Árvores Binárias

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2025

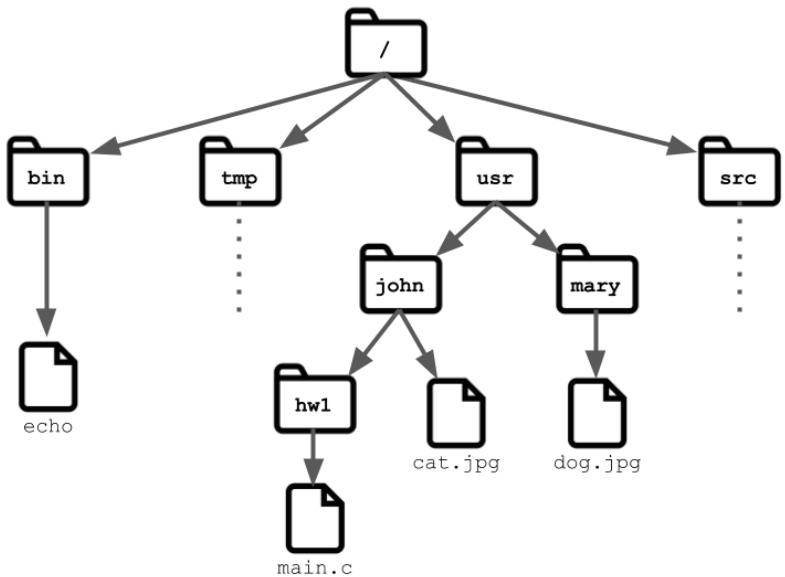
Introdução

Representando uma hierarquia

- Vetores, listas, filas e pilhas são estruturas lineares.
  - A importância dessas estruturas é inegável, mas elas não são adequadas para representar dados dispostos de maneira hierárquica.

Representando uma hierarquia

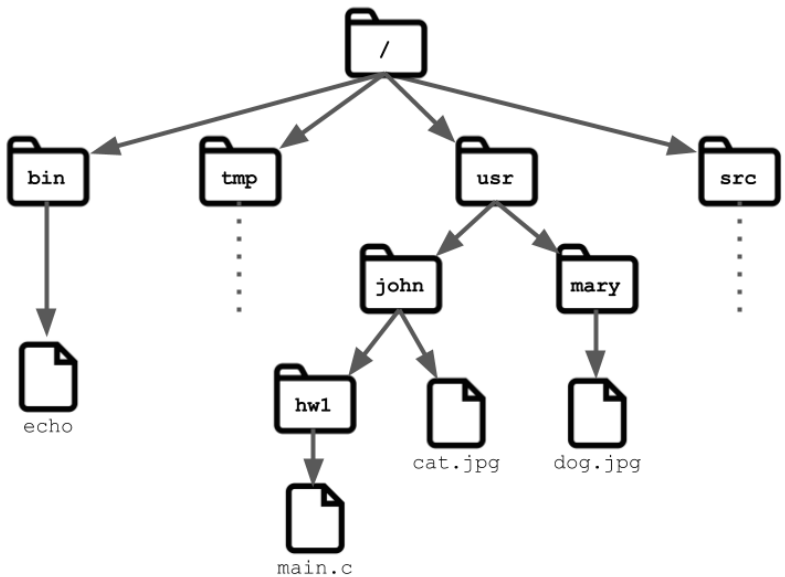
- Vetores, listas, filas e pilhas são estruturas lineares.
  - A importância dessas estruturas é inegável, mas elas não são adequadas para representar dados dispostos de maneira hierárquica.



Hierarquia do sistema de arquivos de um sistema operacional

Representando uma hierarquia

- Vetores, listas, filas e pilhas são estruturas lineares.
  - A importância dessas estruturas é inegável, mas elas não são adequadas para representar dados dispostos de maneira hierárquica.



Hierarquia do sistema de arquivos de um sistema operacional

- As árvores são estruturas de dados mais adequadas para representar hierarquias.

Árvore — Definição Recursiva

Uma árvore T é um conjunto finito de elementos denominados nós, tais que:

Árvore — Definição Recursiva

Uma árvore T é um conjunto finito de elementos denominados nós, tais que:

(a) $T = \emptyset$, e a árvore é dita vazia; ou

Árvore — Definição Recursiva

Uma árvore T é um conjunto finito de elementos denominados nós, tais que:

(a) $T = \emptyset$, e a árvore é dita vazia; ou

(b) $T \neq \emptyset$ e ela possui um nó especial r, chamado raiz de T; os nós restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.

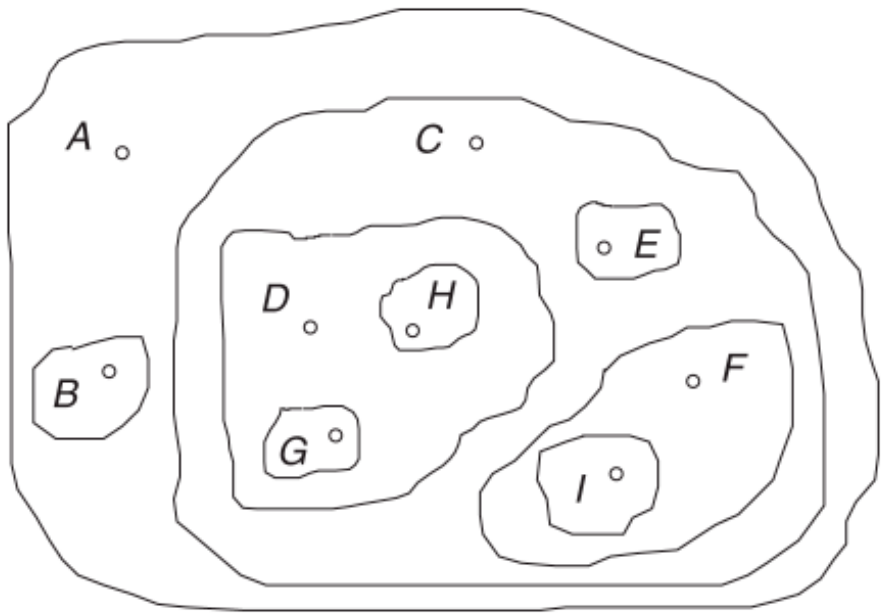
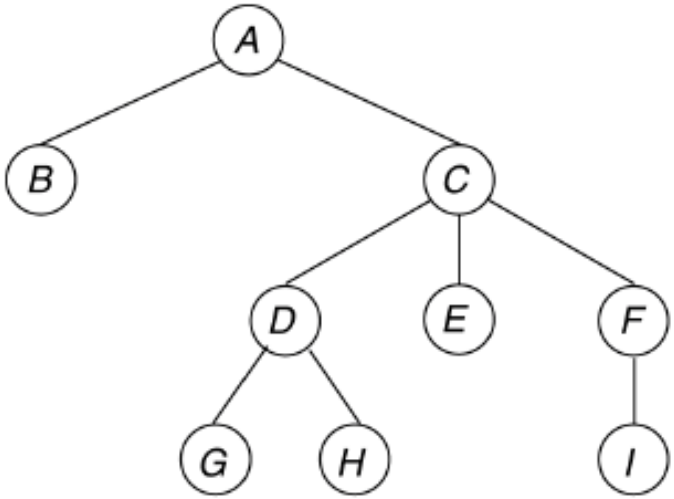


