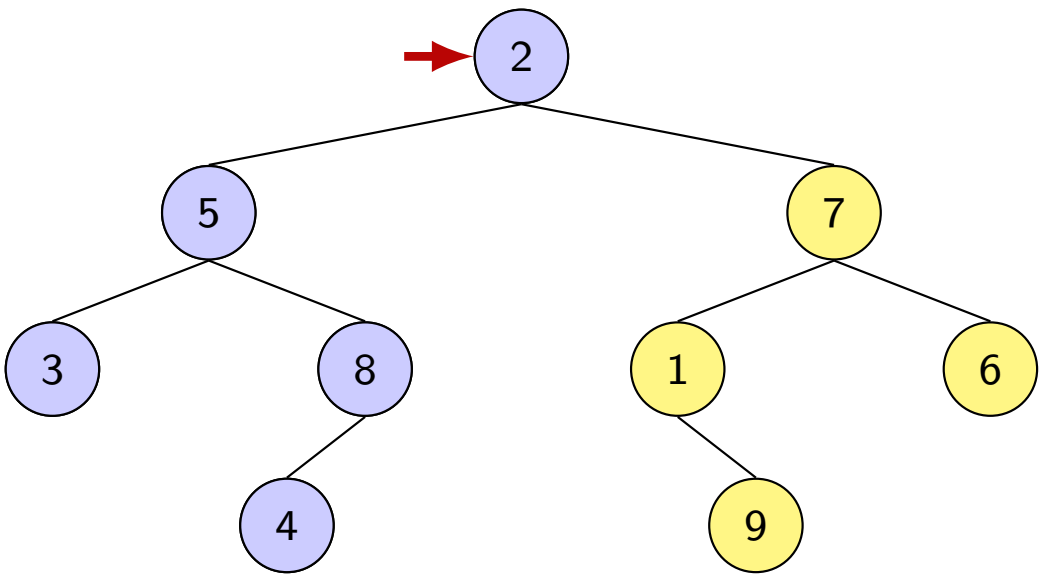


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

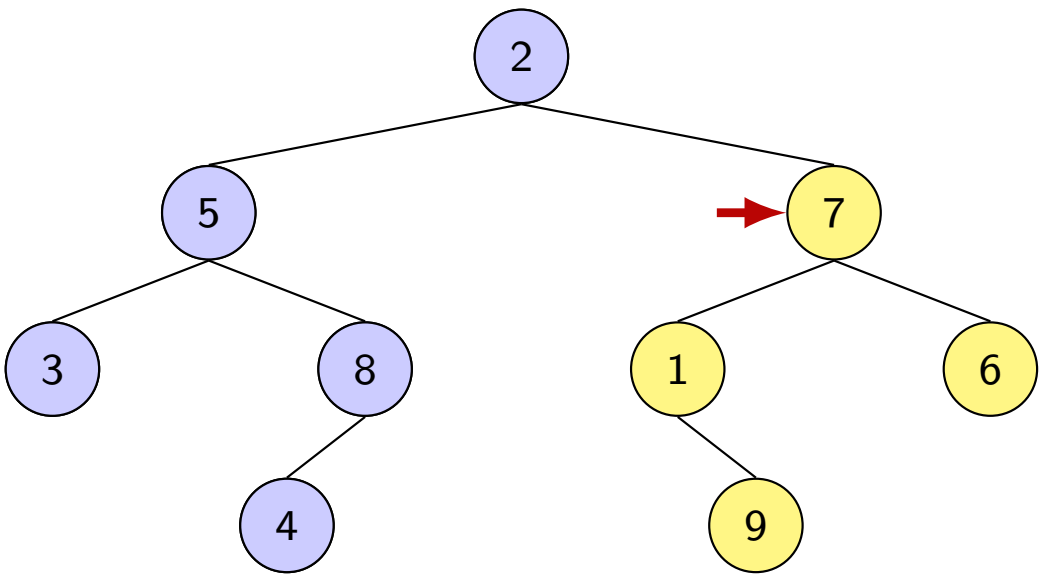


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

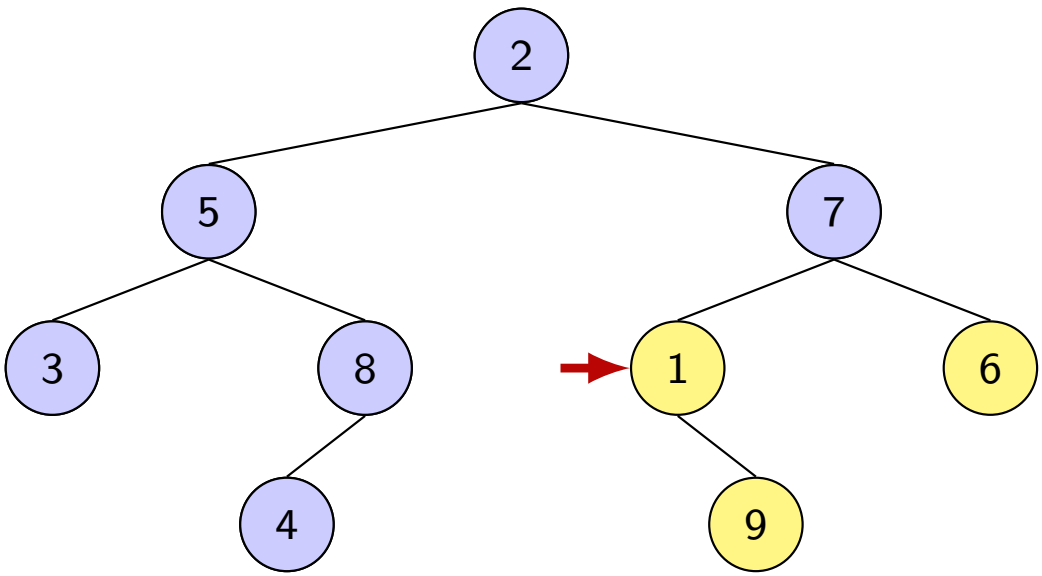


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

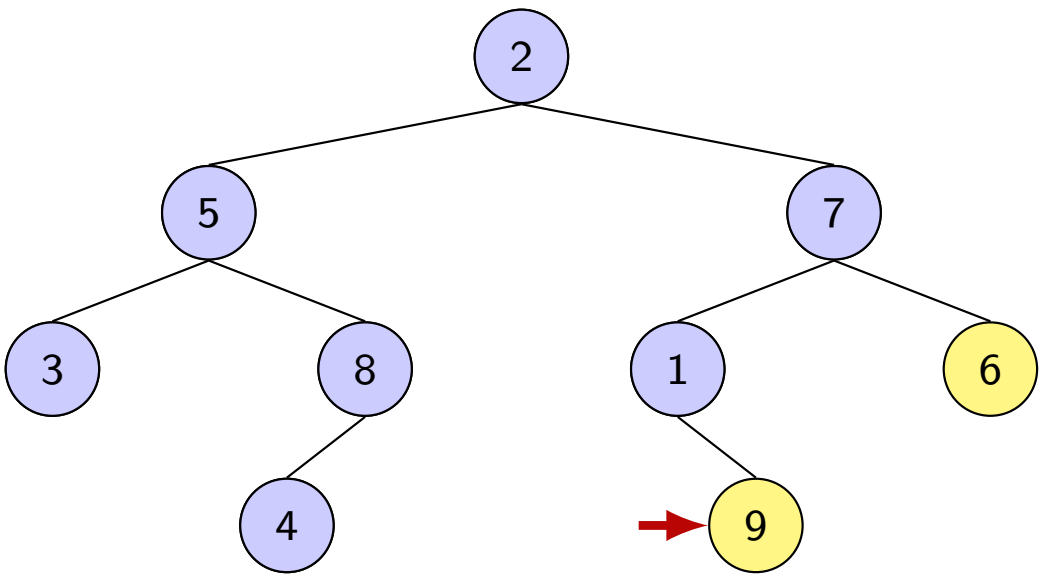


Ex: 2, 5, 3, 8, 4, 7,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

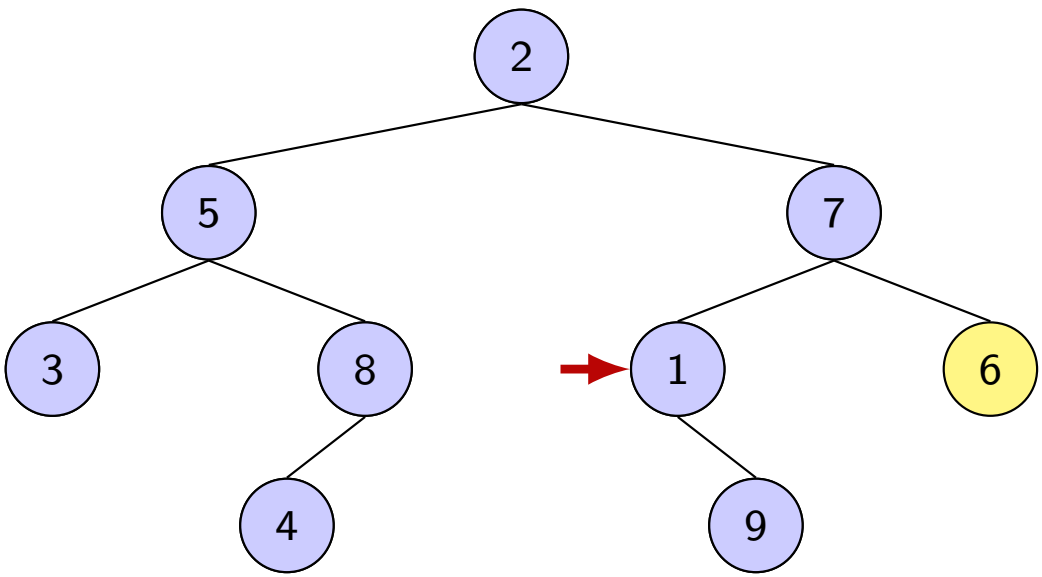


Ex: 2, 5, 3, 8, 4, 7, 1,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

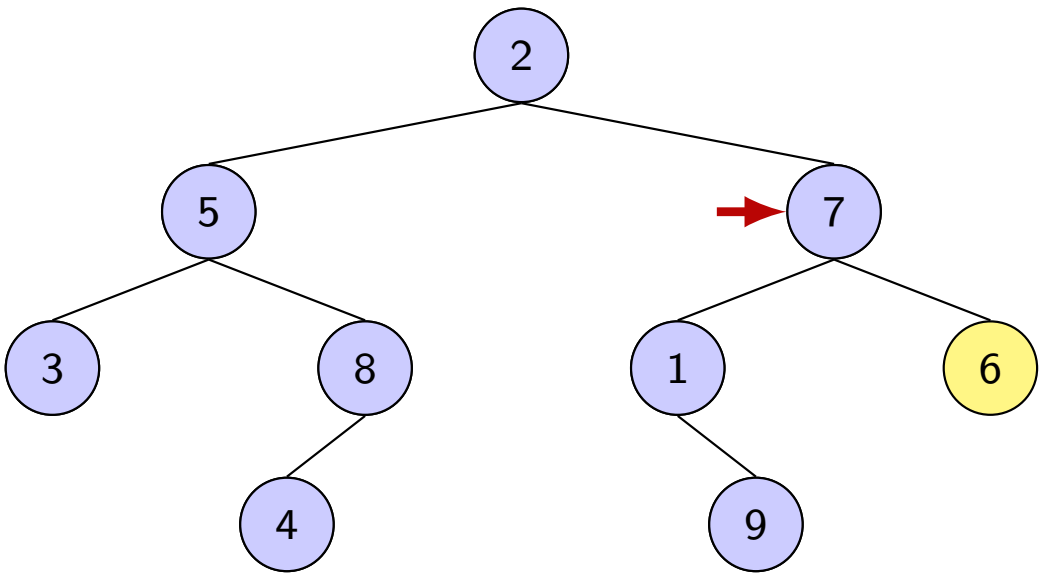


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

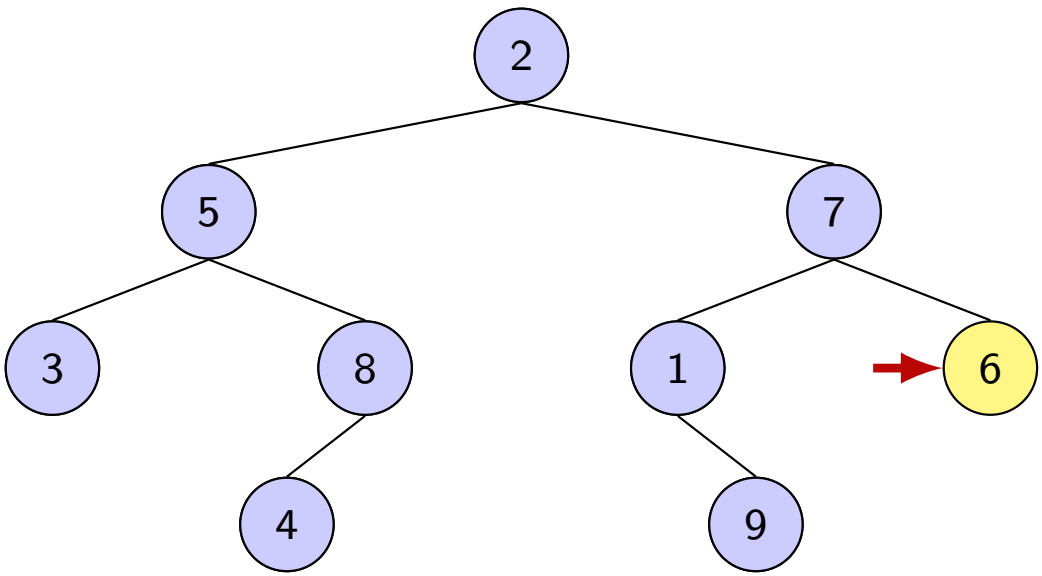


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

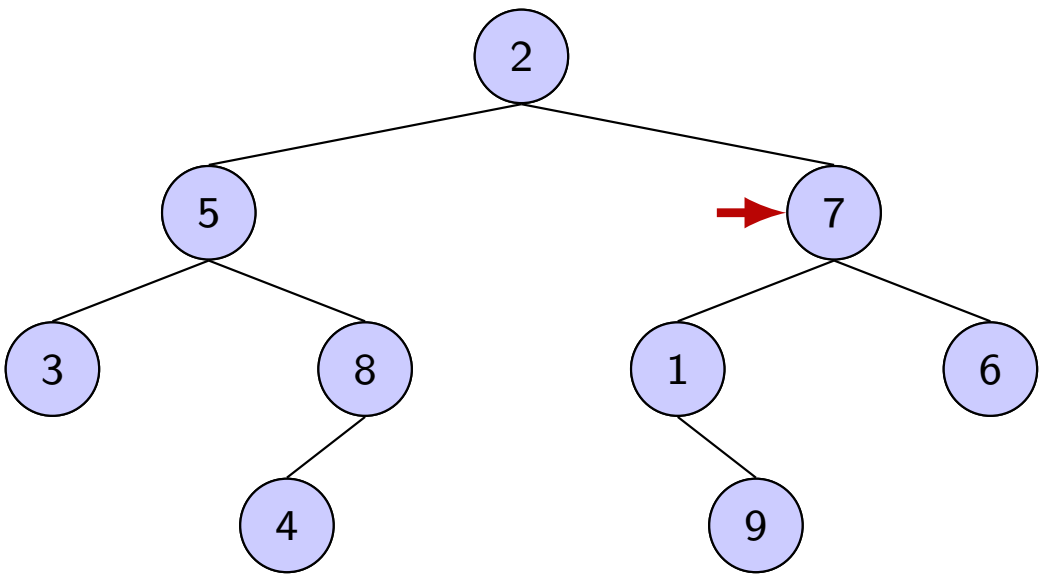


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

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

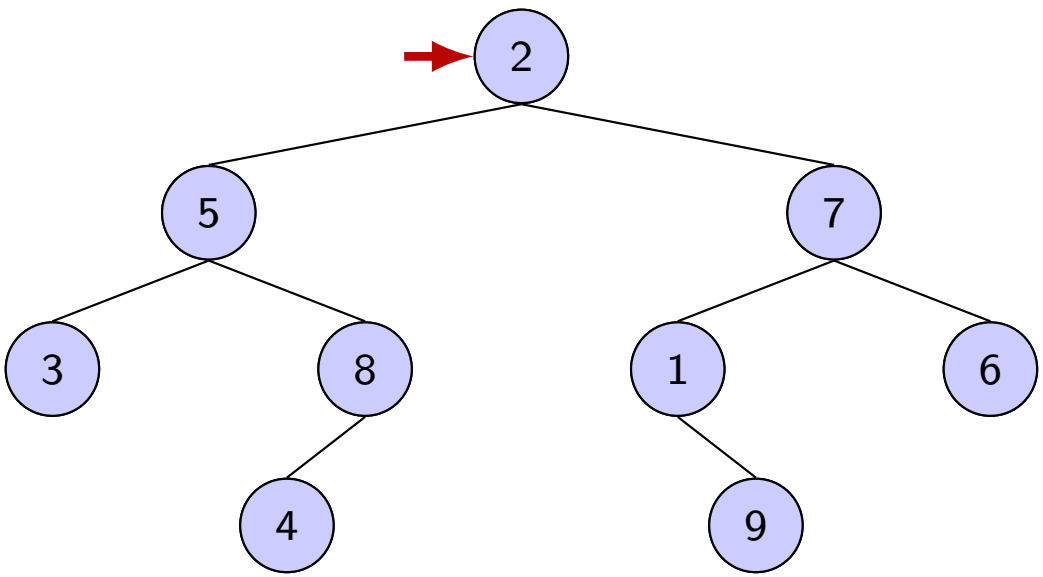


Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

Percorrendo os nós - Pré-ordem

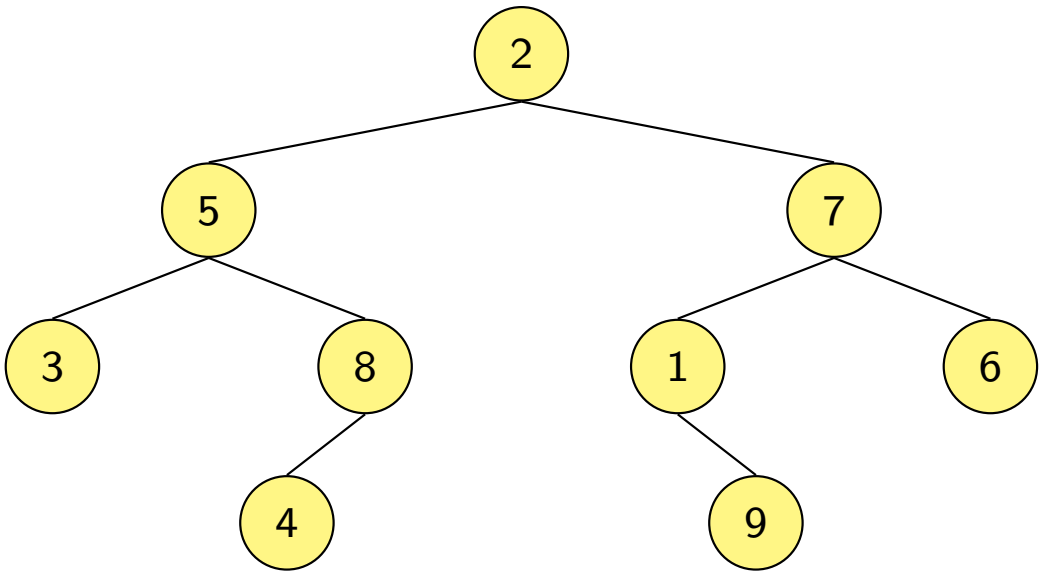
A pré-ordem

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



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

Implementação de percurso Pré-ordem




```
63 void pre_ordem(No *raiz) {
64   if (raiz != NULL) {
65     printf("%d ", raiz->dado); /* visita raiz */
66     pre_ordem(raiz->esq);
67     pre_ordem(raiz->dir);
68   }
69 }
```