Diagrama de inclusão

Árvore — Outras Representações



Representação hierárquica

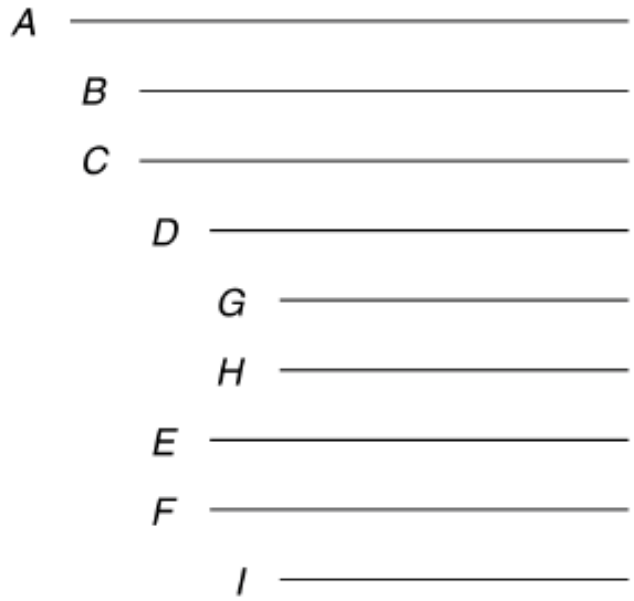
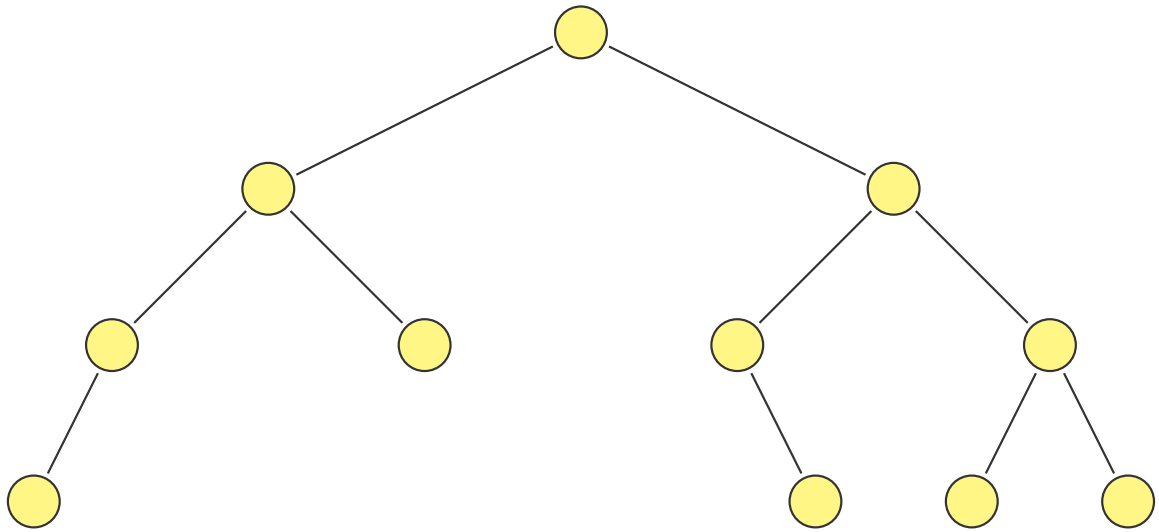


Diagrama de barras

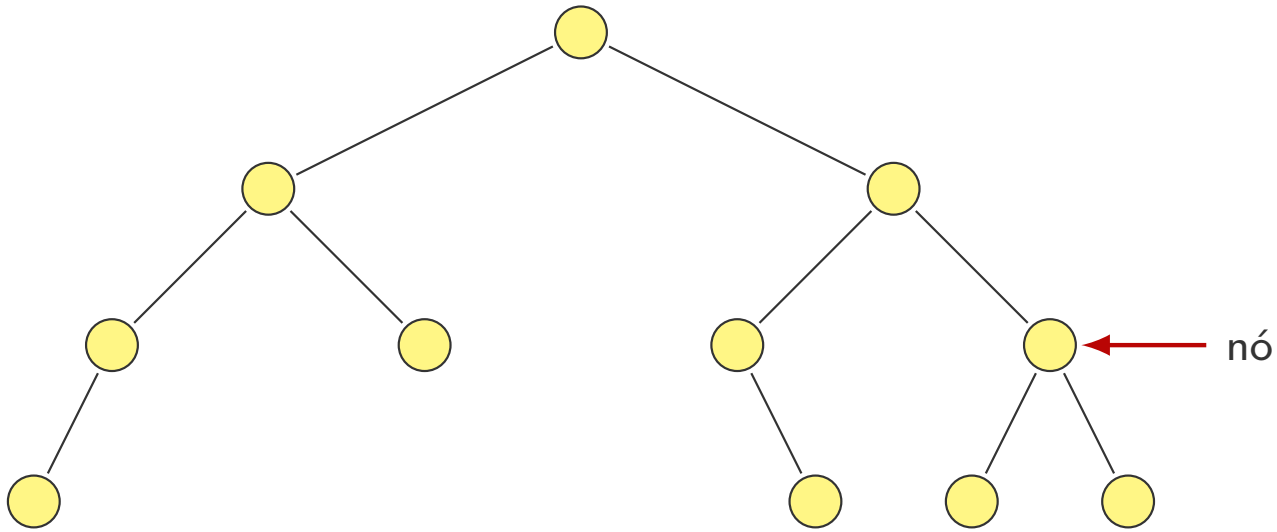
(A (B) (C (D (G) (H)) (E) (F (I))))

Representação por parênteses aninhados

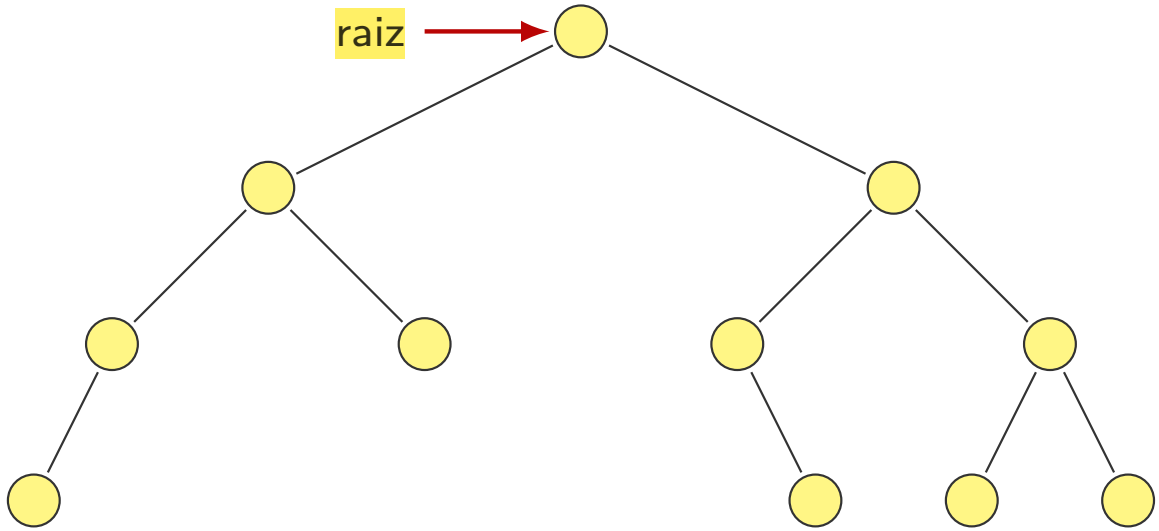
Definições Adicionais



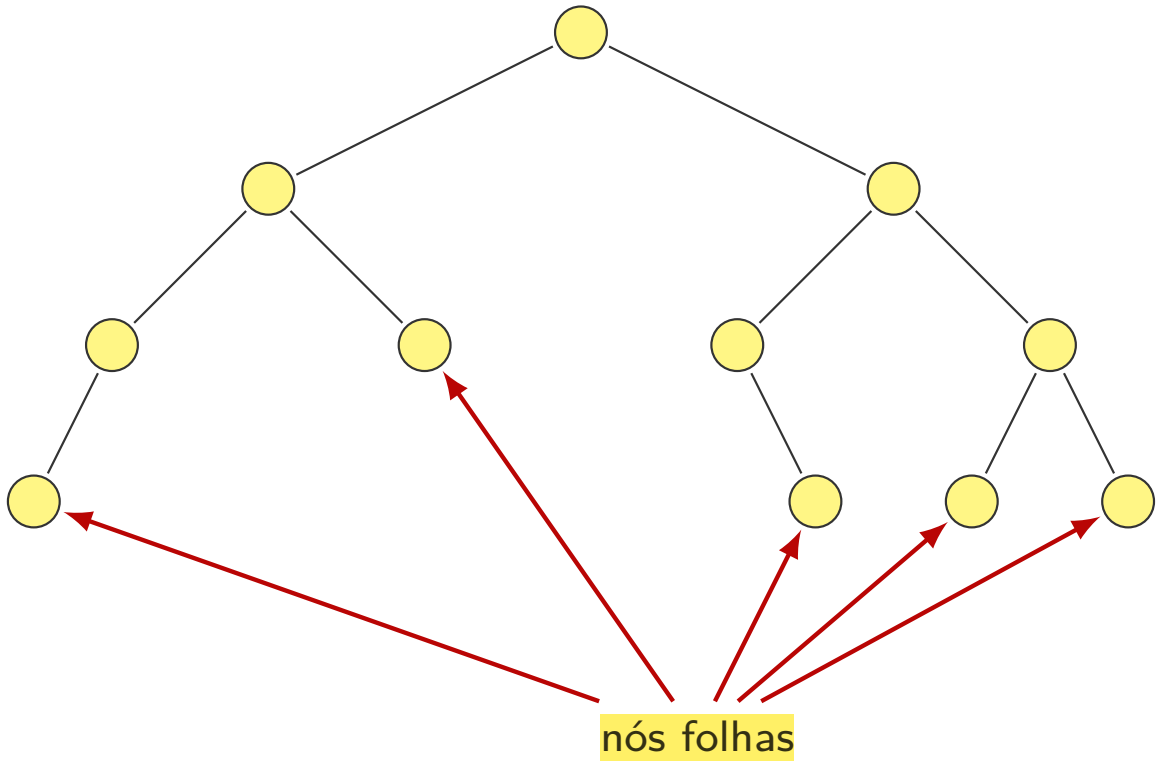
Definições Adicionais



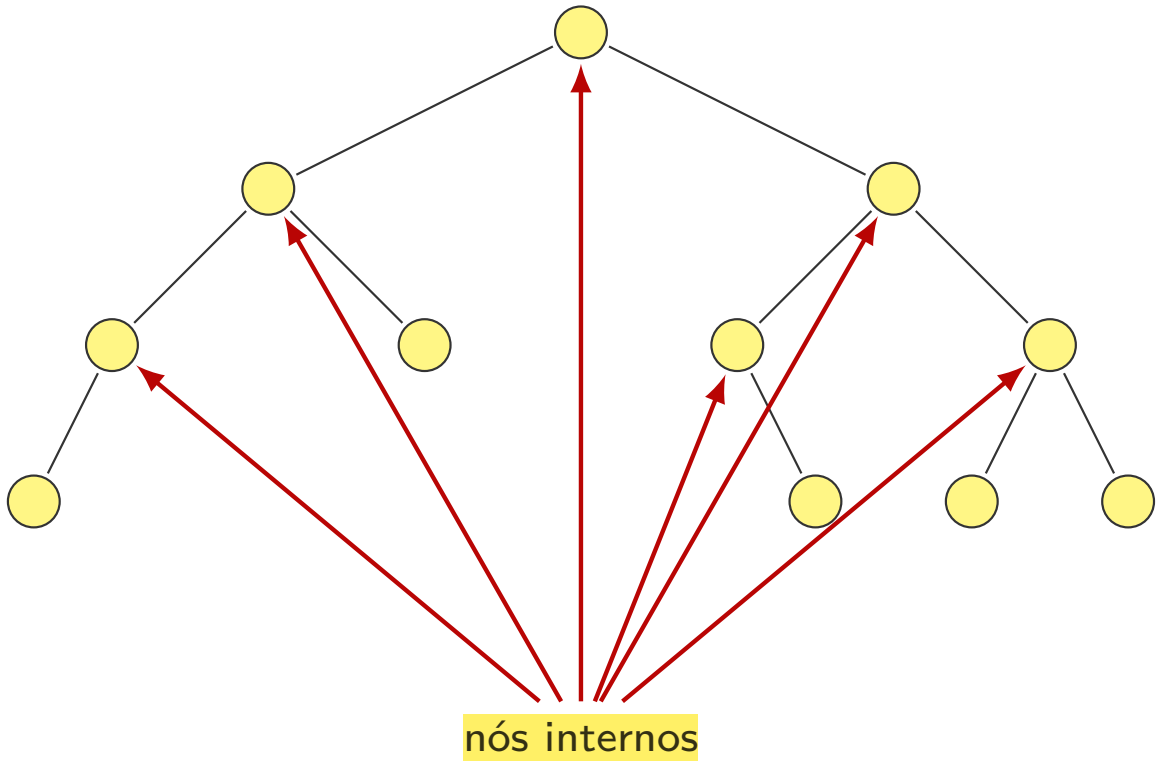
Definições Adicionais



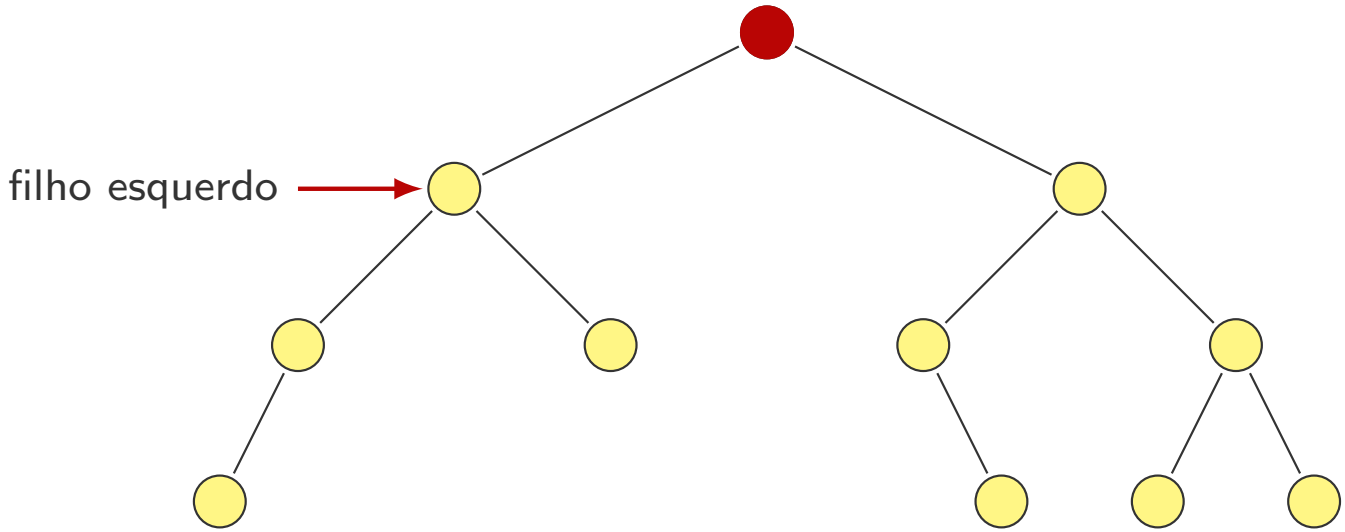
Definições Adicionais