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 acessa a subárvore esquerda

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita

Percorrendo os nós - Pós-ordem

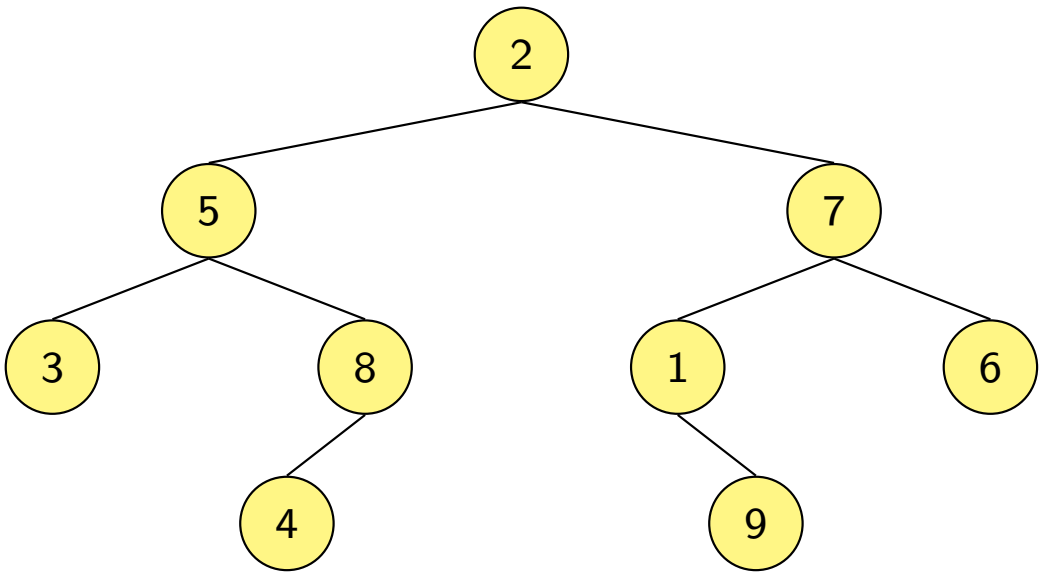
A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