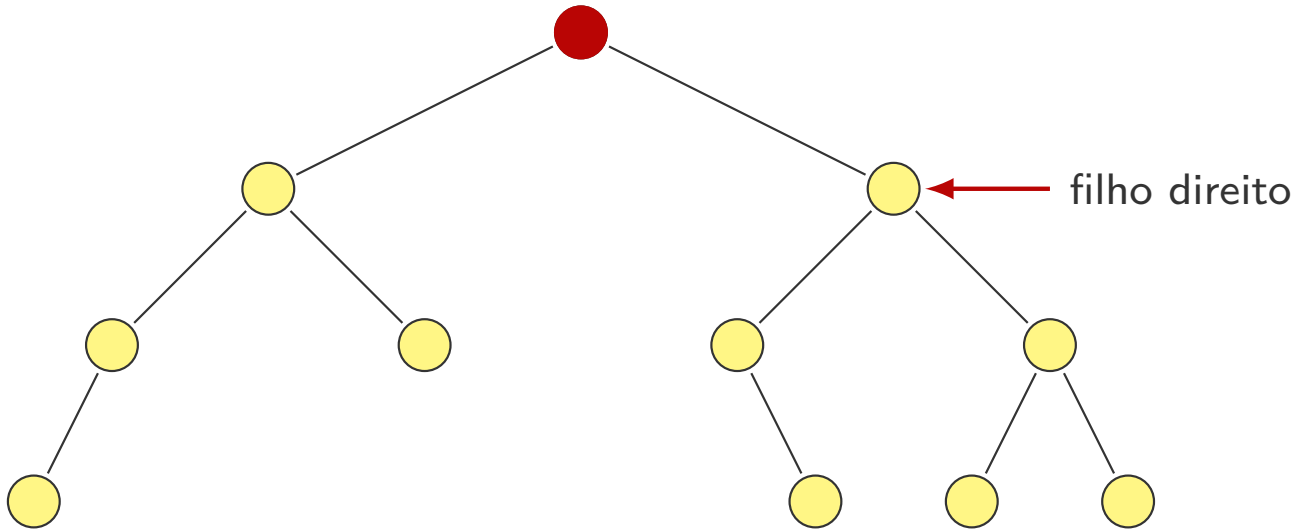
Definições Adicionais



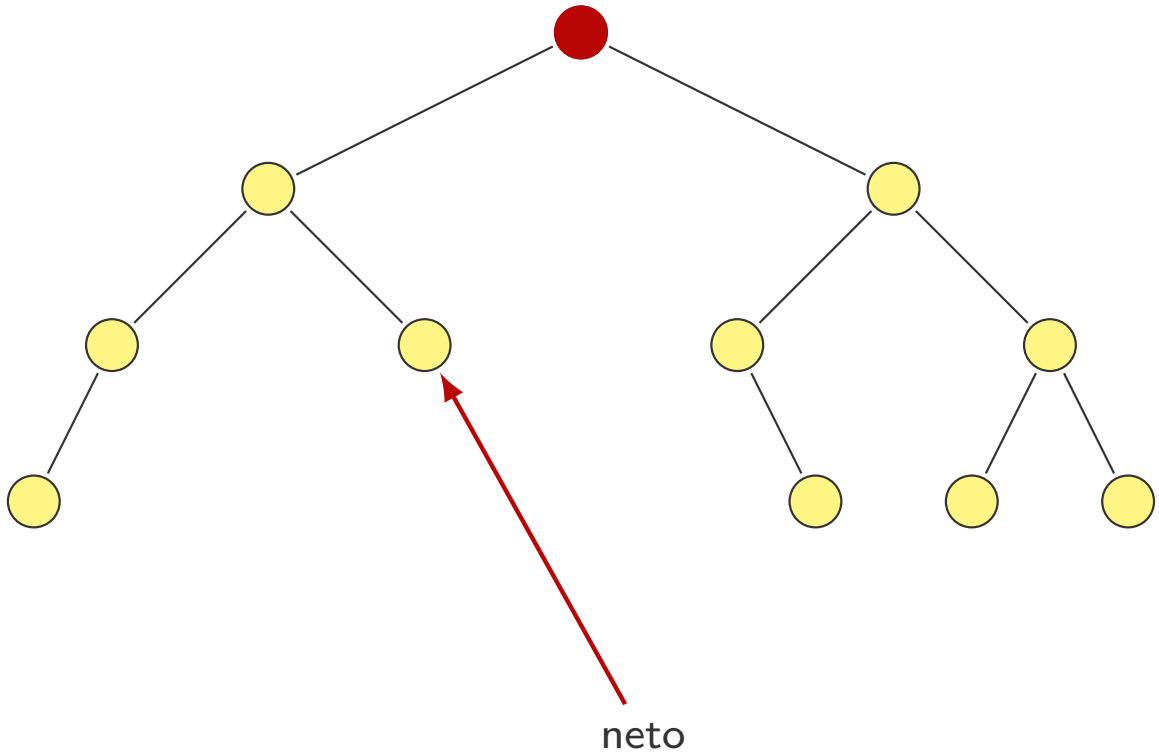
Definições Adicionais



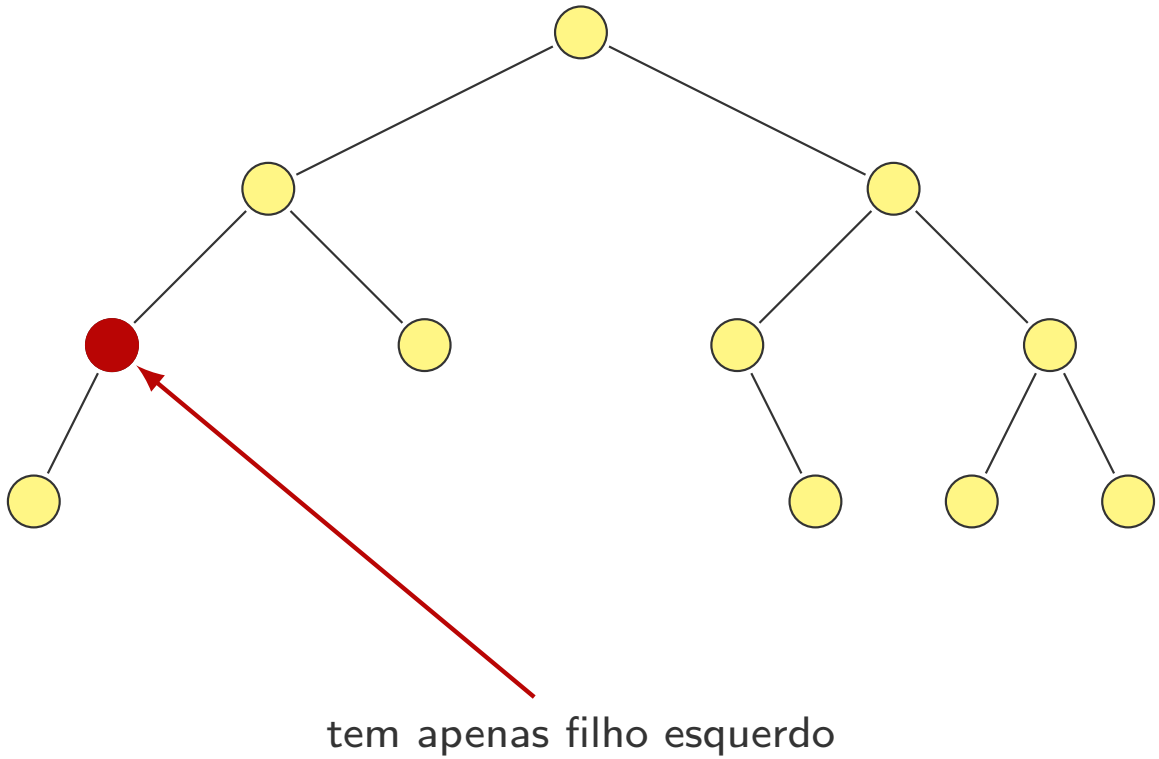
Definições Adicionais



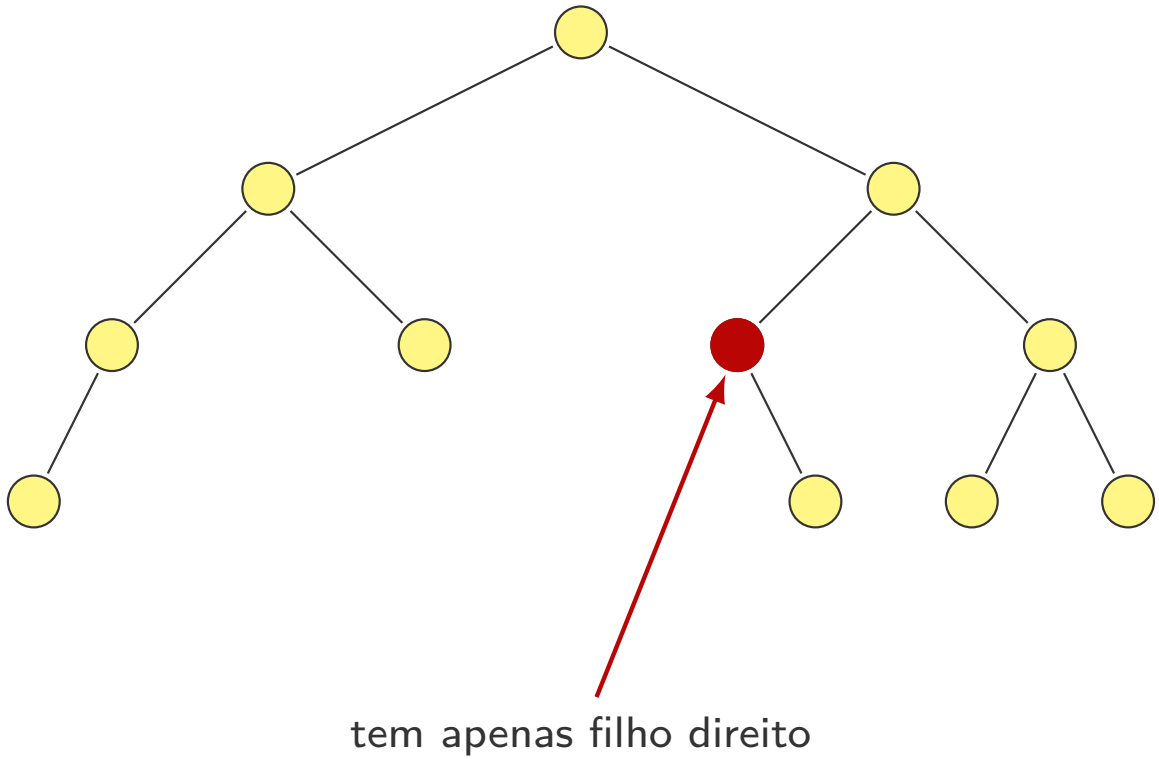
Definições Adicionais



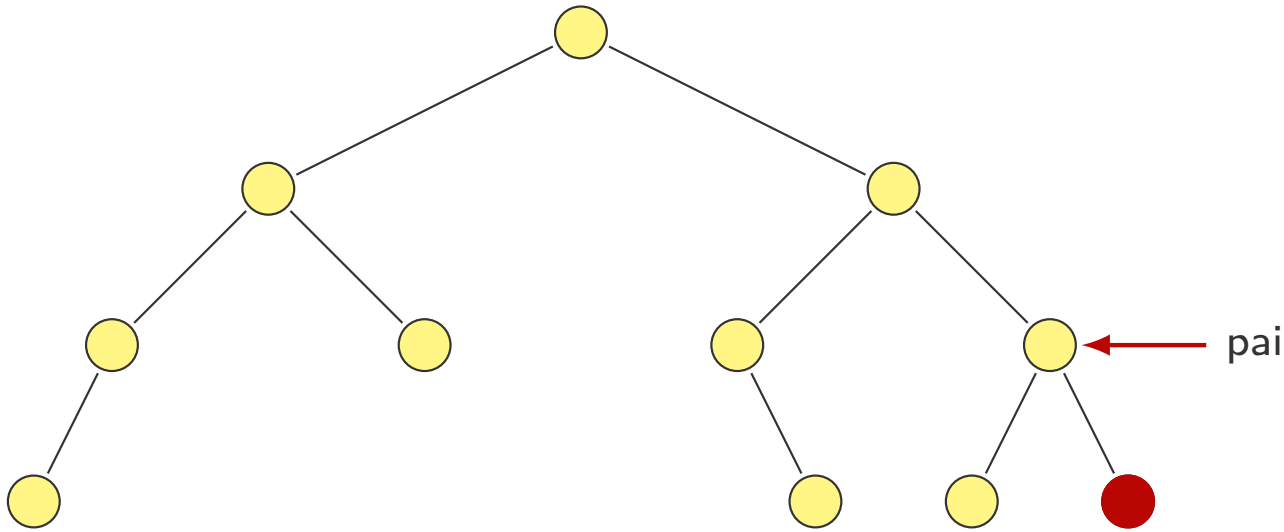
Definições Adicionais



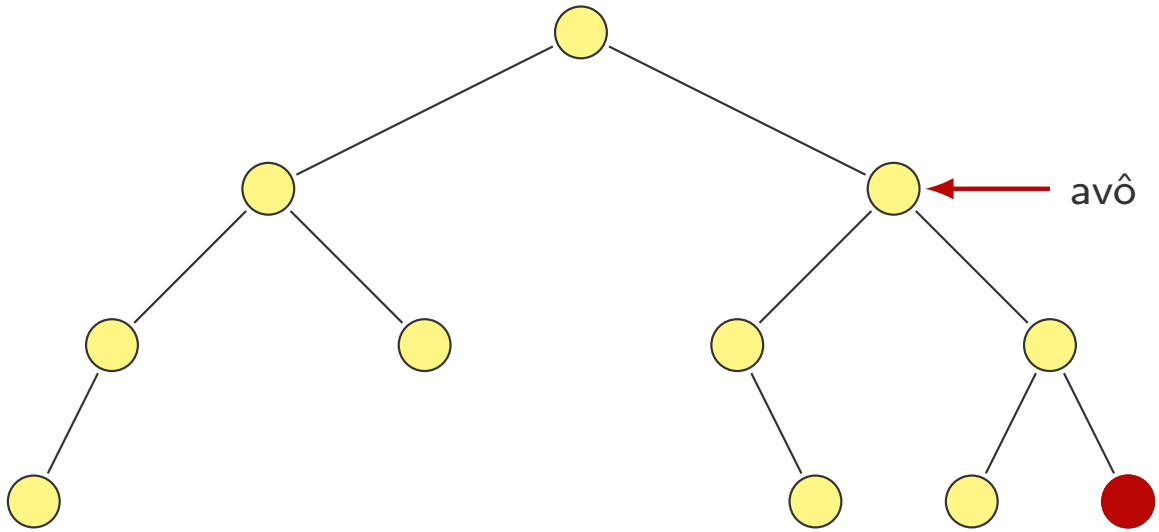
Definições Adicionais



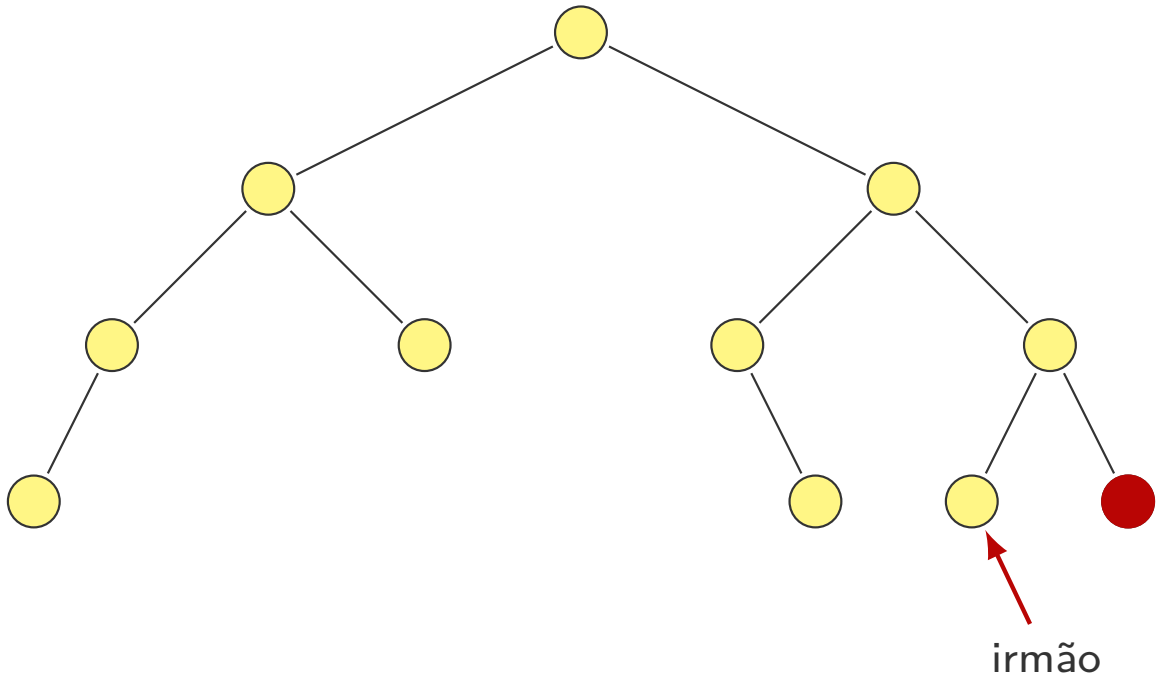
Definições Adicionais



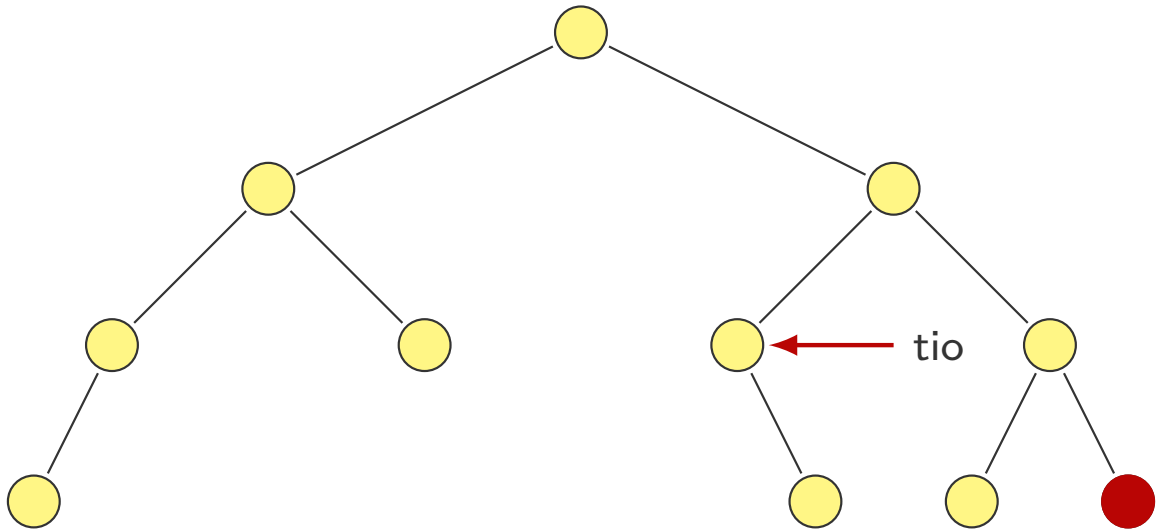
Definições Adicionais



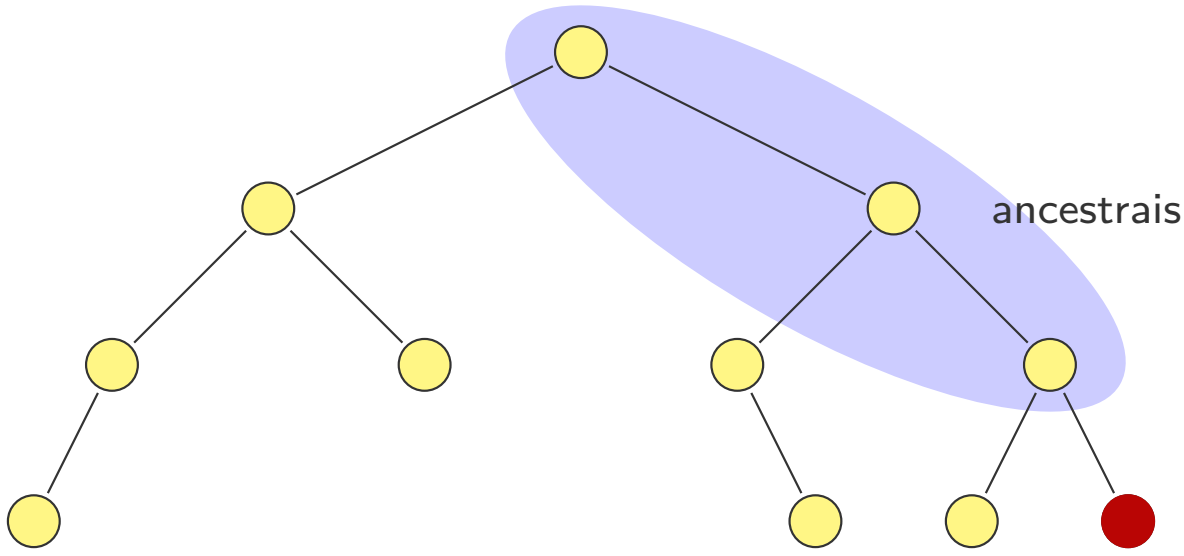
Definições Adicionais



Definições Adicionais