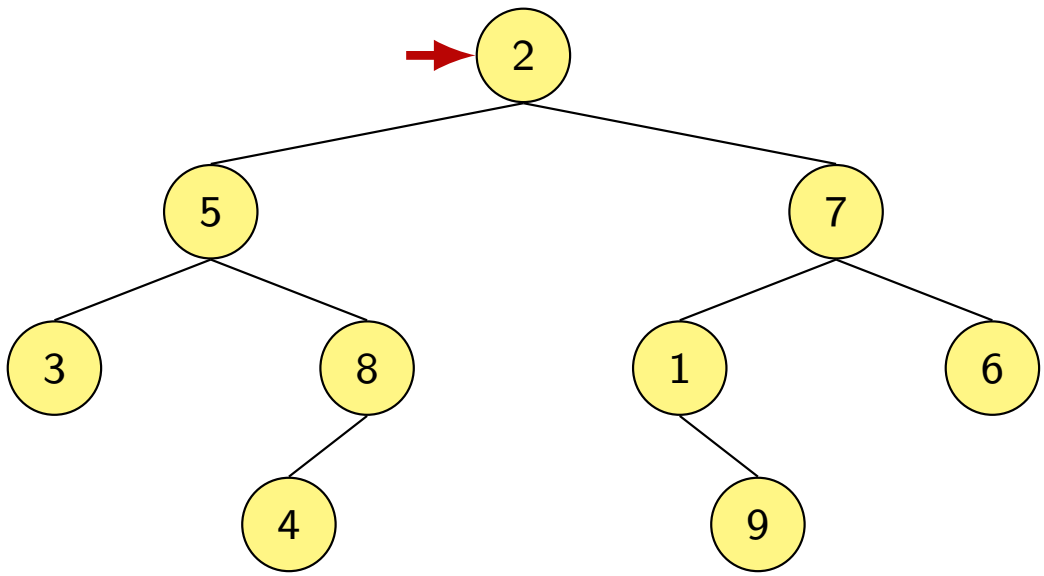


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

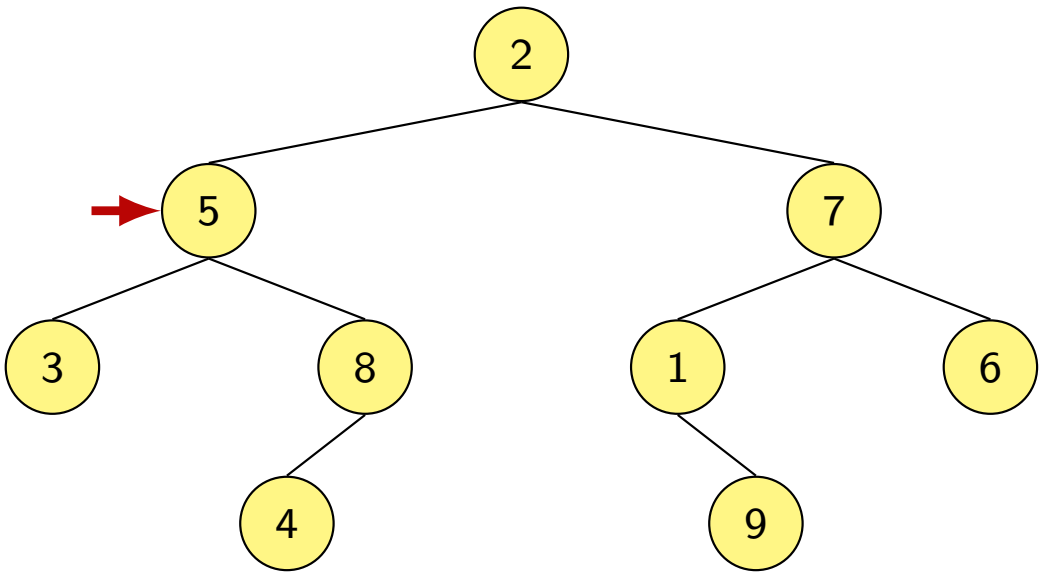


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

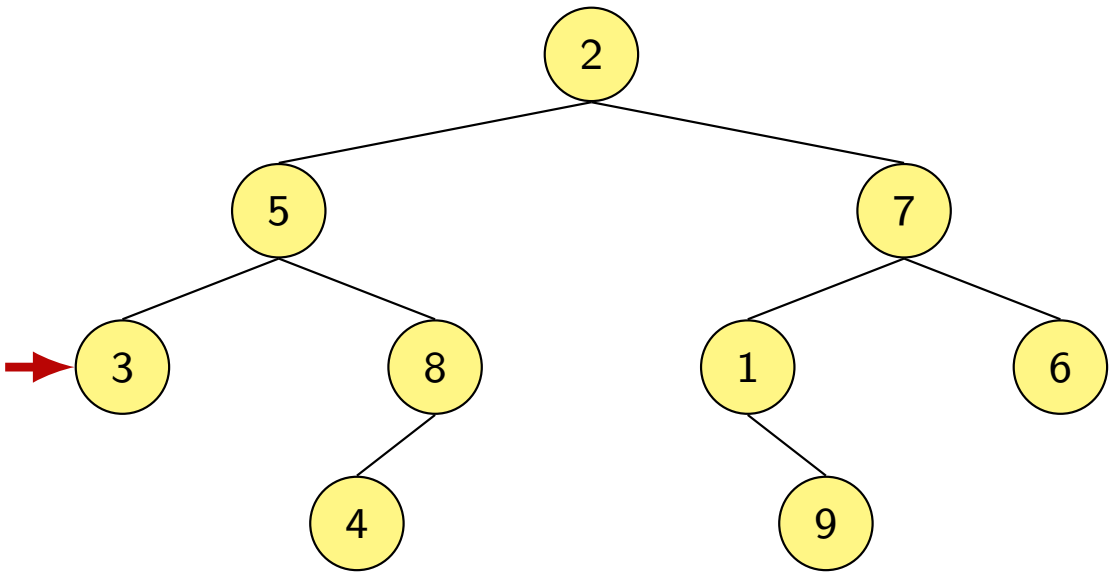


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

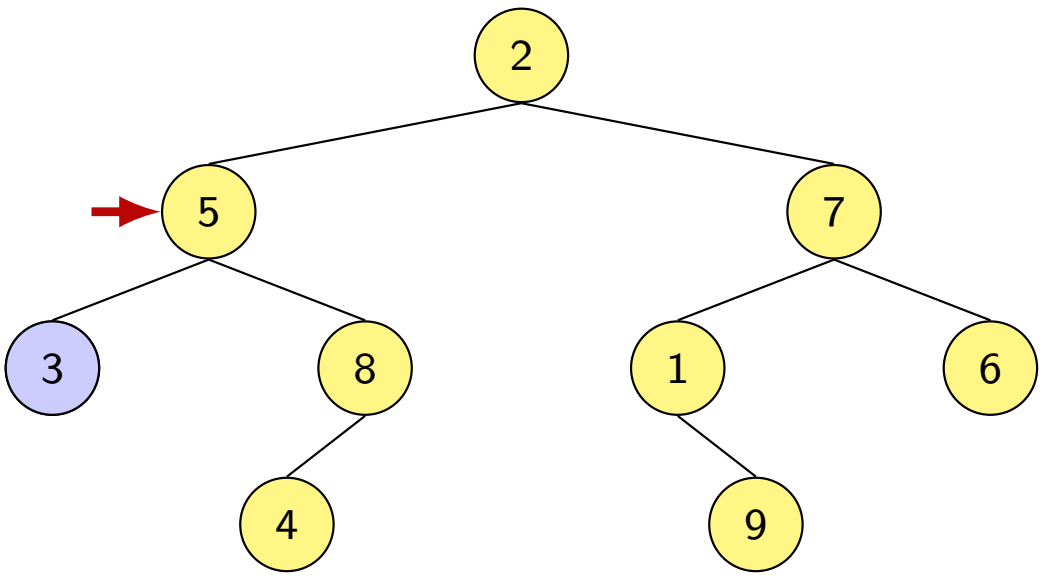


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

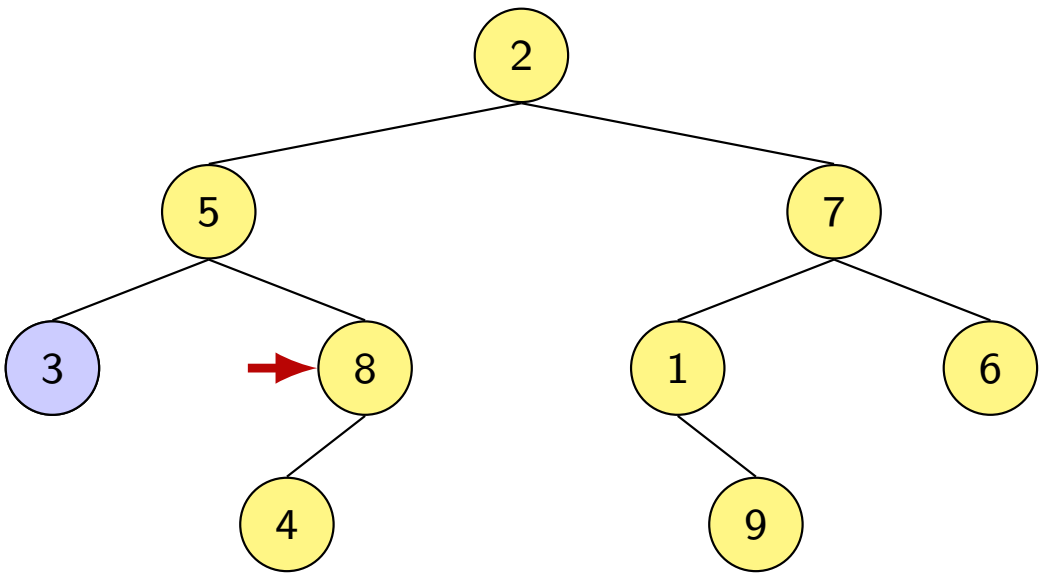


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

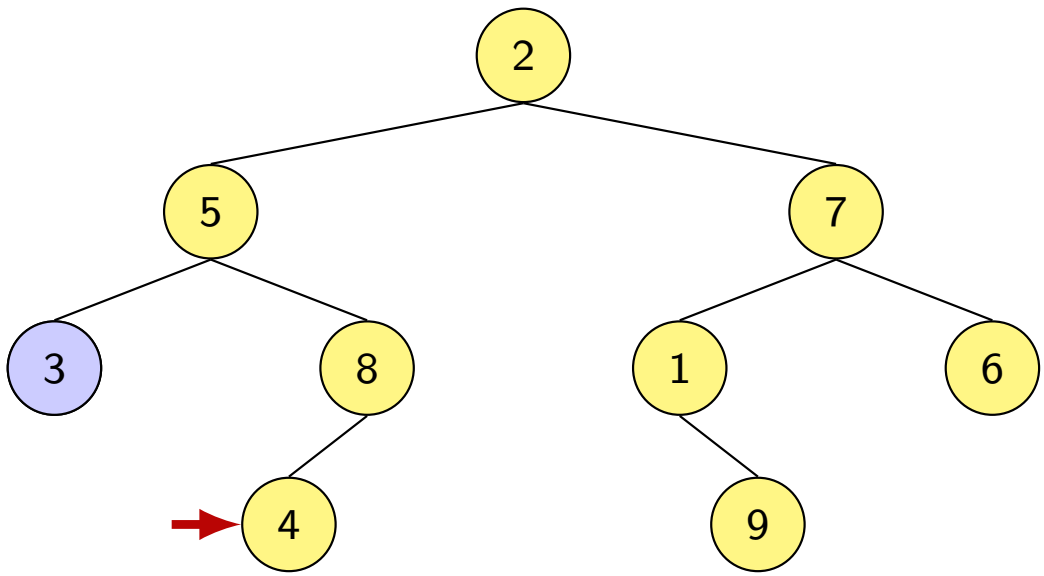


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

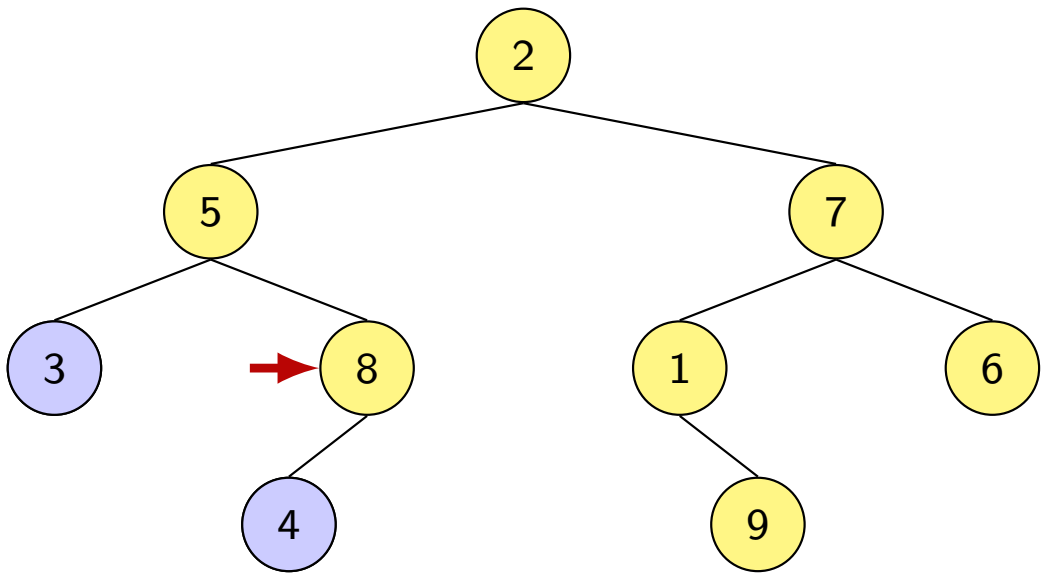


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

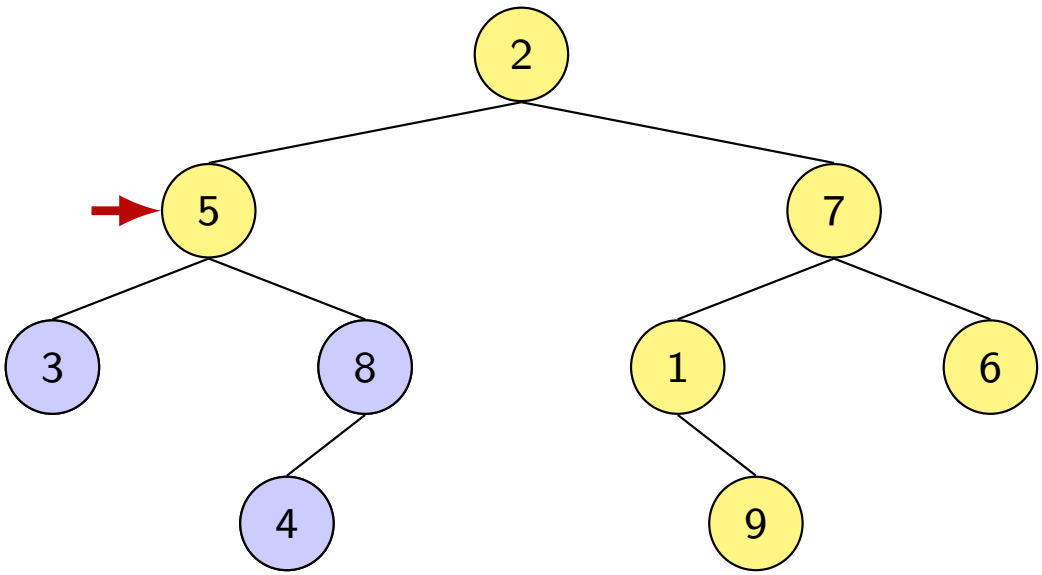


Ex: 3, 4,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

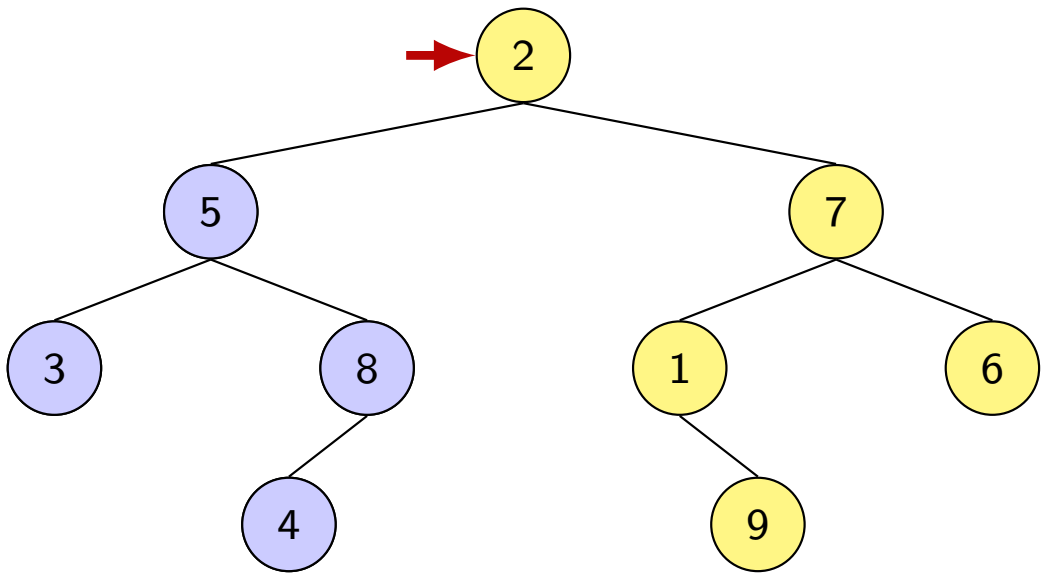


Ex: 3, 4, 8,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

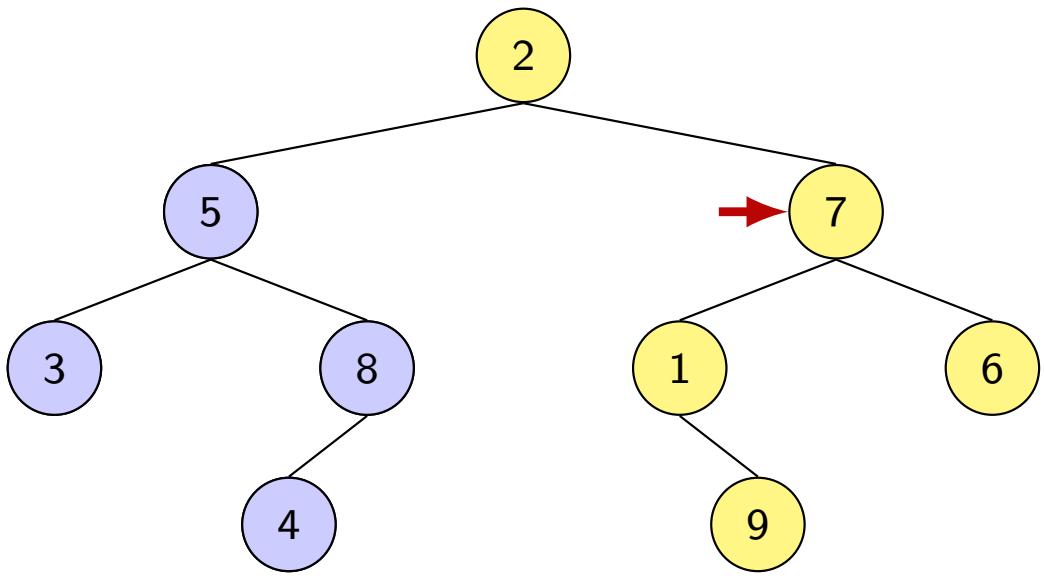


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

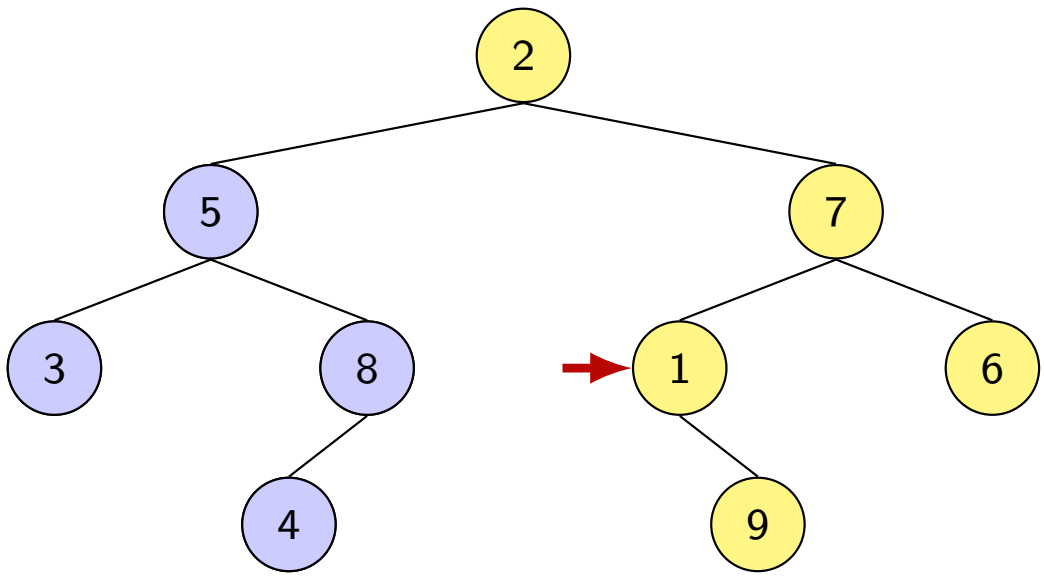


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

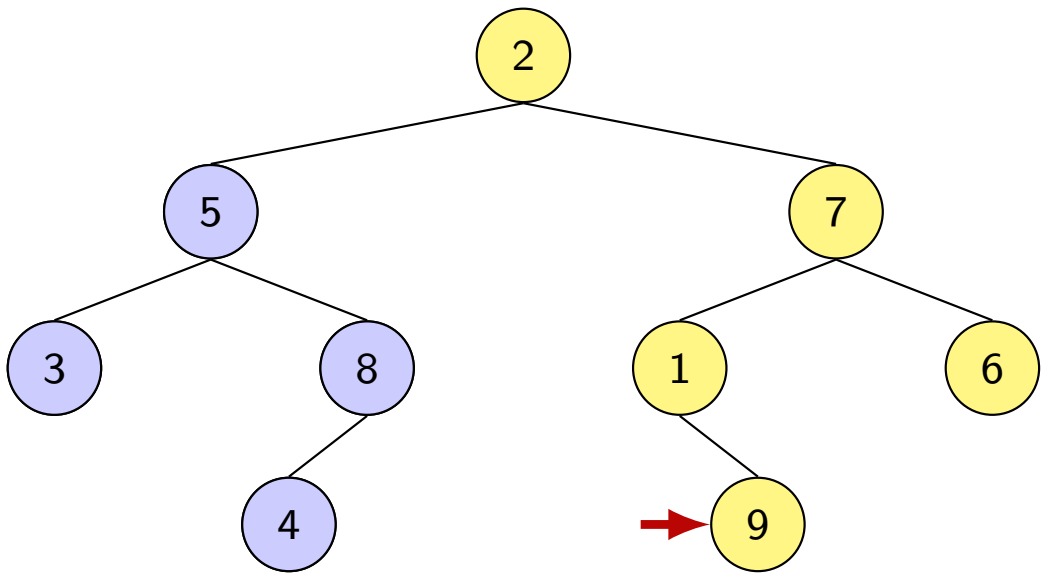


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

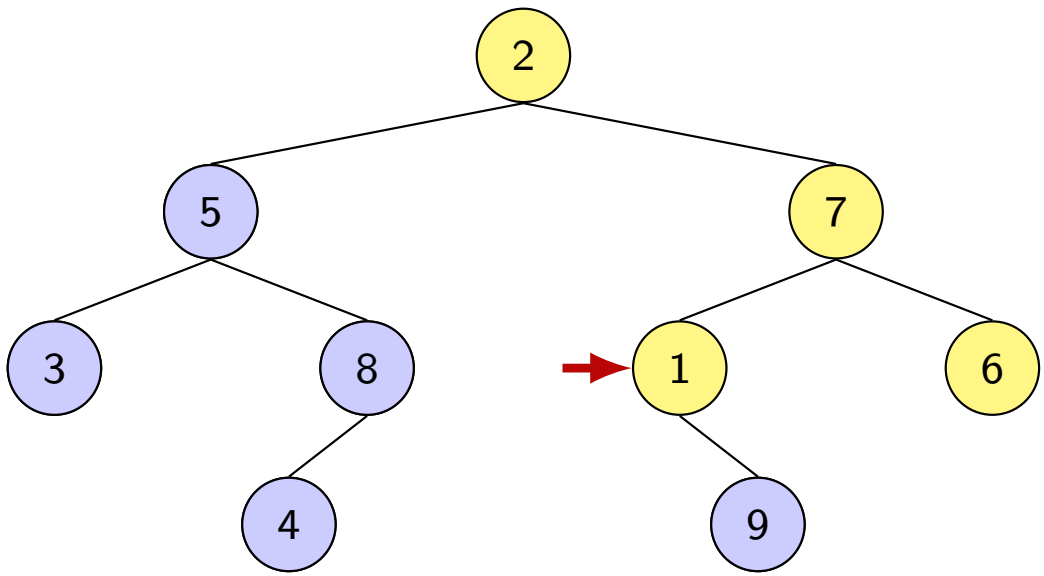


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

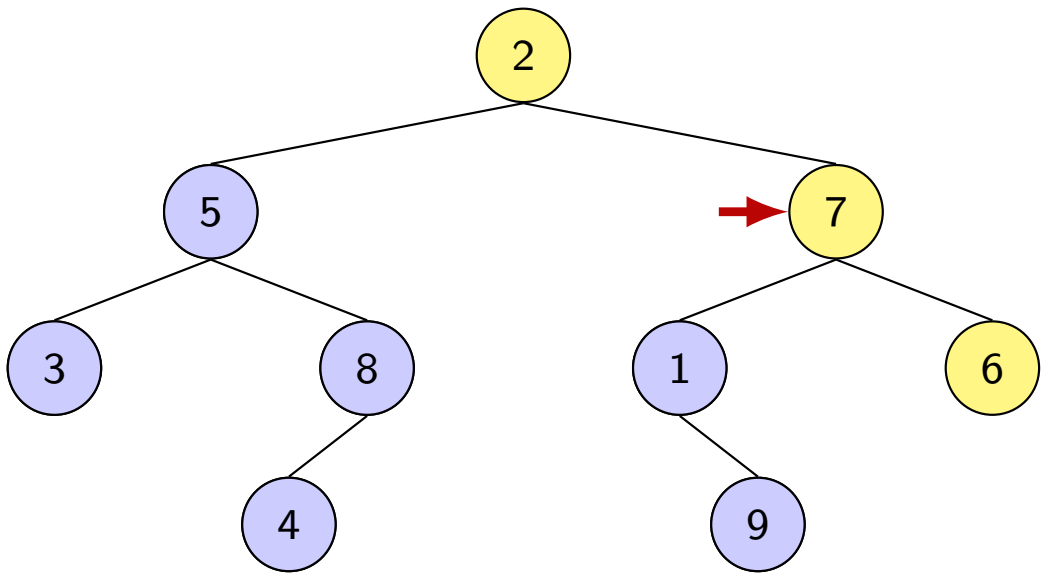


Ex: 3, 4, 8, 5, 9,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

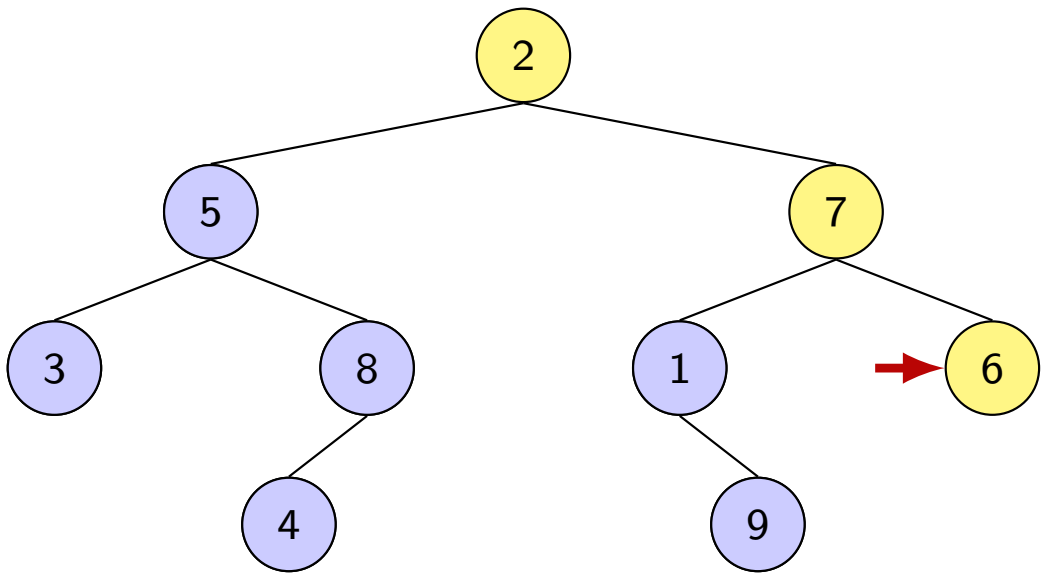


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