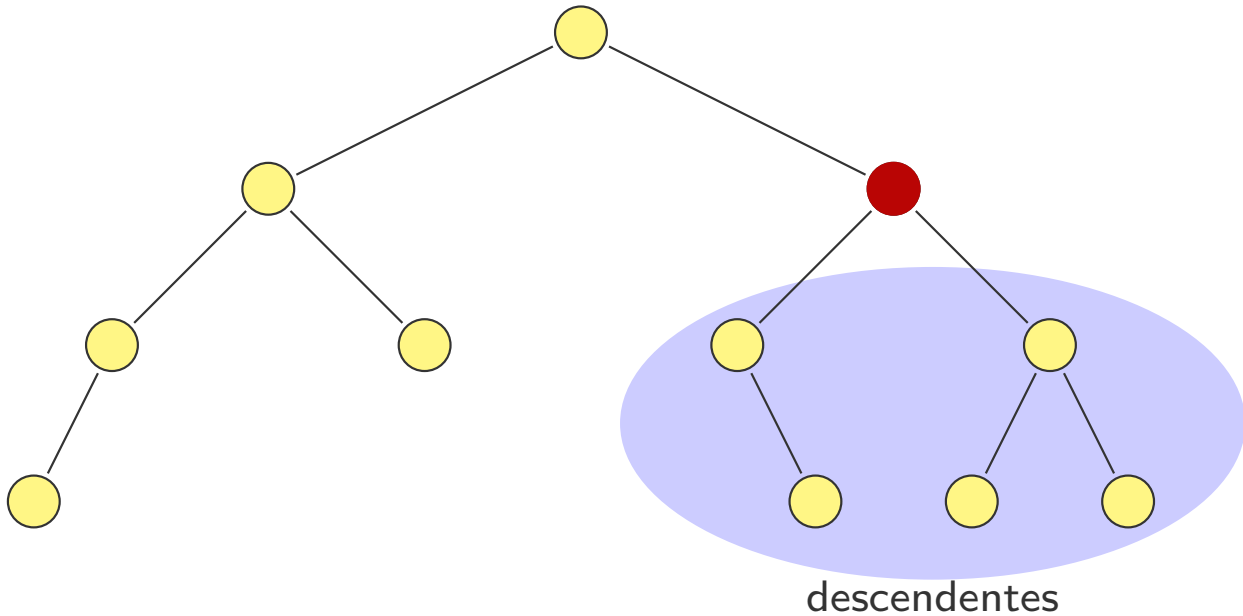


Definições Adicionais

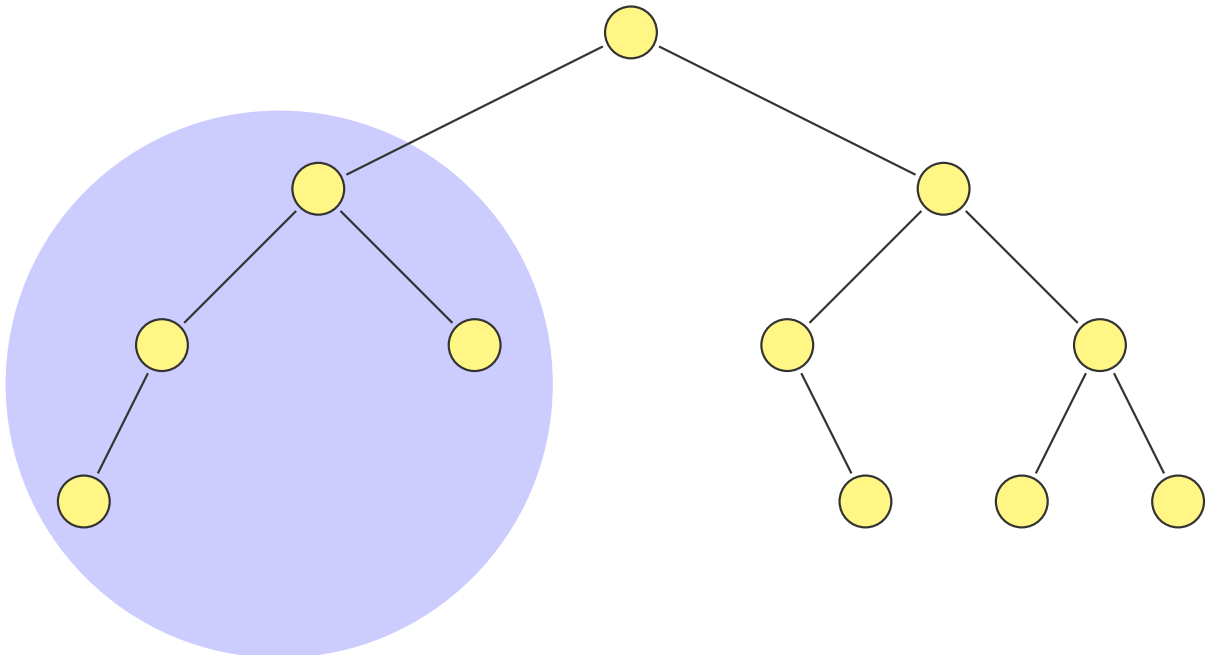


Um caminho na árvore é uma sequência de nós distintos $v_1, v_2, \ldots, v_k$, tal que existe sempre entre nós consecutivos a relação "é filho de" ou "é pai de". O comprimento do caminho é o número de arestas no caminho.