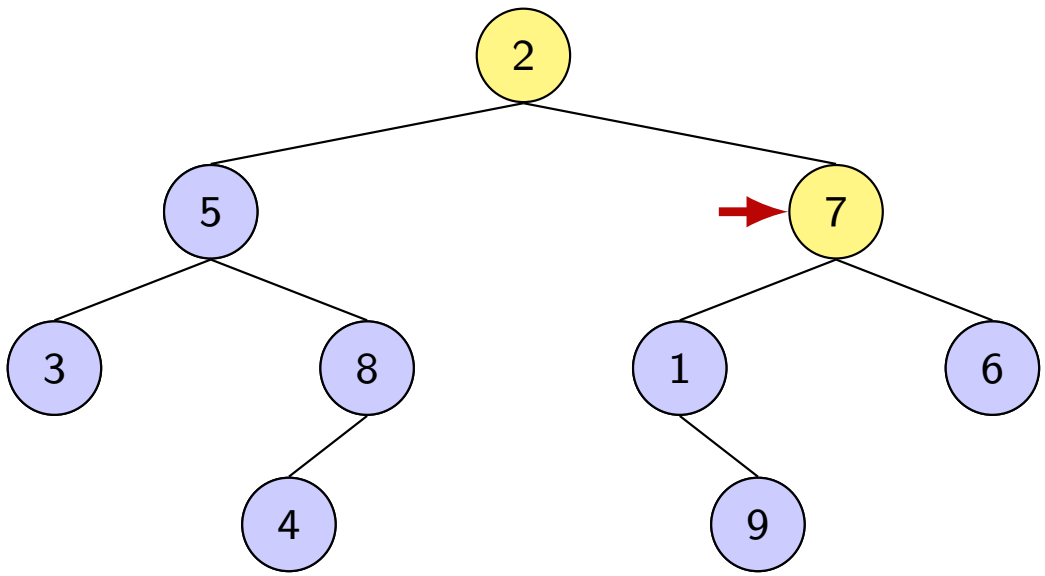


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

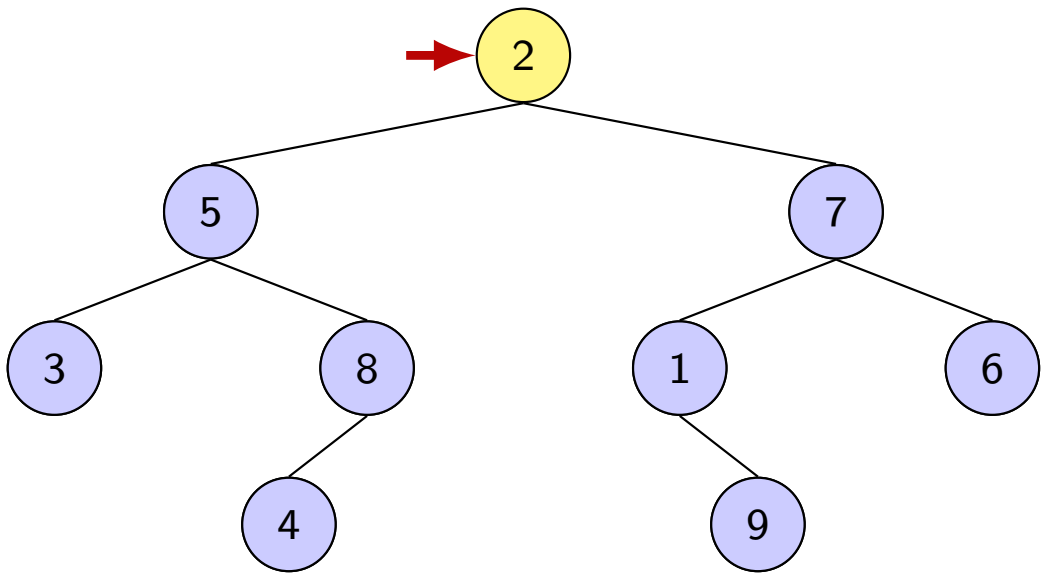


Ex: 3, 4, 8, 5, 9, 1, 6,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz

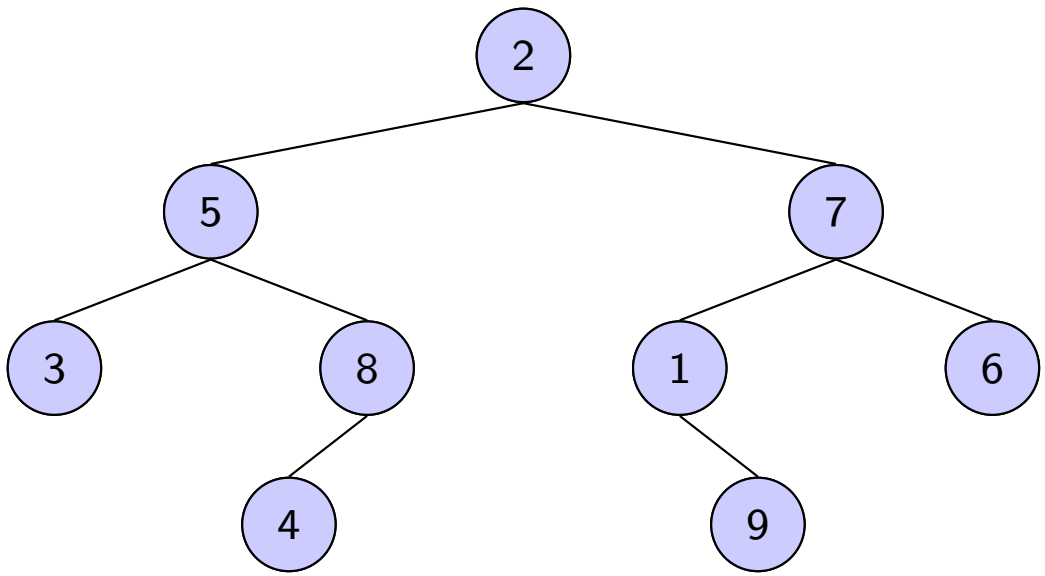


Ex: 3, 4, 8, 5, 9, 1, 6, 7,

Percorrendo os nós - Pós-ordem

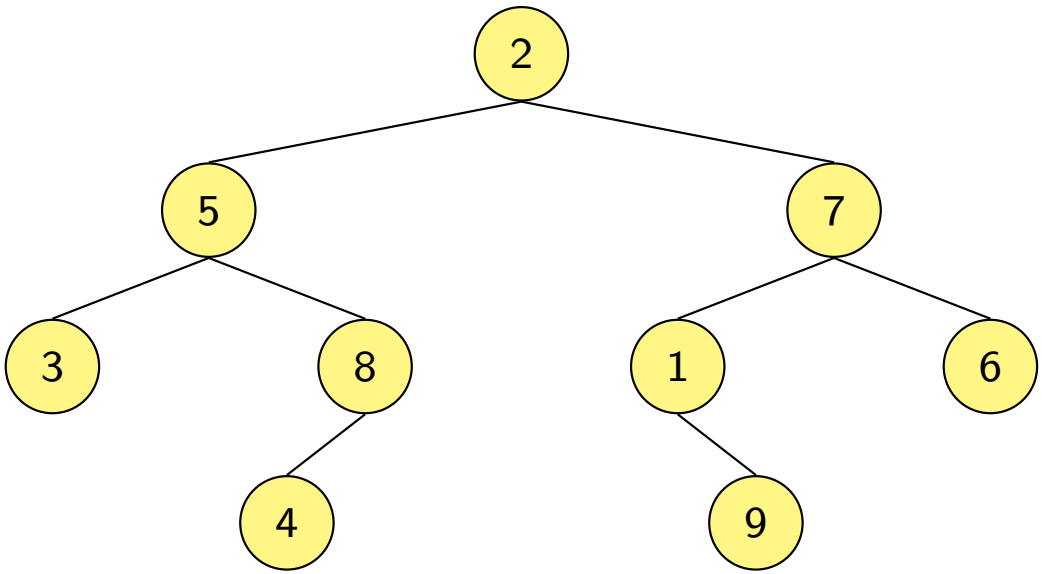
A pós-ordem

- primeiro acessa a subárvore esquerda
- depois acessa a subárvore direita
- e por último visita (processa) a raiz



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

Implementação de percurso Pós-ordem




```
71 void pos_ordem(No *raiz) {
72   if (raiz != NULL) {
73     pos_ordem(raiz->esq);
74     pos_ordem(raiz->dir);
75     printf("%d ", raiz->dado); /* visita raiz */
76   }
77 }
```

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

Percorrendo os nós - In-ordem

A in-ordem

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda

Percorrendo os nós - In-ordem

A in-ordem

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

Percorrendo os nós - In-ordem

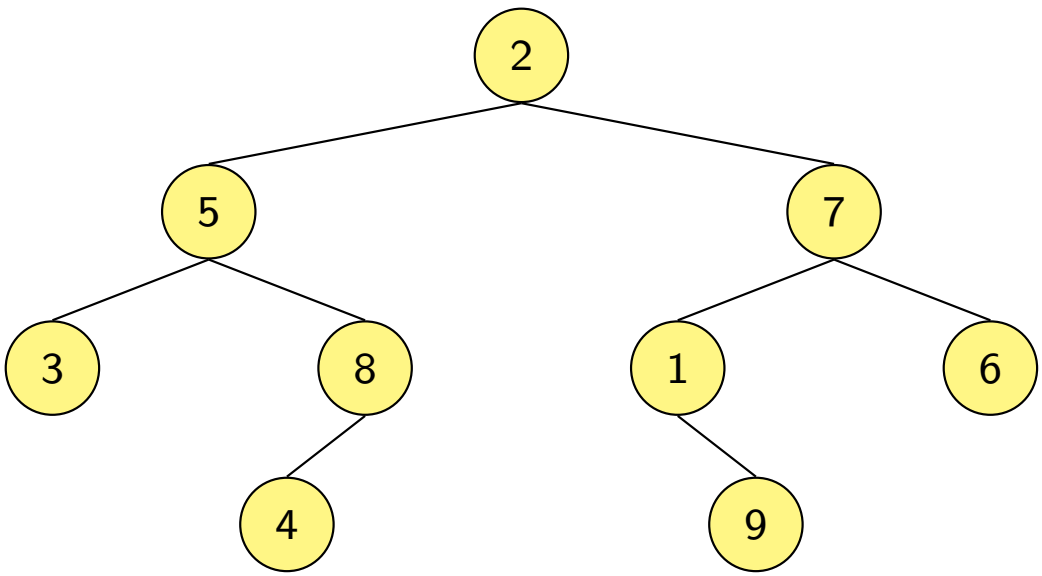
A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

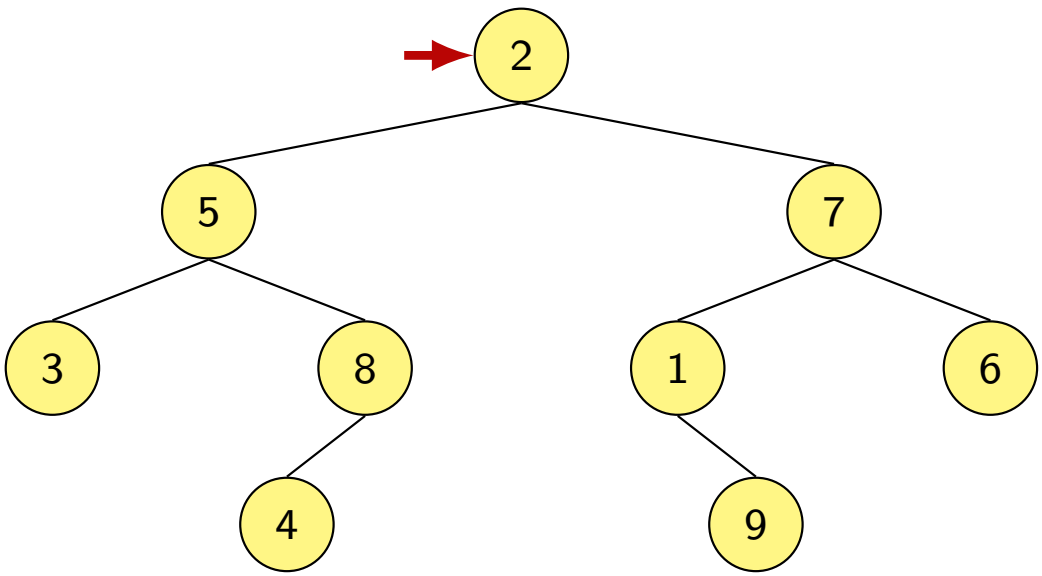


Ex:

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

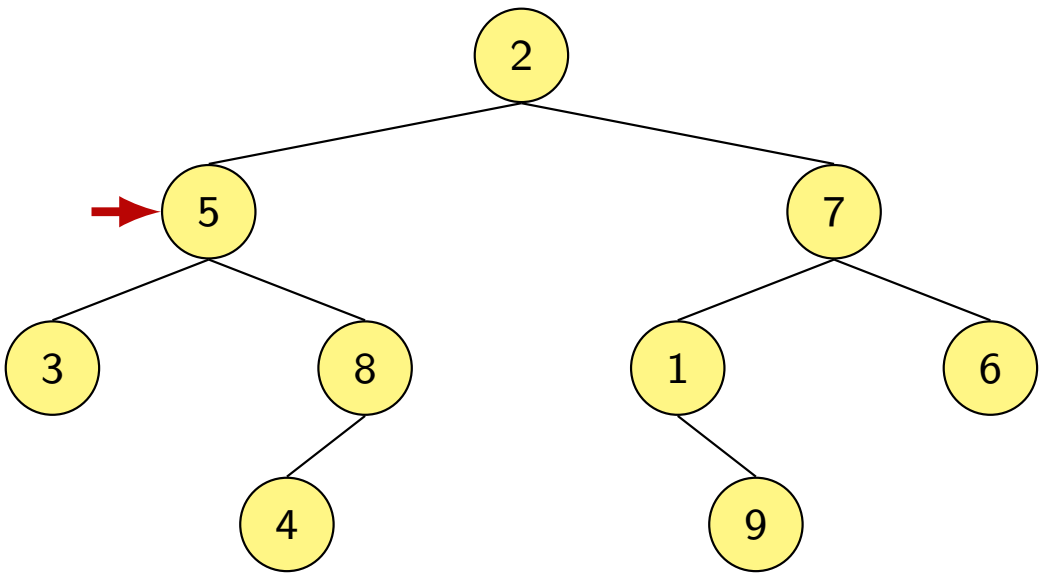


Ex:

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

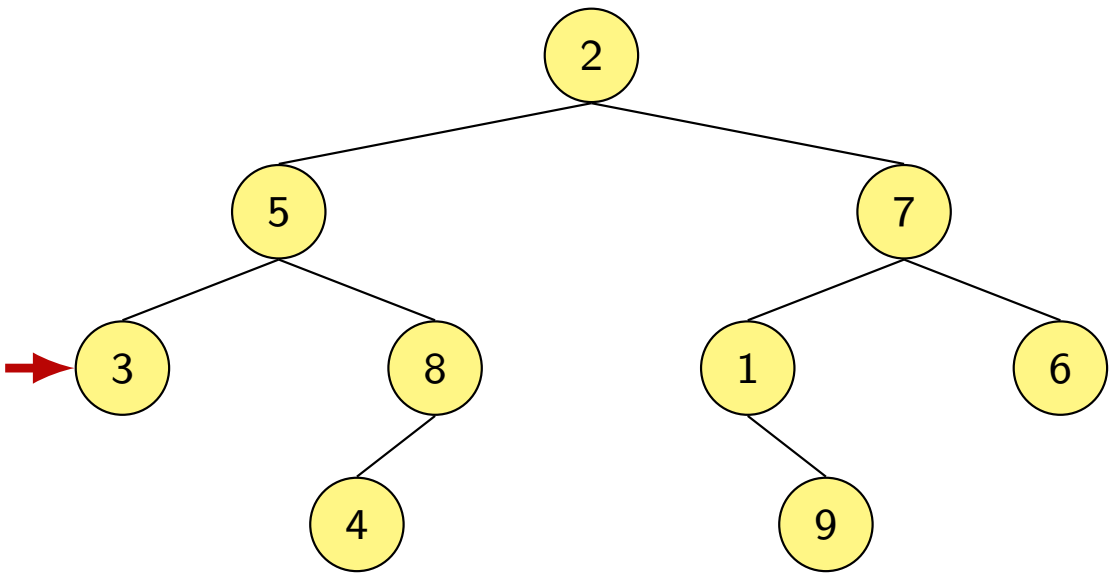


Ex:

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

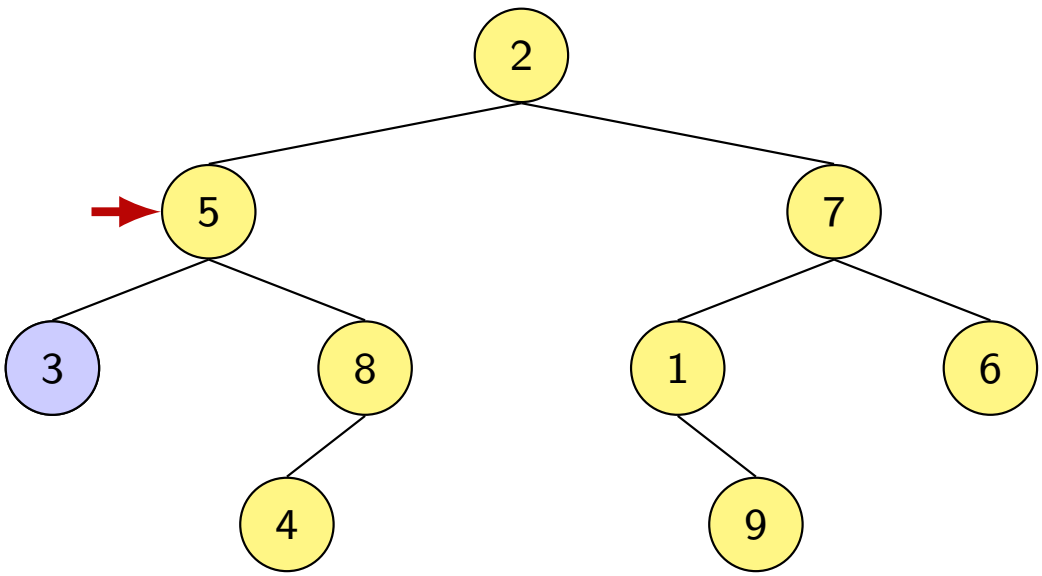


Ex:

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

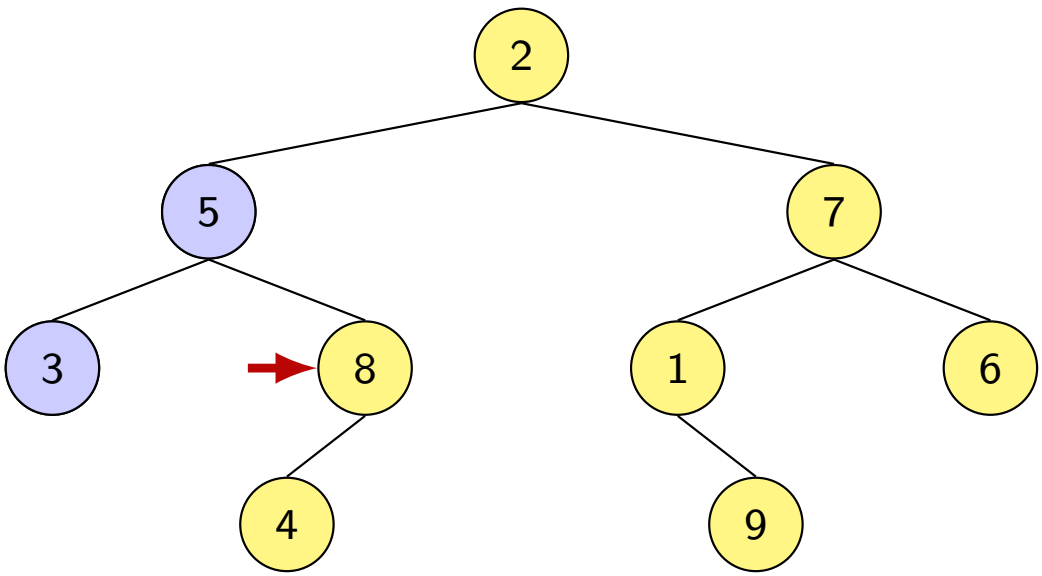


Ex: 3,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

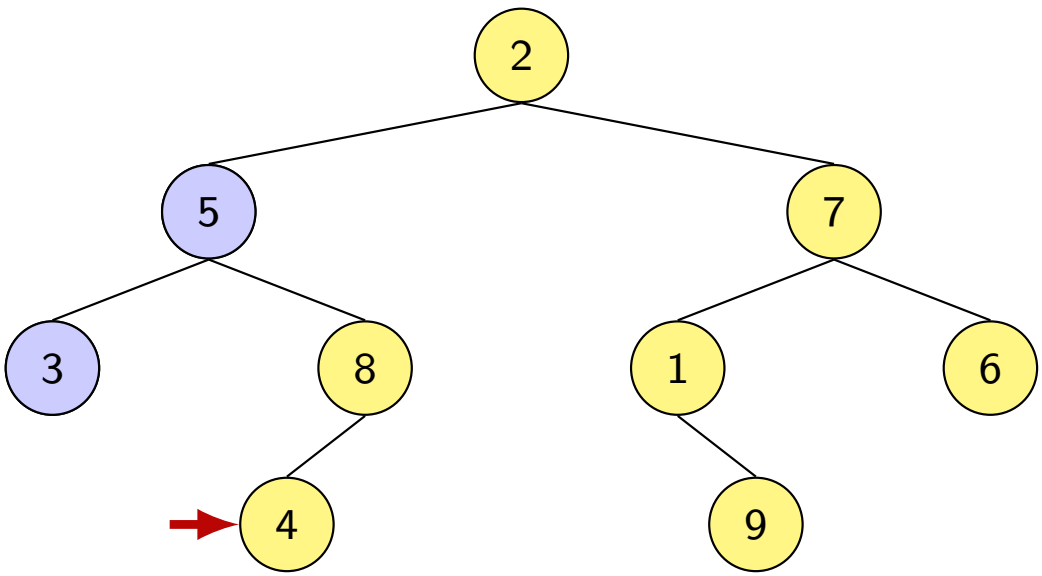


Ex: 3, 5,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

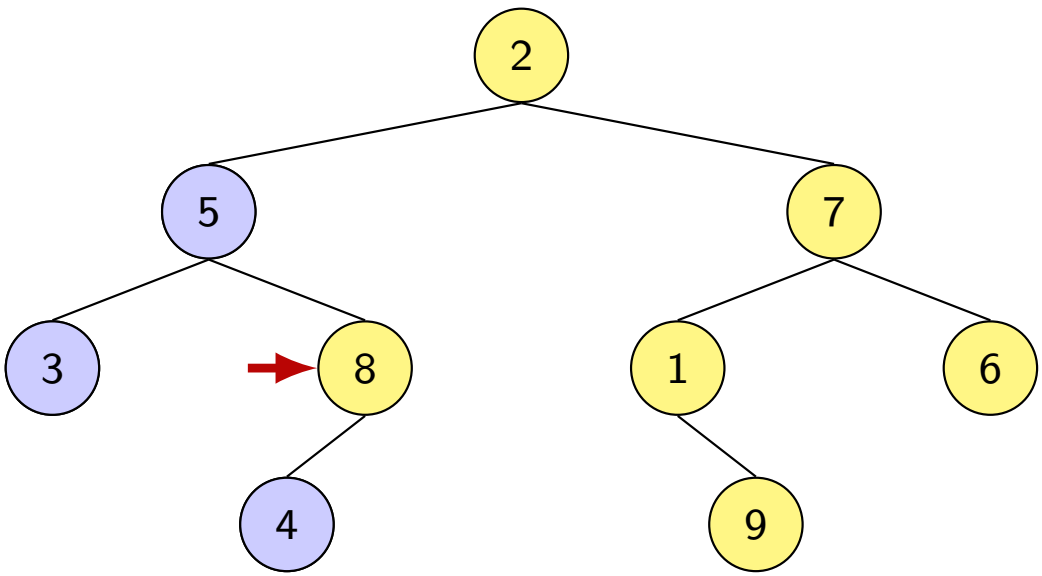


Ex: 3, 5,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

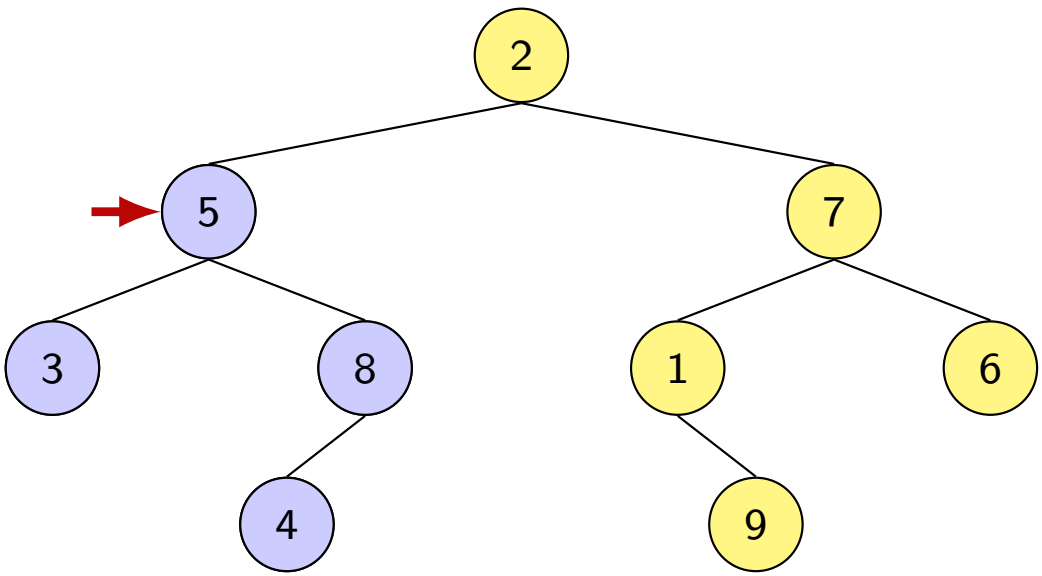


Ex: 3, 5, 4,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

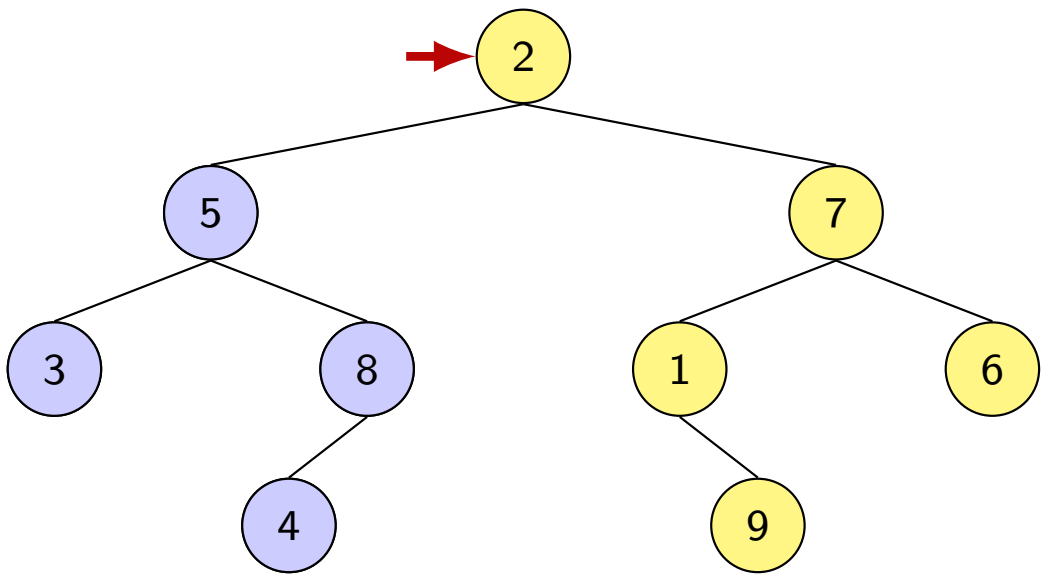


Ex: 3, 5, 4, 8,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

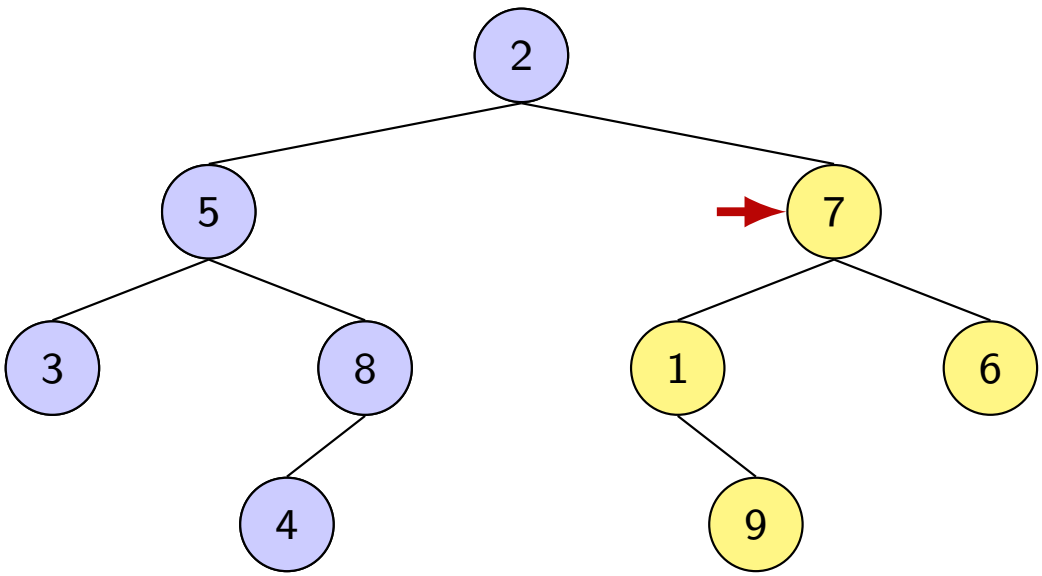


Ex: 3, 5, 4, 8,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

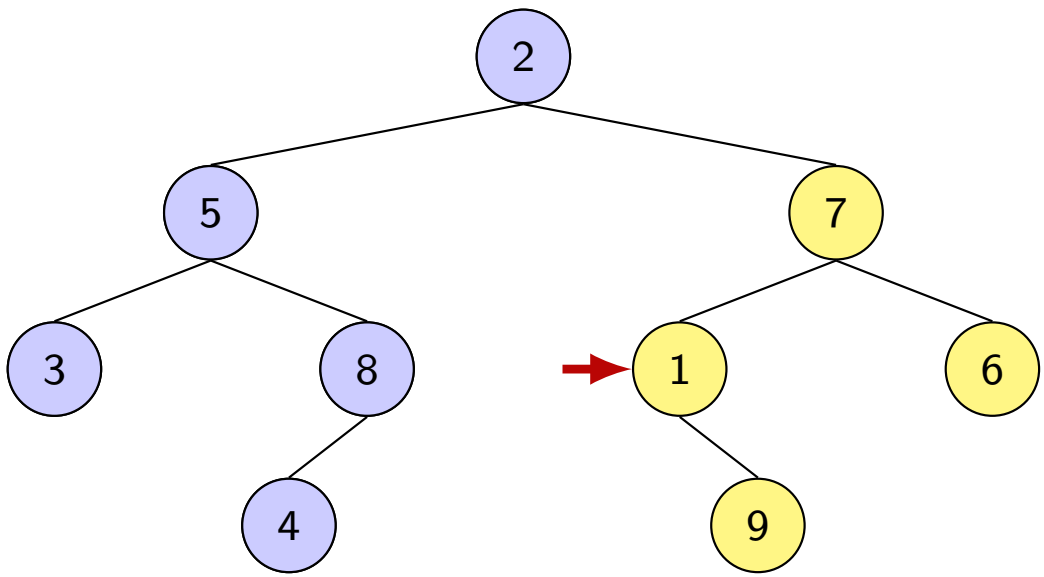


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

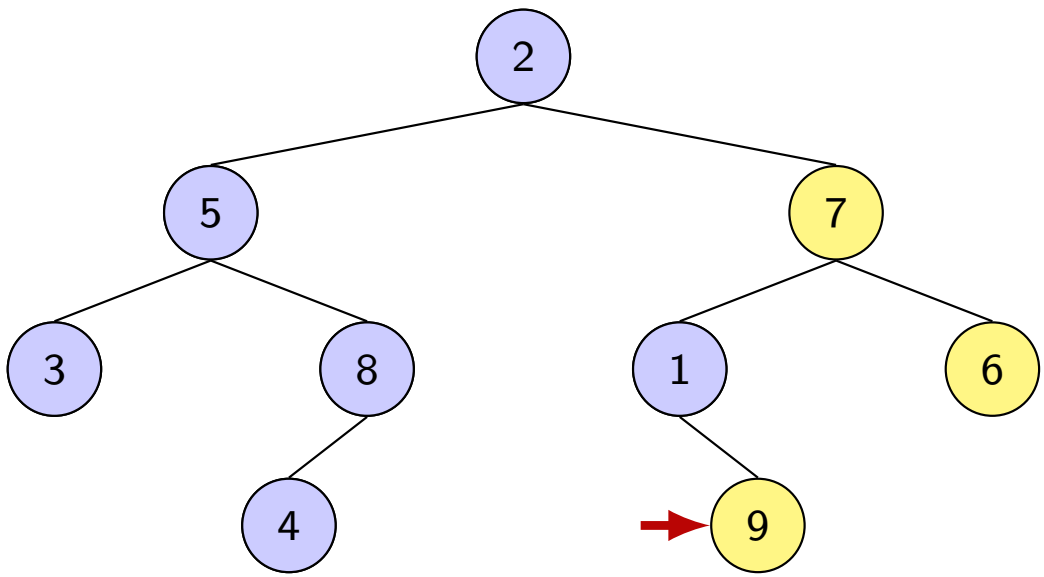


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

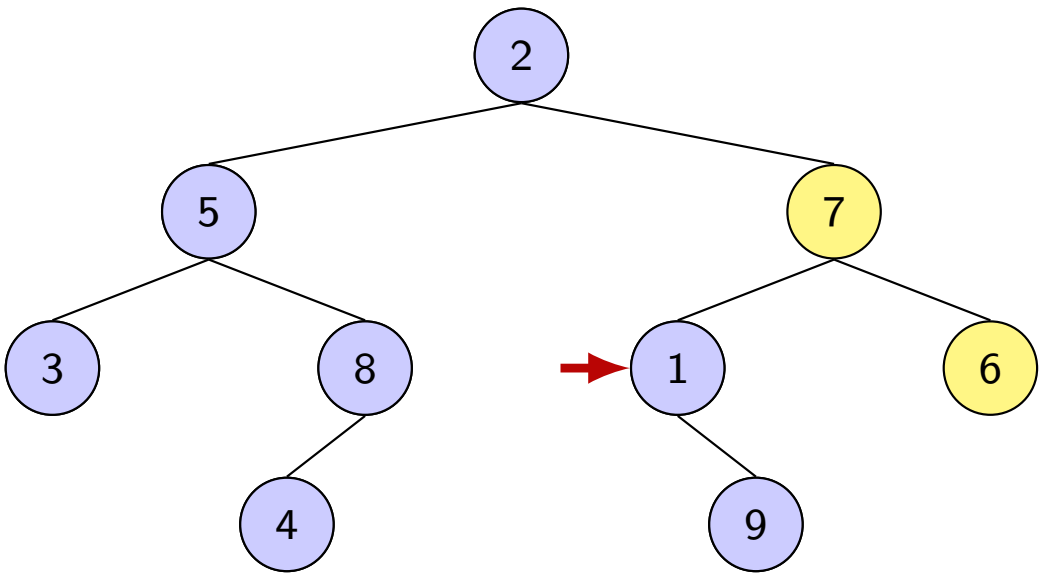


Ex: 3, 5, 4, 8, 2, 1,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

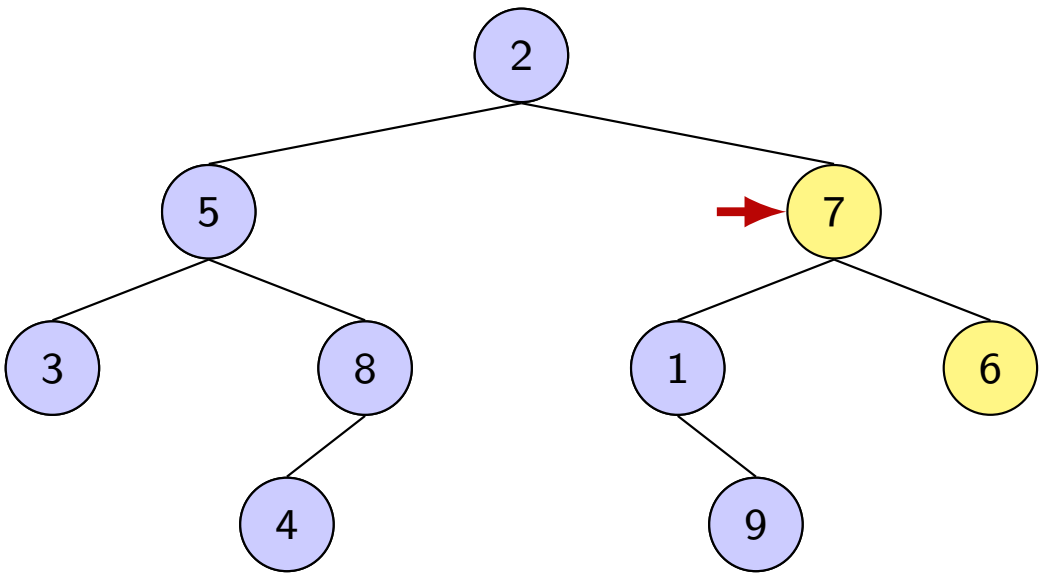


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