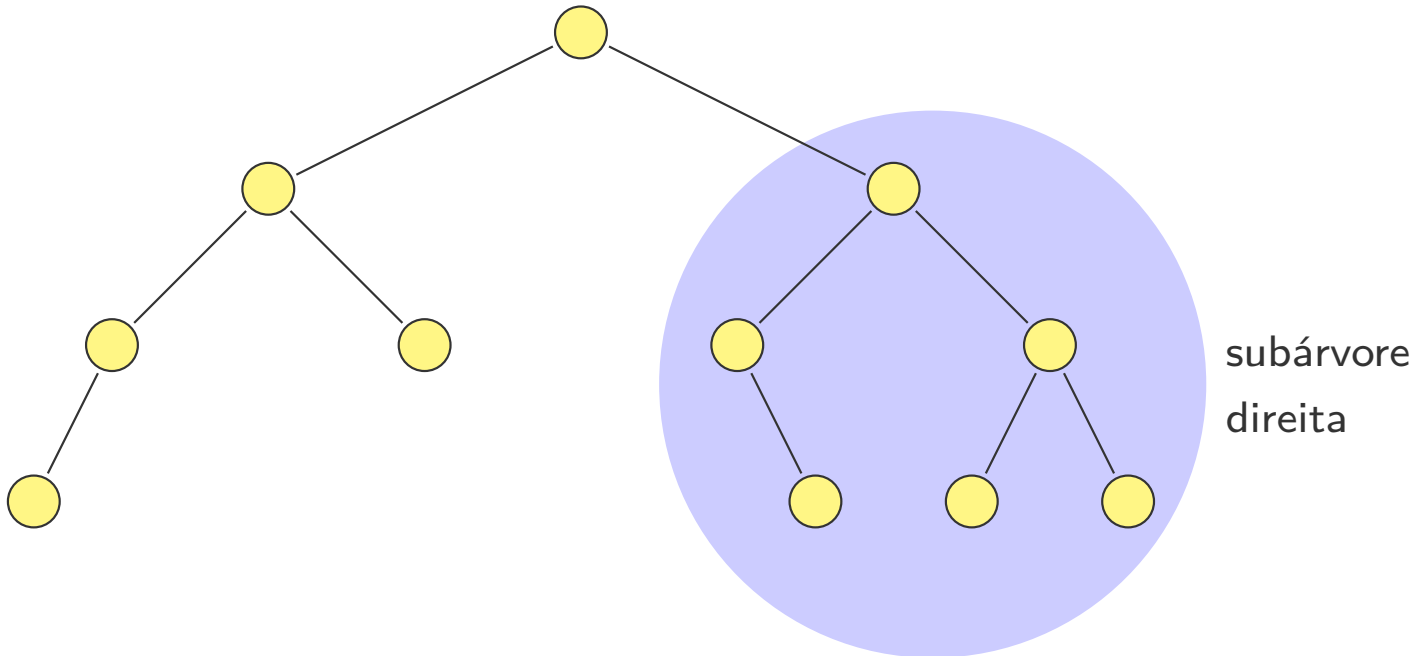
Definições Adicionais



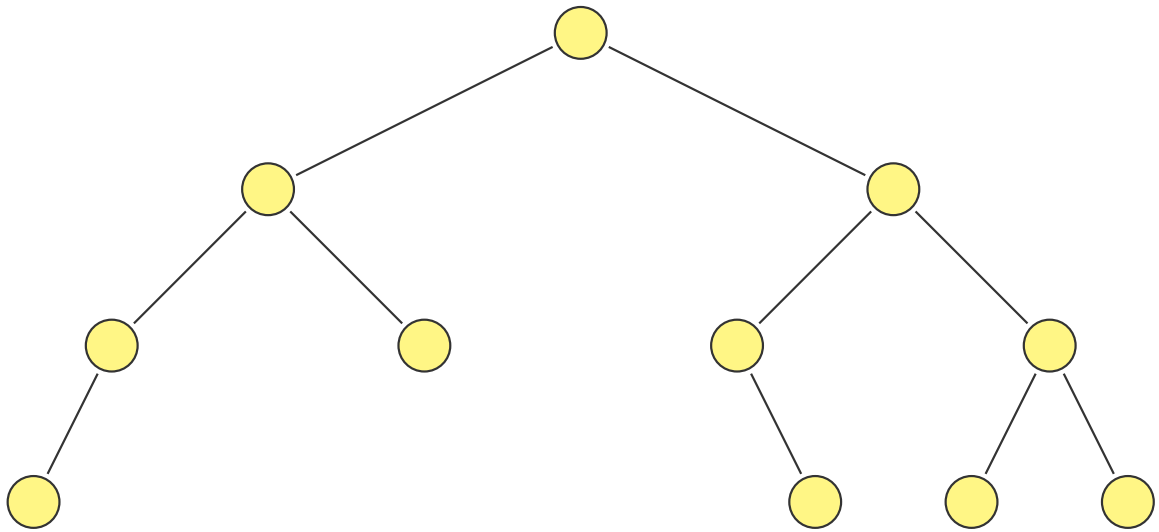
Definições Adicionais



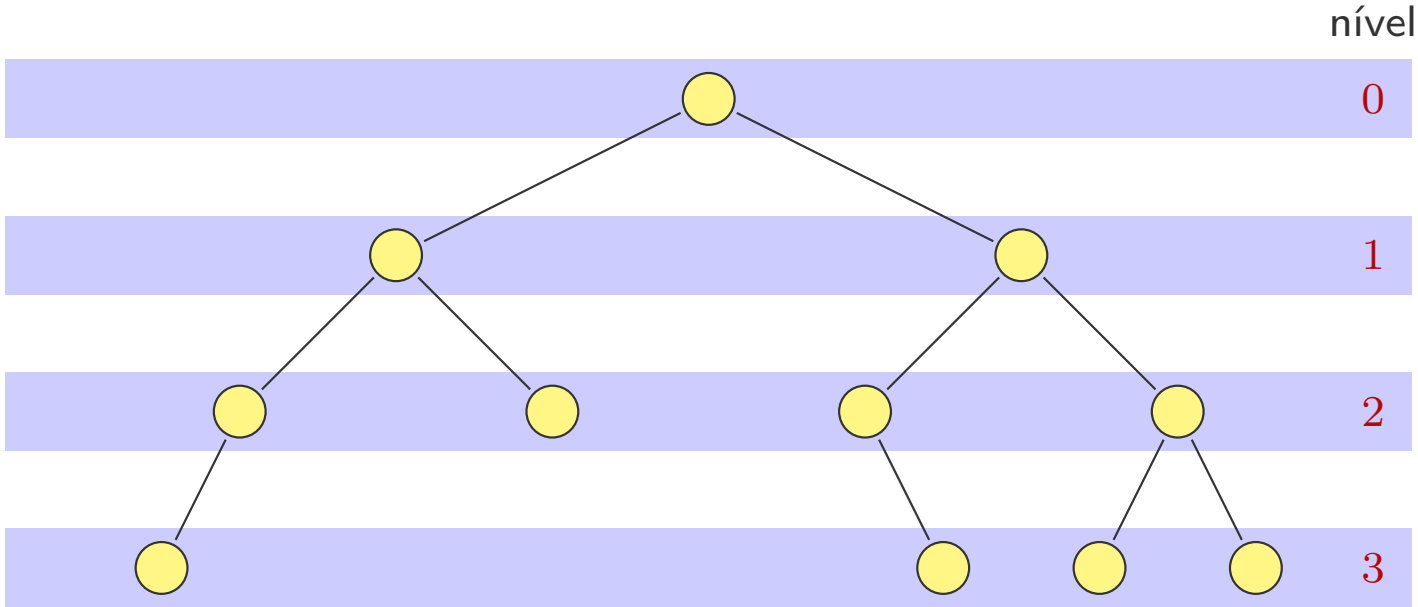
Definições Adicionais



Definições — Nível e Altura

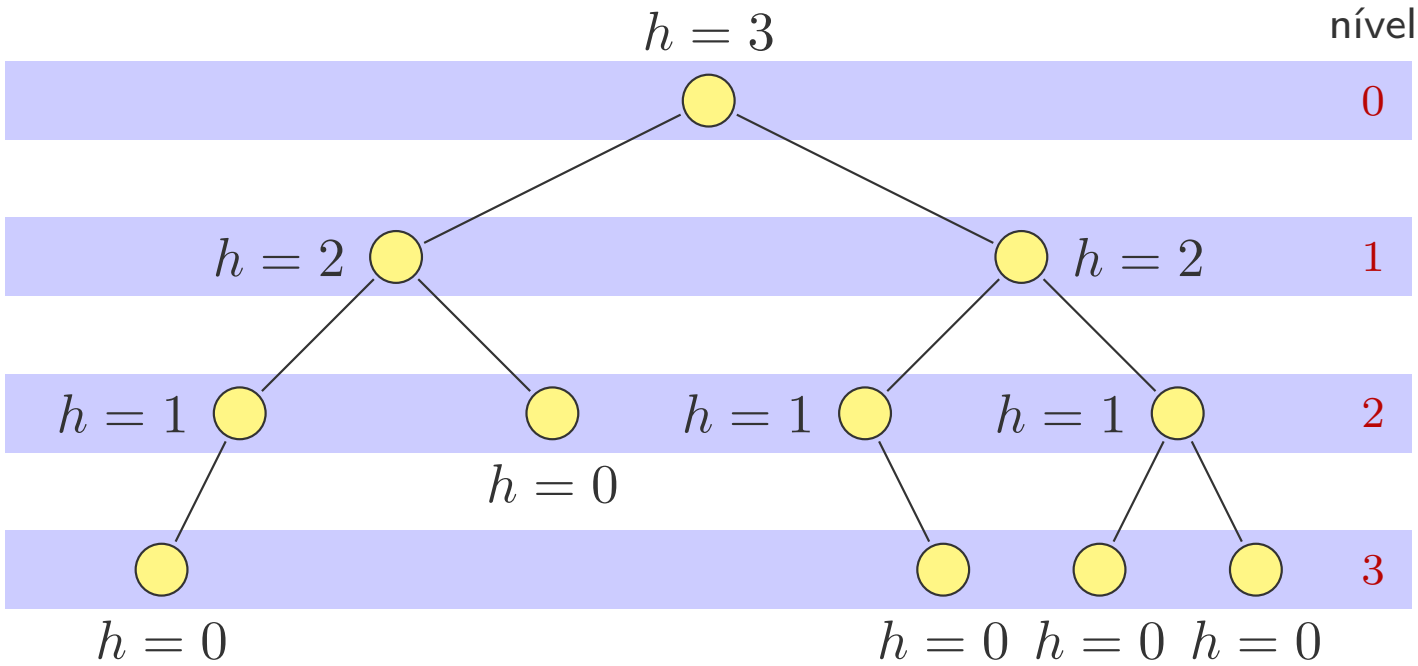


Definições — Nível e Altura



Nível de um nó v: é o comprimento do caminho de v até a raiz.

Definições — Nível e Altura

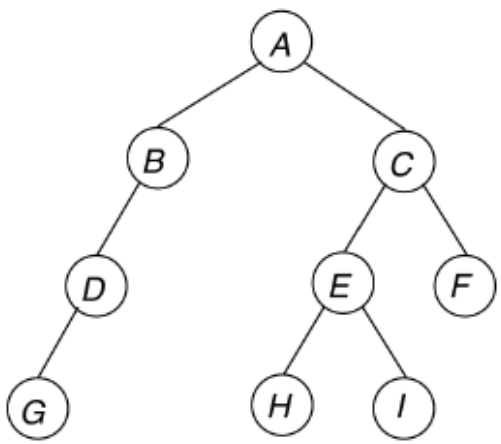


Nível de um nó v: é o comprimento do caminho de v até a raiz.

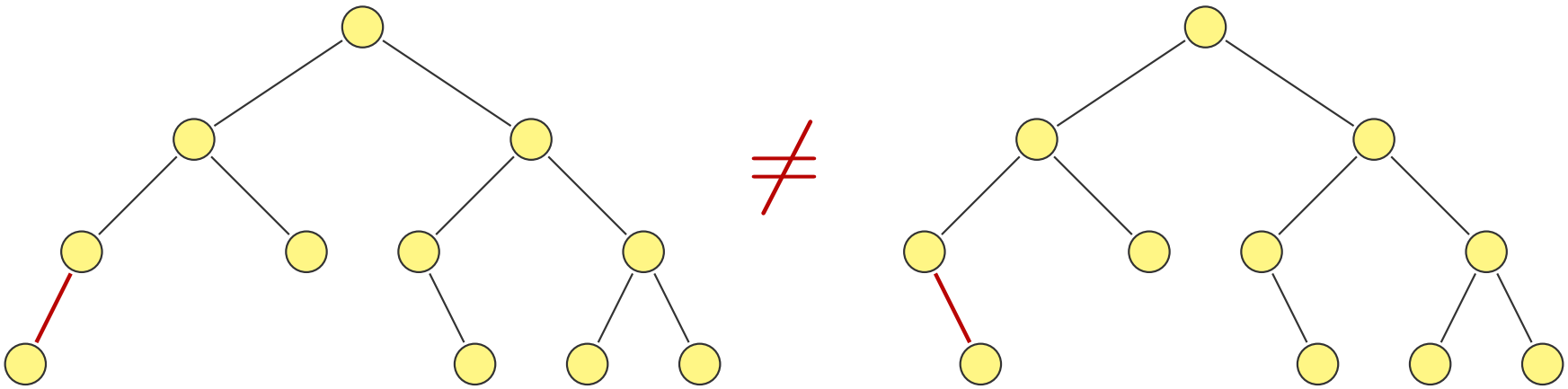
Altura h de um nó v: comprimento do caminho mais longo de v até uma de suas folhas descendentes.

Árvore Binária — Definição Recursiva

- Uma árvore binária T é um conjunto finito de elementos denominados nós, tal que:
  - $T = \emptyset$ e a árvore é dita vazia; ou
  - $T \neq \emptyset$ e existe um nó especial r, chamado raiz de T, e os restantes podem ser divididos em dois subconjuntos disjuntos, T_r^E e T_r^D, a subárvore esquerda e a subárvore direita de r, respectivamente, as quais são também árvores binárias.



Quando duas árvores binárias são iguais?



A posição dos filhos é relevante!

Tipos específicos de árvores binárias