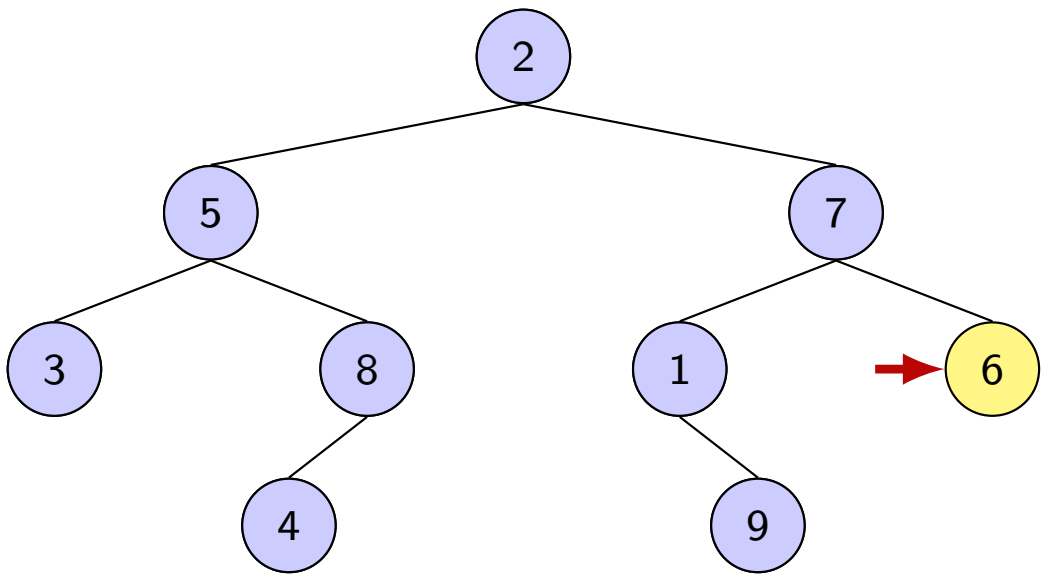


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

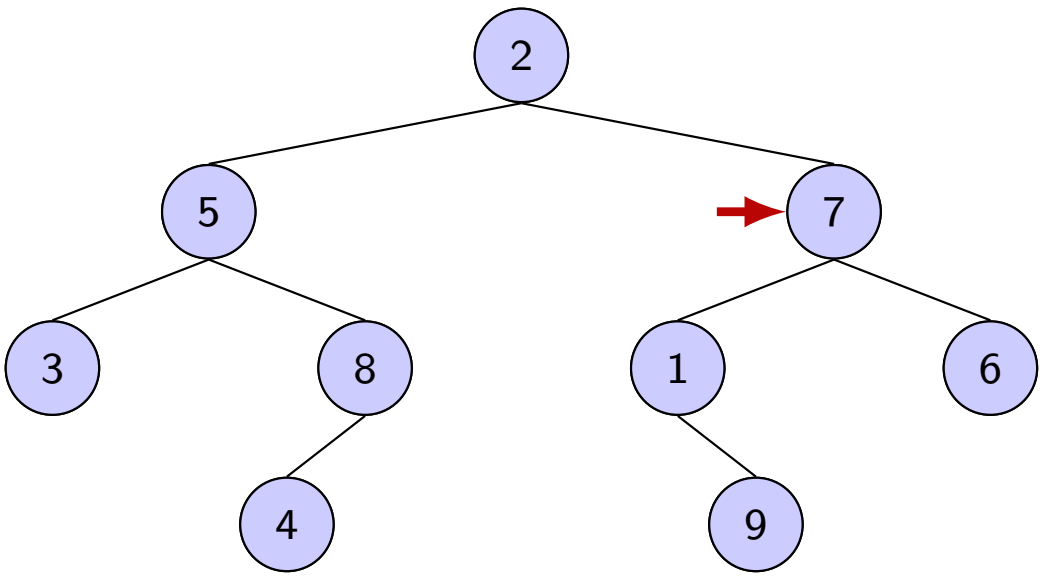


Ex: 3, 5, 4, 8, 2, 1, 9, 7,

Percorrendo os nós - In-ordem

A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita

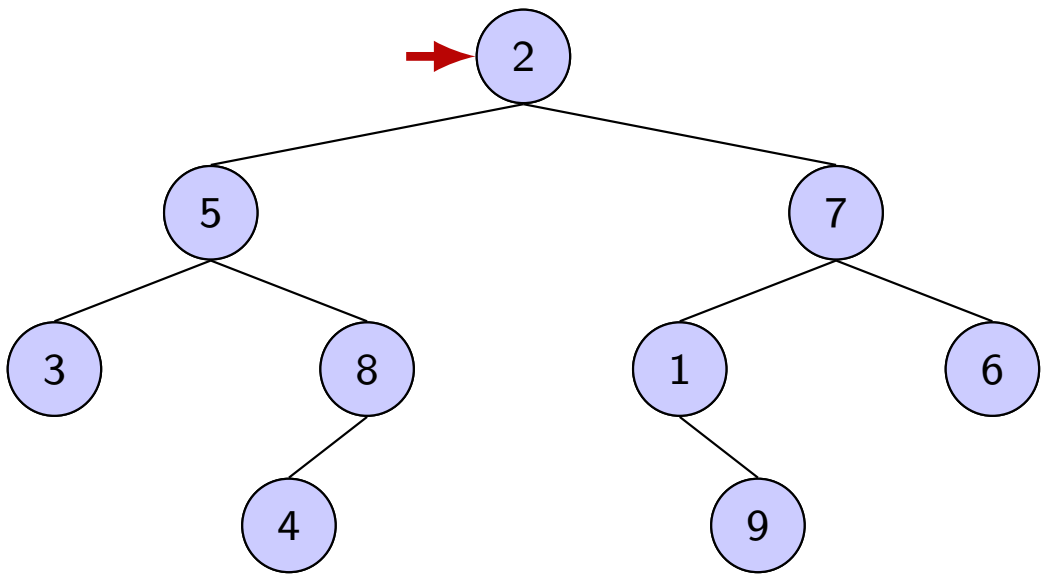


Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

Percorrendo os nós - In-ordem

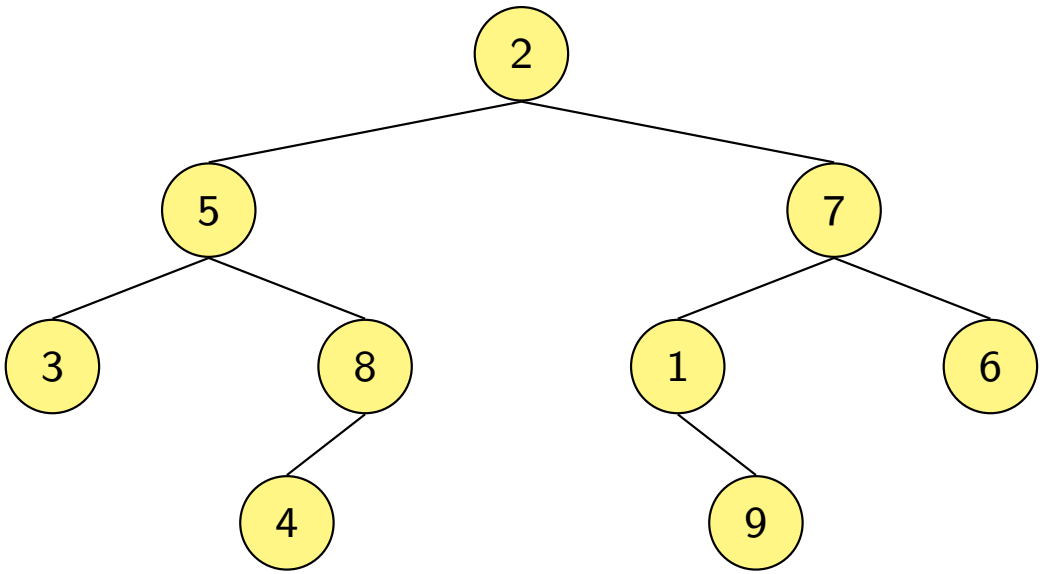
A in-ordem

- primeiro acessa a subárvore esquerda
- depois visita (processa) a raiz
- e por última acessa a subárvore direita



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

Implementação de percurso In-ordem




```
79 void inordem(No *raiz) {
80   if (raiz != NULL) {
81     inordem(raiz->esq);
82     printf("%d ", raiz->dado); /* visita raiz */
83     inordem(raiz->dir);
84   }
85 }
```

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

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

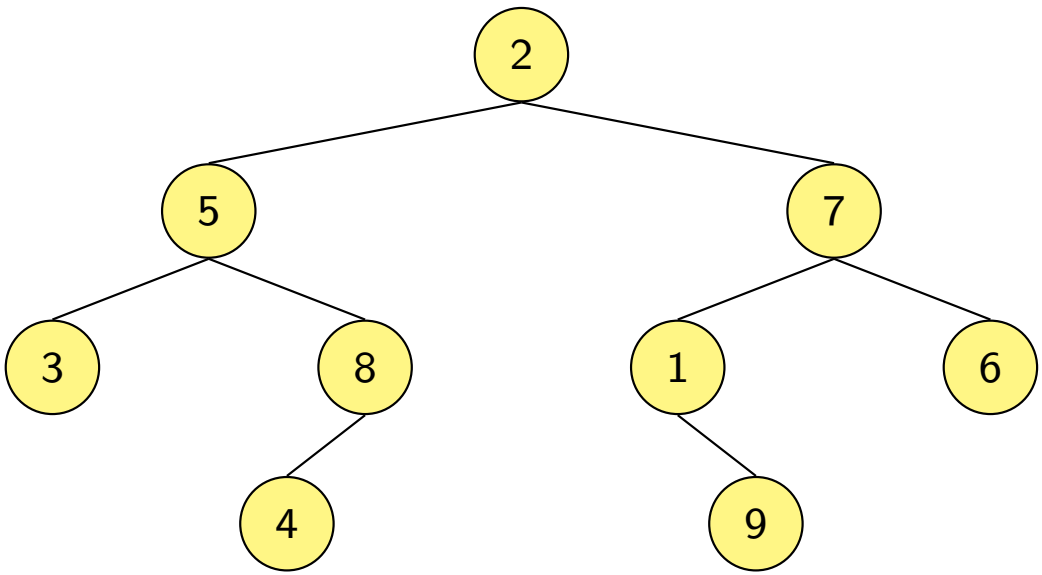
O percurso em largura

- 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

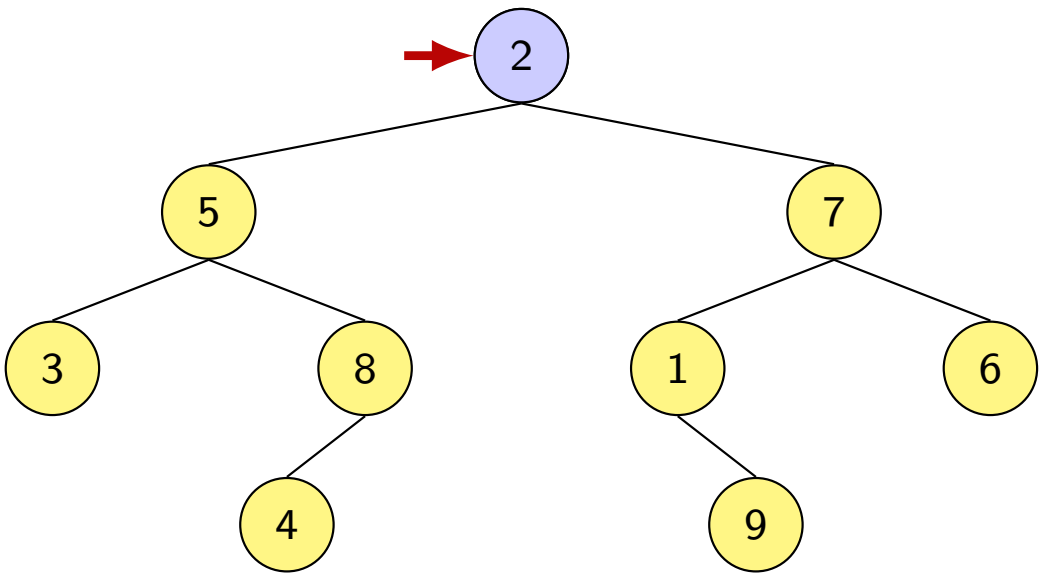


Ex:

Percorrendo os nós - em largura

O percurso em largura

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

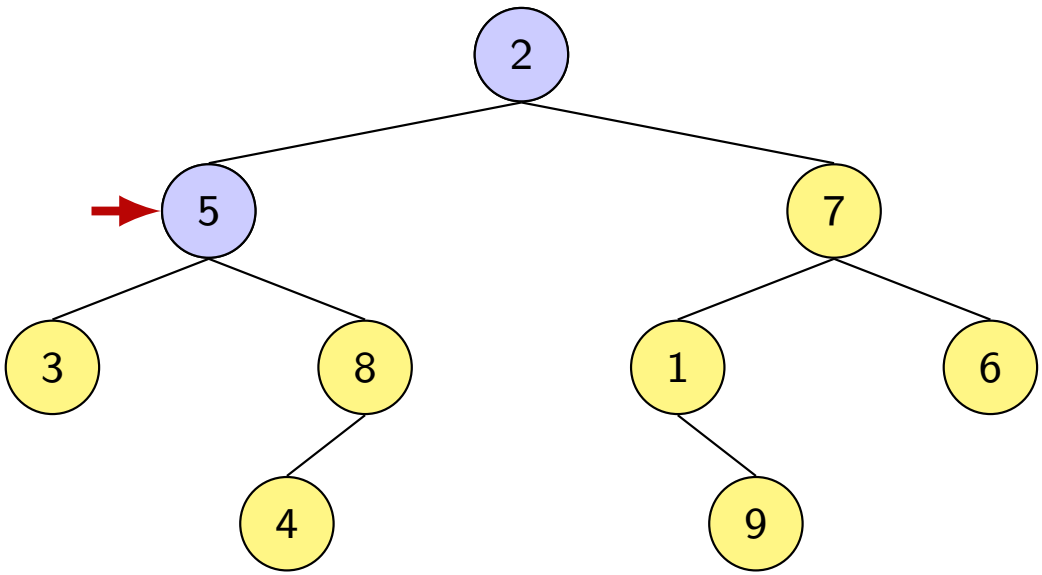


Ex: 2,

Percorrendo os nós - em largura

O percurso em largura

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

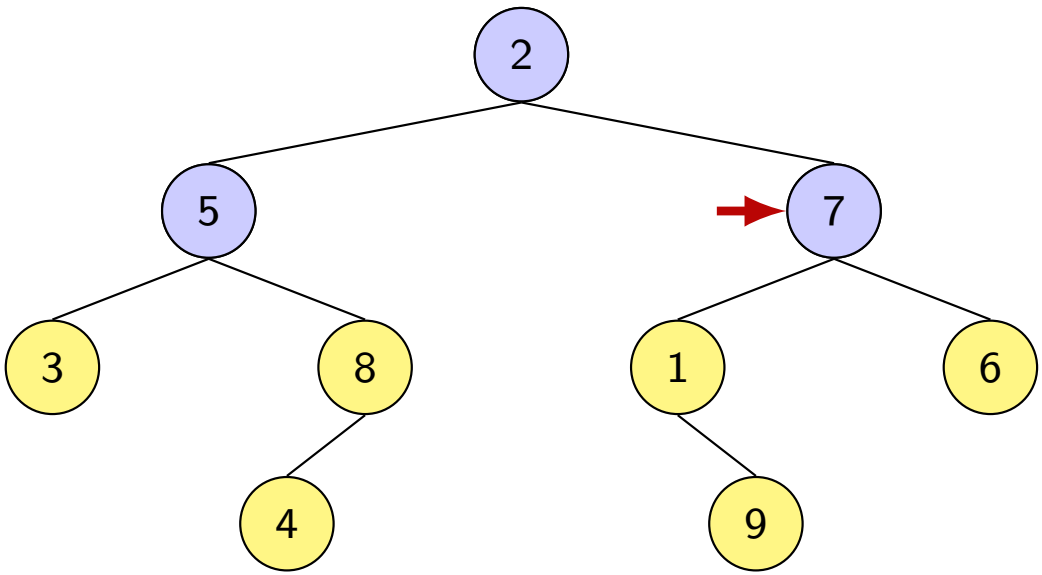


Ex: 2, 5,

Percorrendo os nós - em largura

O percurso em largura

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

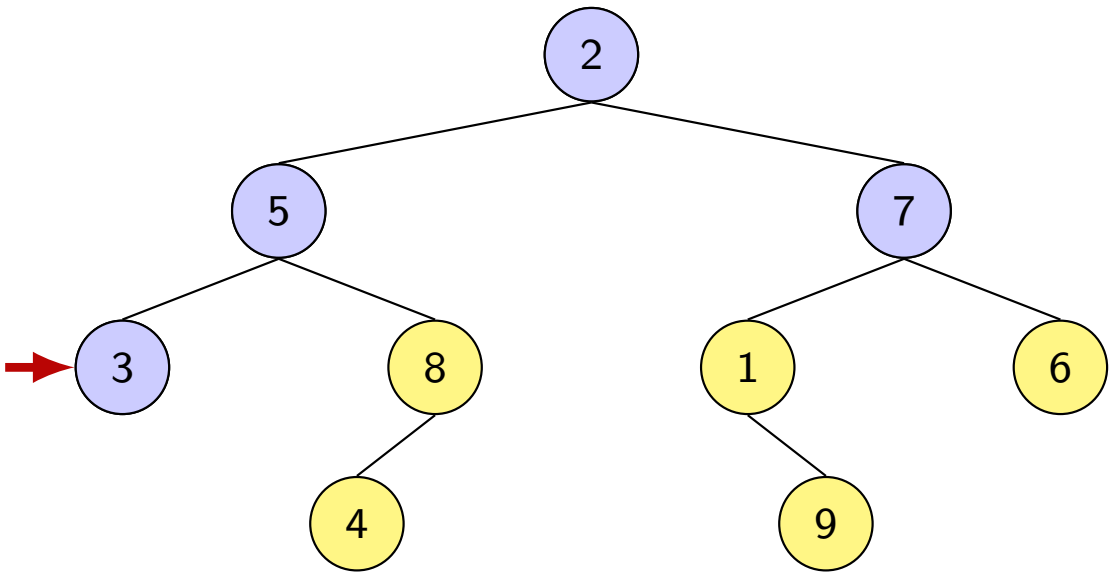


Ex: 2, 5, 7,

Percorrendo os nós - em largura

O percurso em largura

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

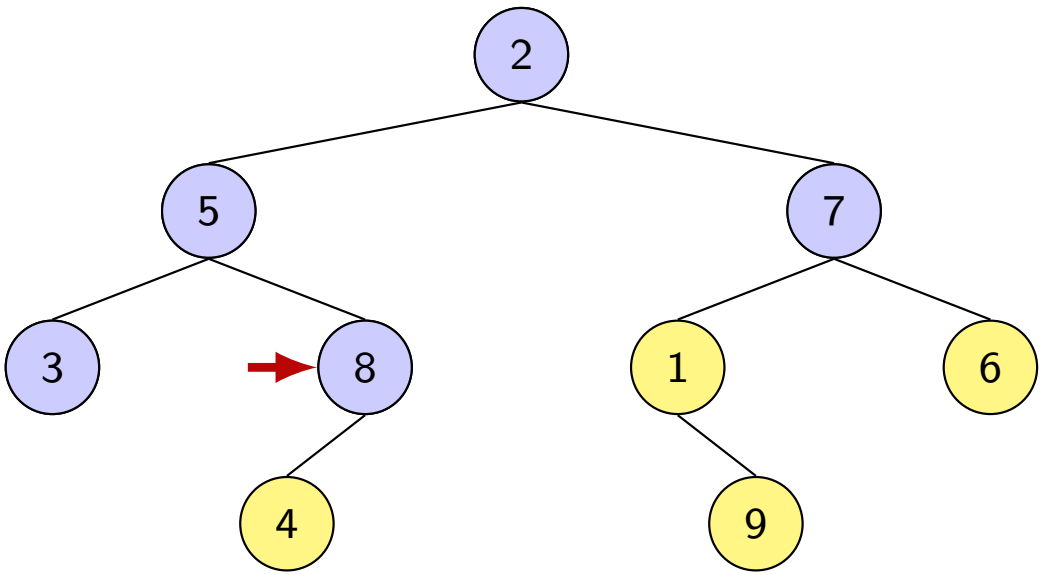


Ex: 2, 5, 7, 3,

Percorrendo os nós - em largura

O percurso em largura

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

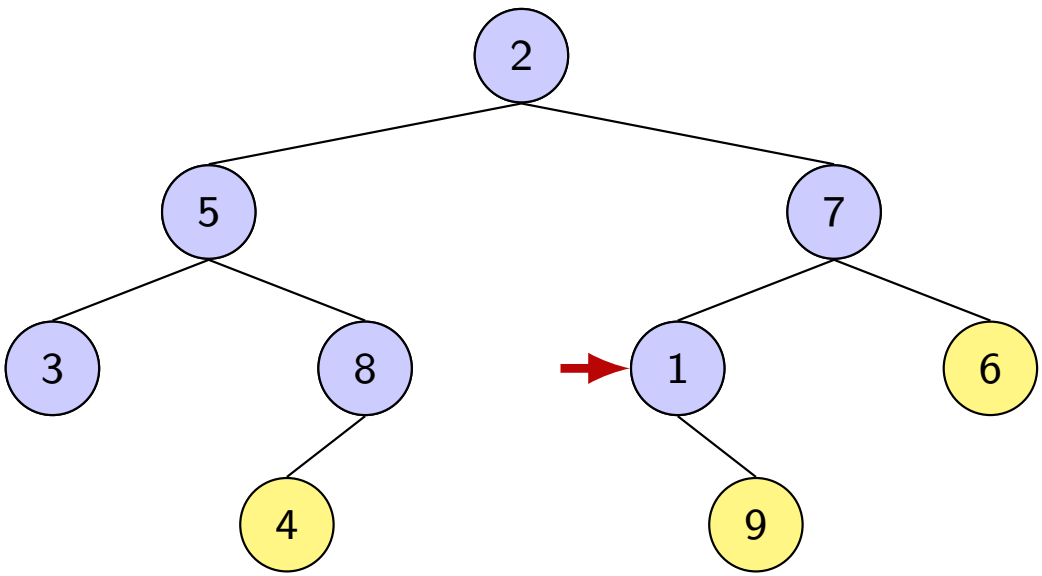


Ex: 2, 5, 7, 3, 8,

Percorrendo os nós - em largura

O percurso em largura

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

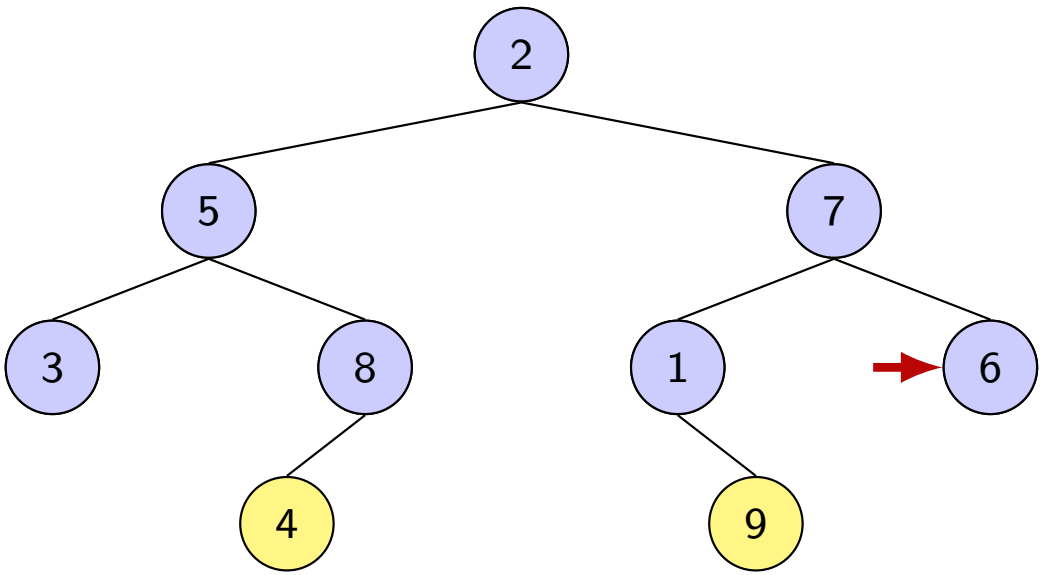


Ex: 2, 5, 7, 3, 8, 1,

Percorrendo os nós - em largura

O percurso em largura

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

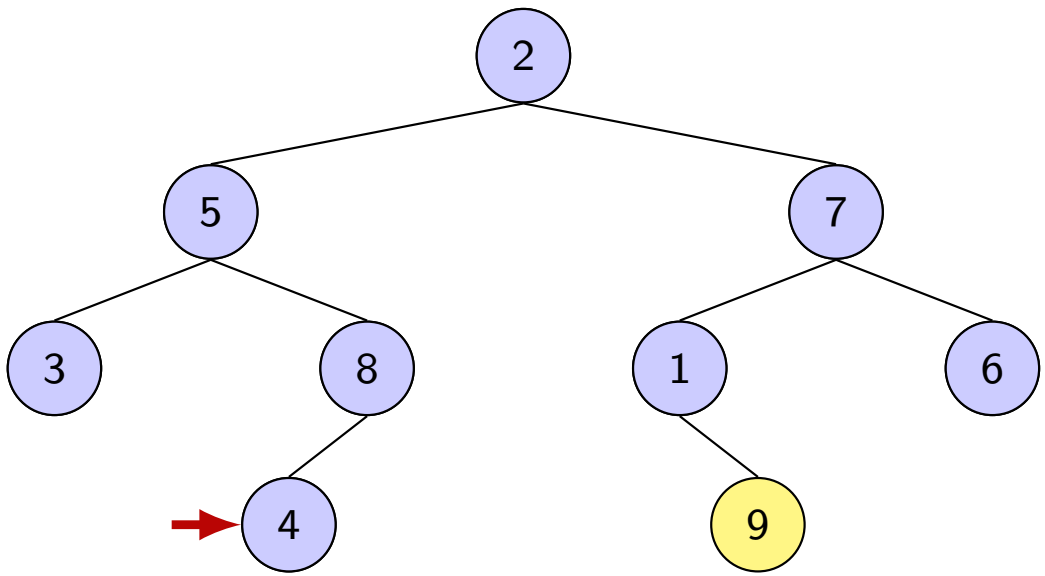


Ex: 2, 5, 7, 3, 8, 1, 6,

Percorrendo os nós - em largura

O percurso em largura

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

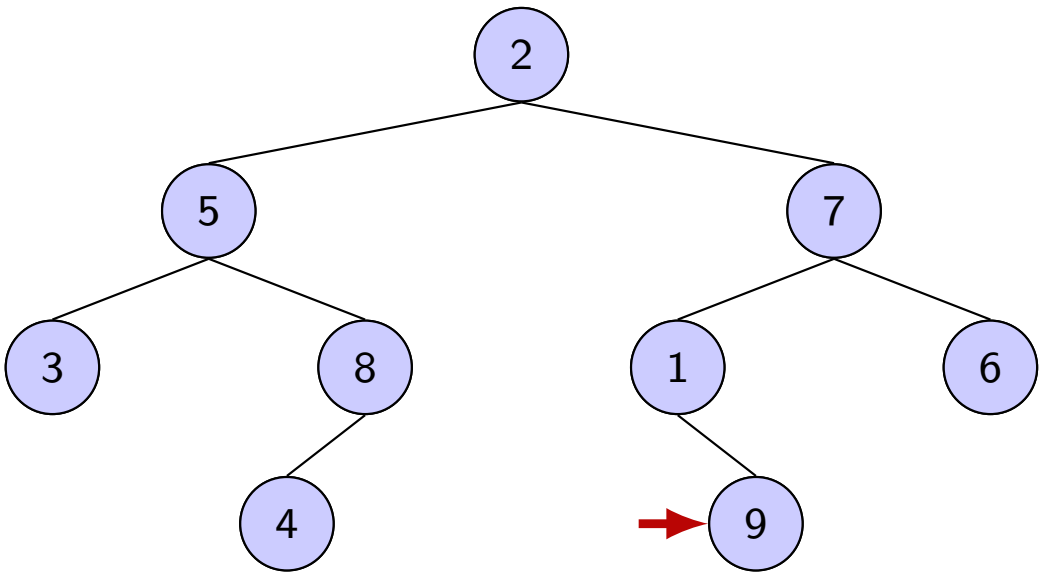


Ex: 2, 5, 7, 3, 8, 1, 6, 4,

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

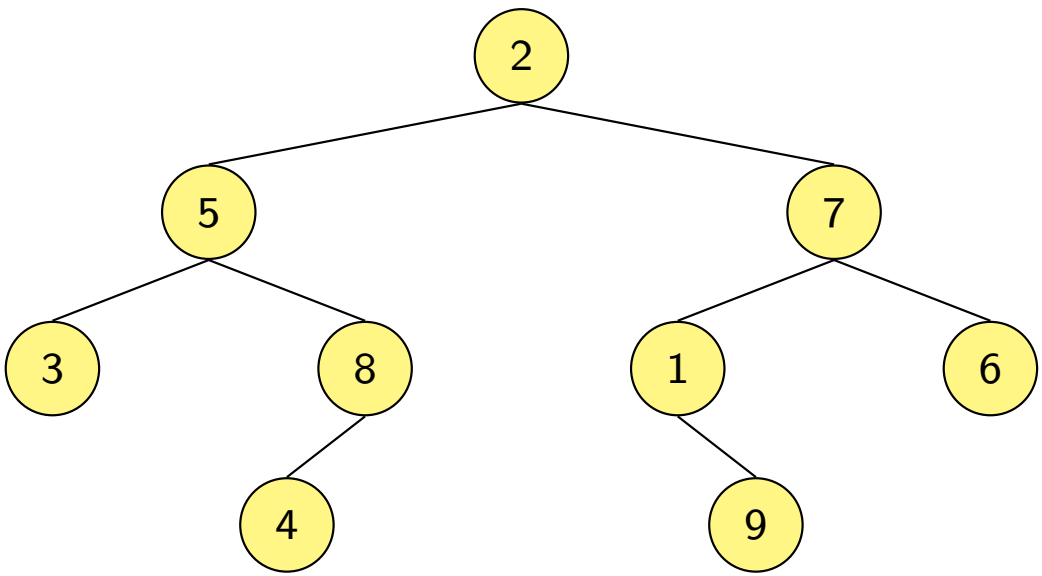
Como implementar o percurso em largura?

- 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

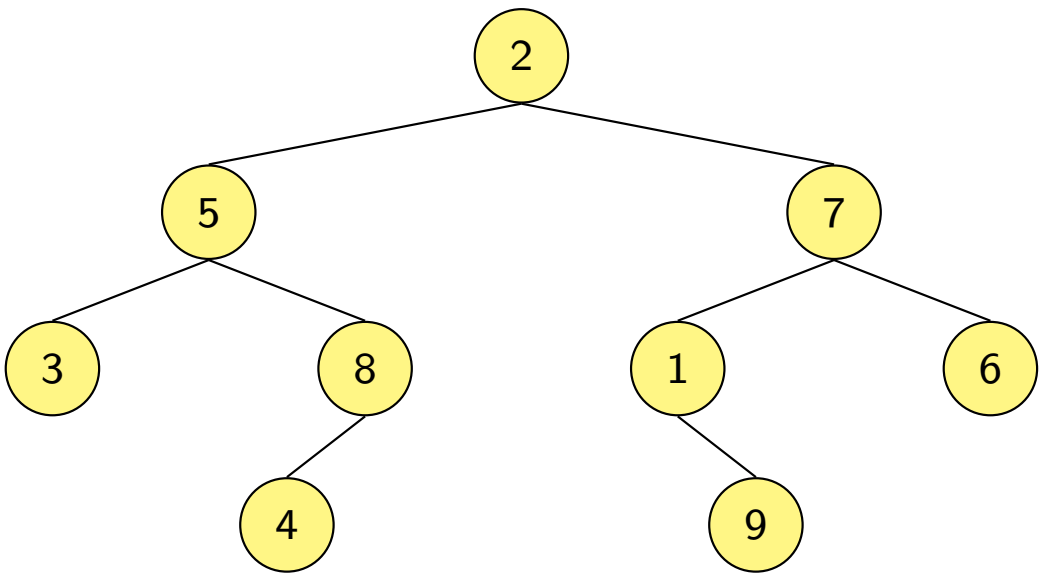


Fila

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

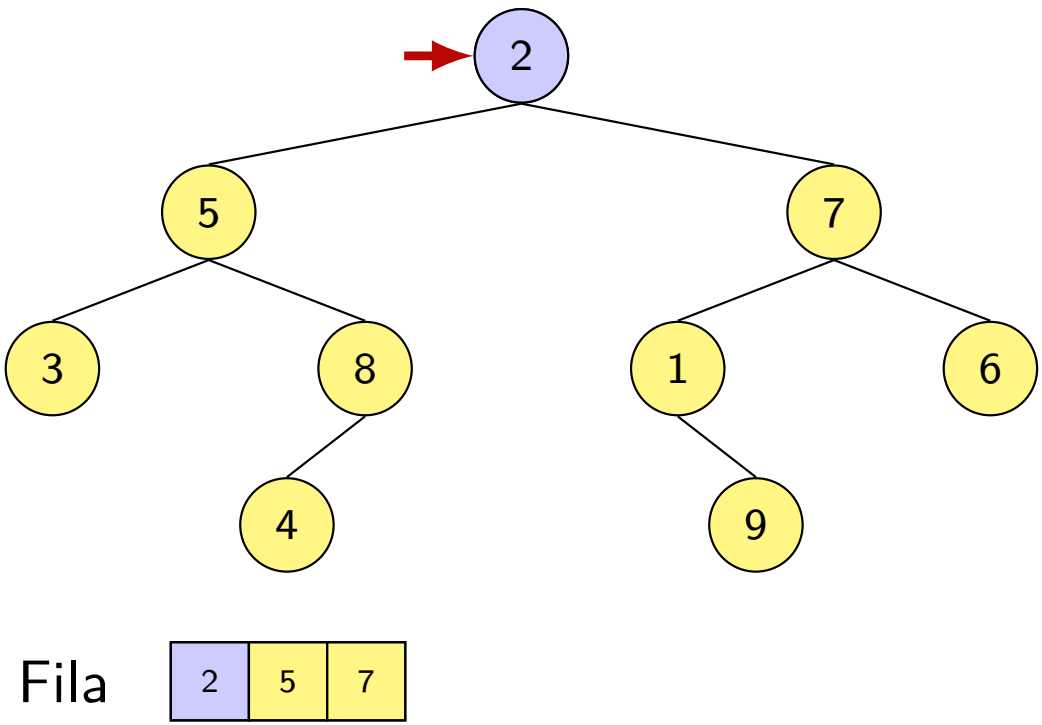


Fila 

Implementação do percurso em largura

Como implementar o percurso em largura?

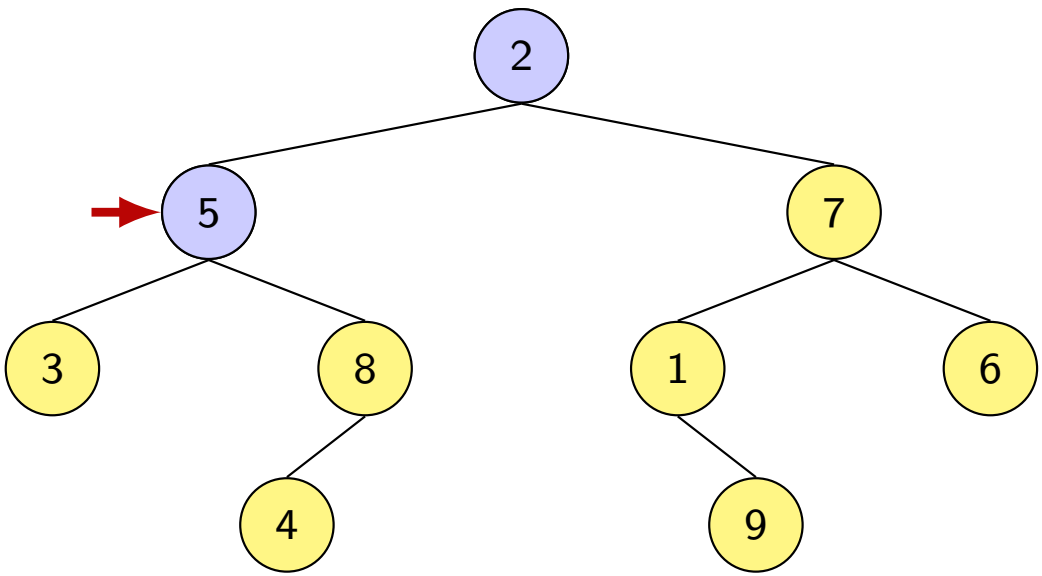
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



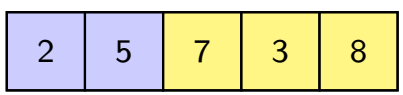
Implementação do percurso em largura

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



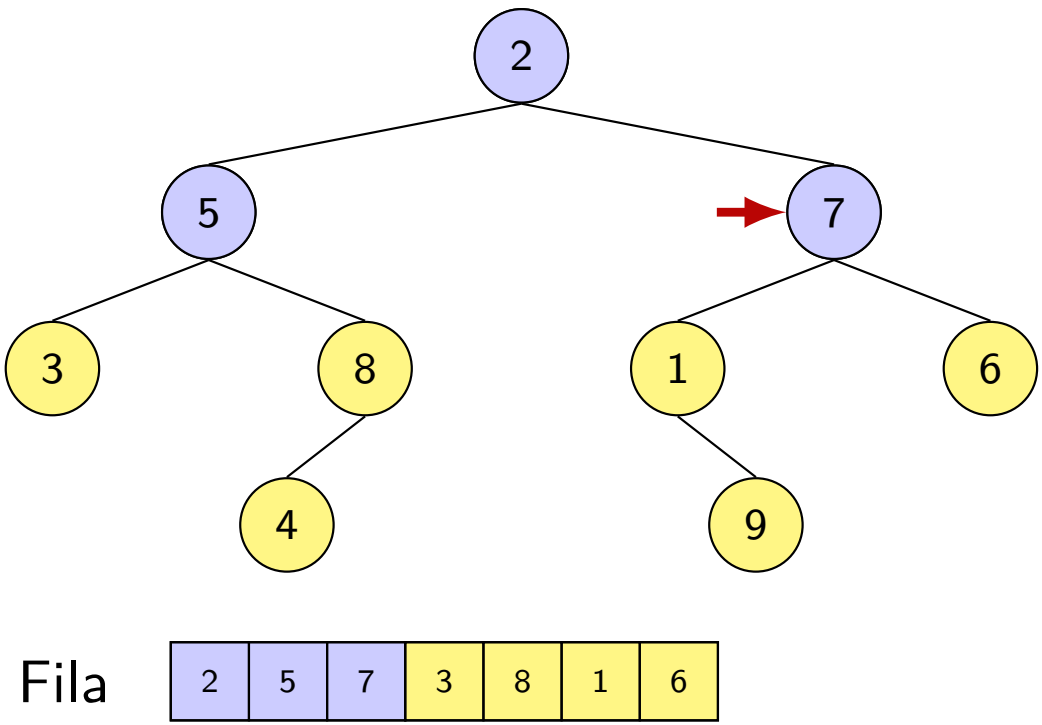
Fila



Implementação do percurso em largura

Como implementar o percurso em largura?

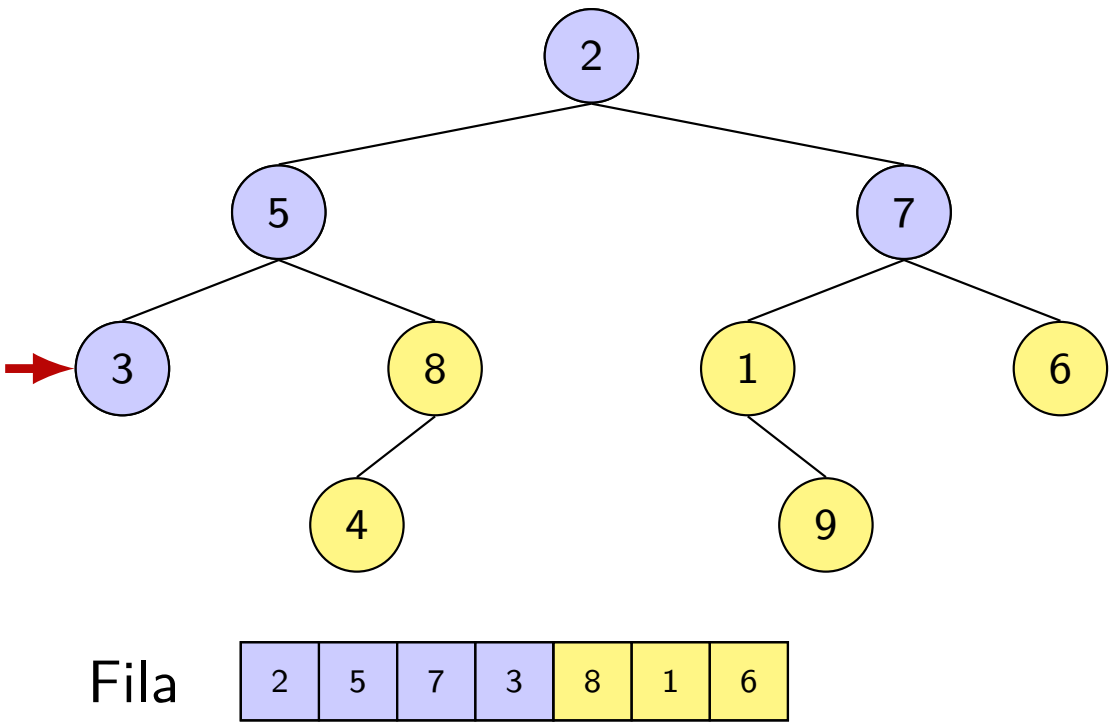
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

Como implementar o percurso em largura?

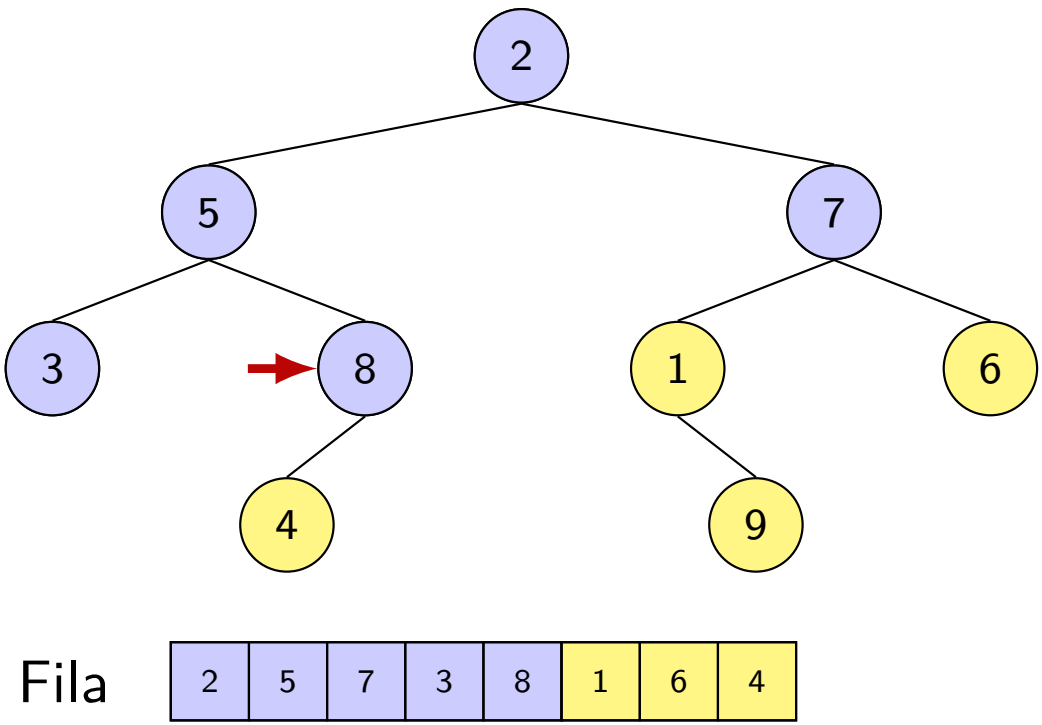
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

Como implementar o percurso em largura?

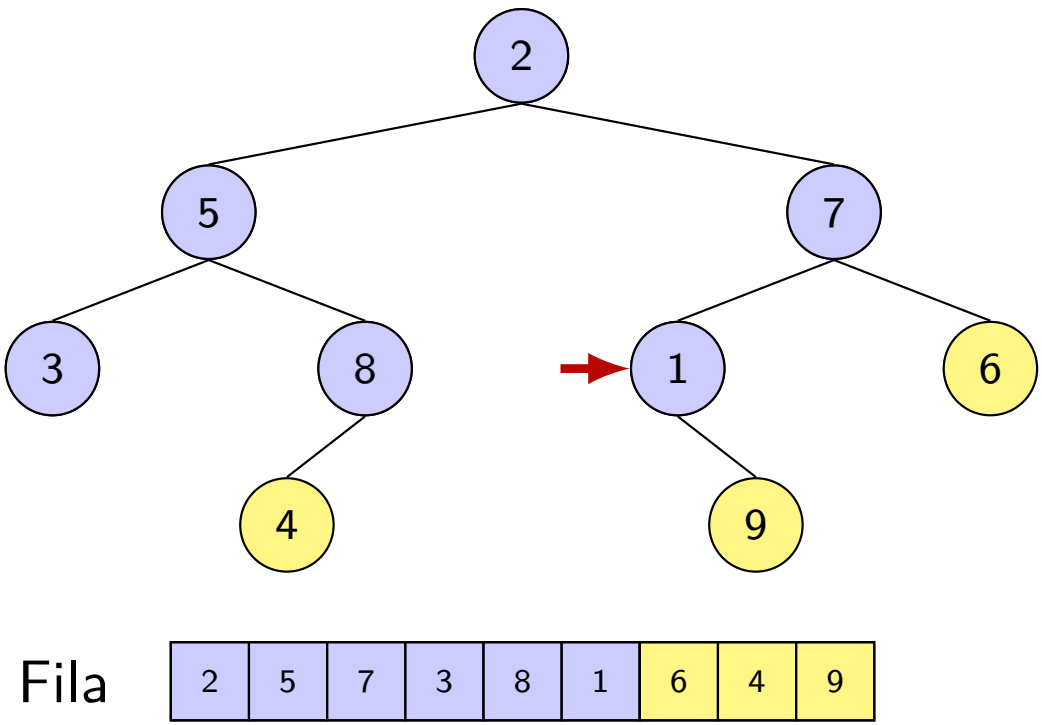
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

Como implementar o percurso em largura?

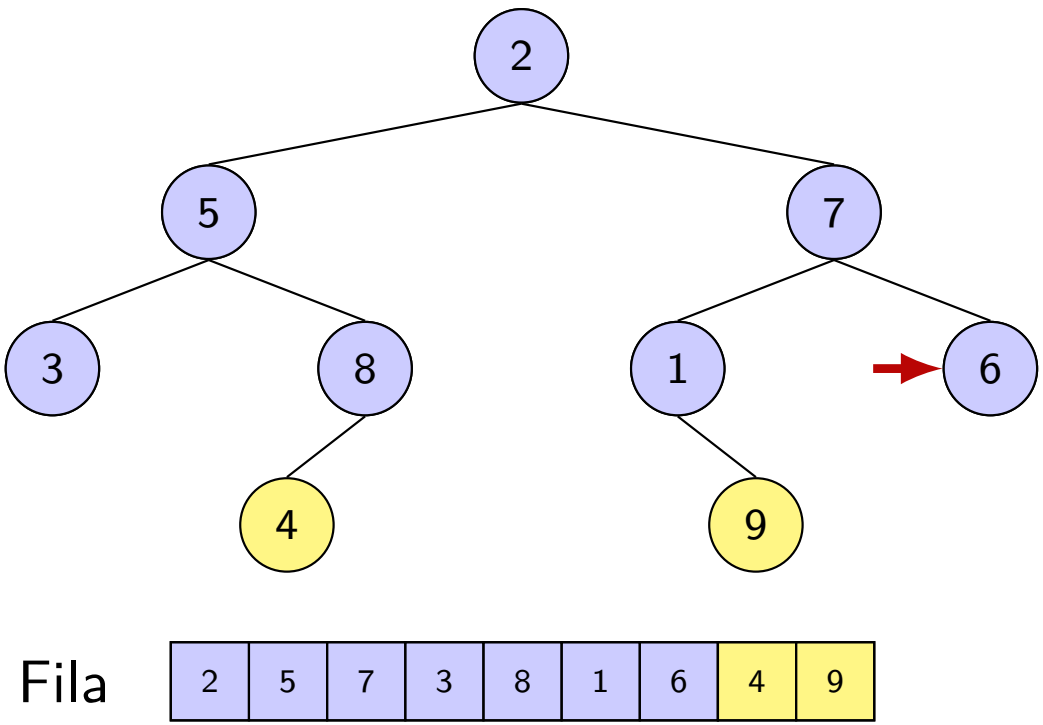
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

Como implementar o percurso em largura?

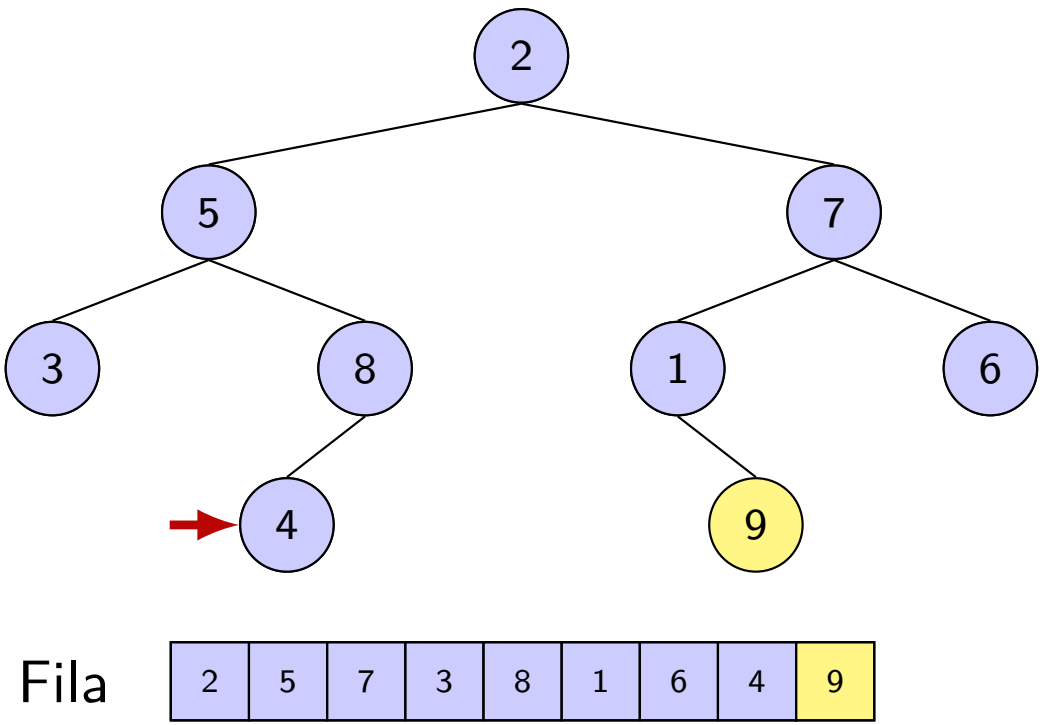
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

Como implementar o percurso em largura?

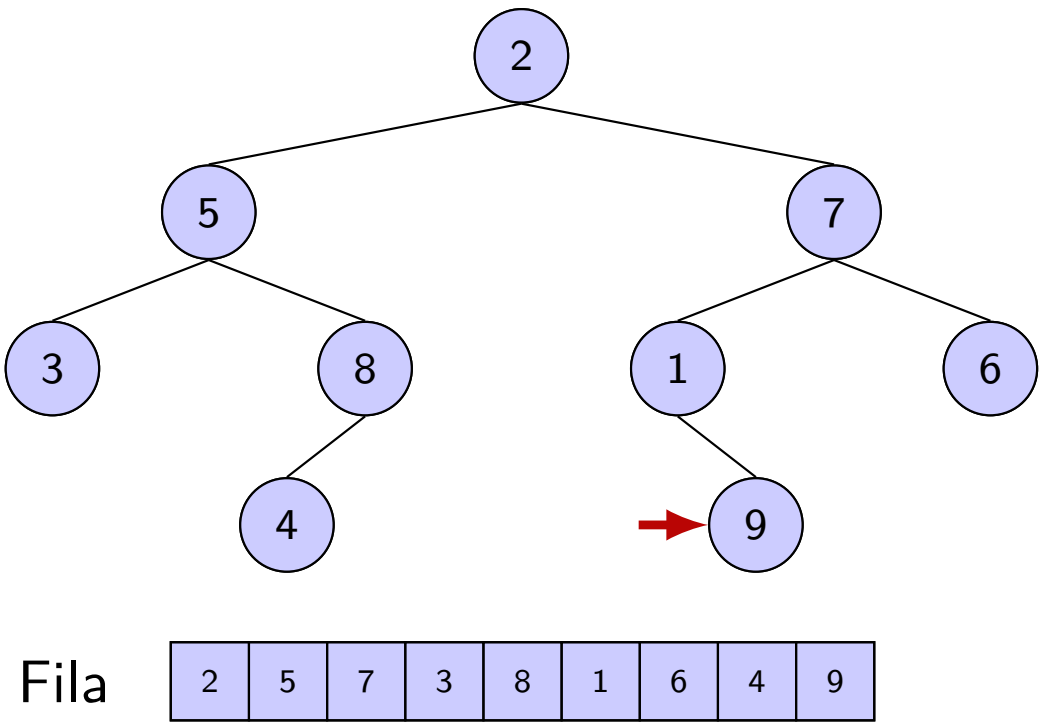
- Usamos uma fila
- Colocamos a raiz na fila e depois
- pegamos um elemento da fila e enfileiramos seus filhos



Implementação do percurso em largura

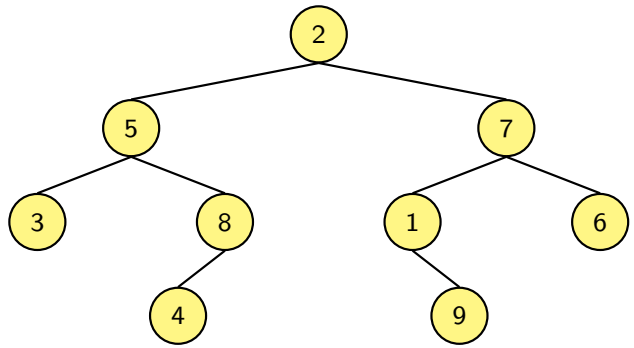
Como implementar o percurso em largura?

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



Percurso em largura

```
87  void percurso_em_largura(No* raiz) {
88    p_fila f;
89    f = criar_fila();
90    enfileirar(f, raiz);
91    while(!fila_vazia(f)) {
92      raiz = desenfileirar(f);
93      if (raiz != NULL) {
94        enfileirar(f, raiz->esq);
95        enfileirar(f, raiz->dir);
96        printf("%d ", raiz->dado); /* visita raiz */
97      }
98    }
99    destruir_fila(f);
100 }
```



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

Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Rafael C. S. Schouery, da Universidade Estadual de Campinas.
2. Feofiloff, Paulo. Algoritmos em linguagem C. Elsevier Brasil, 2009.