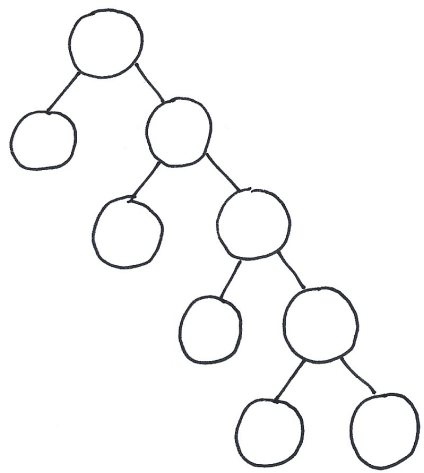
Definição

Uma **árvore estritamente binária** é uma árvore binária na qual todo nó possui zero ou dois filhos.

Tipos específicos de árvores binárias

Definição

Uma **árvore estritamente binária** é uma árvore binária na qual todo nó possui zero ou dois filhos.



Uma árvore estritamente binária

Tipos específicos de árvores binárias

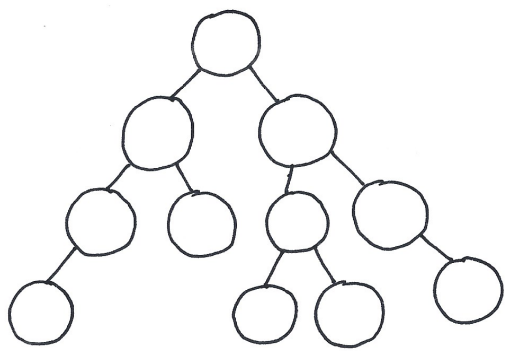
Definição

Uma árvore binária é dita **completa** se todo nó que não tem pelo menos um filho está no penúltimo ou o último nível.

Tipos específicos de árvores binárias

Definição

Uma árvore binária é dita **completa** se todo nó que não tem pelo menos um filho está no penúltimo ou o último nível.



Uma árvore completa

Tipos específicos de árvores binárias

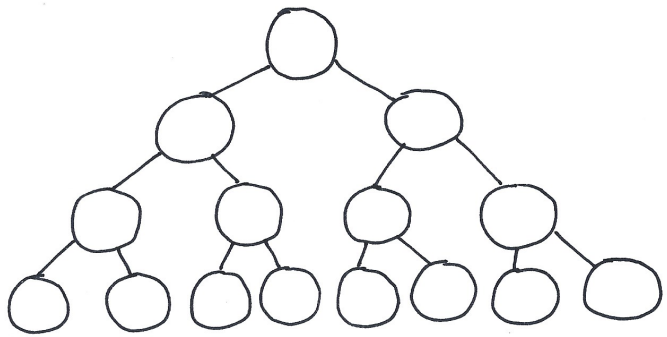
Definição

Uma árvore binária é dita **cheia** se todos os seus nós internos têm dois filhos e todas as folhas estão no último nível da árvore.

Tipos específicos de árvores binárias

Definição

Uma árvore binária é dita **cheia** se todos os seus nós internos têm dois filhos e todas as folhas estão no último nível da árvore.

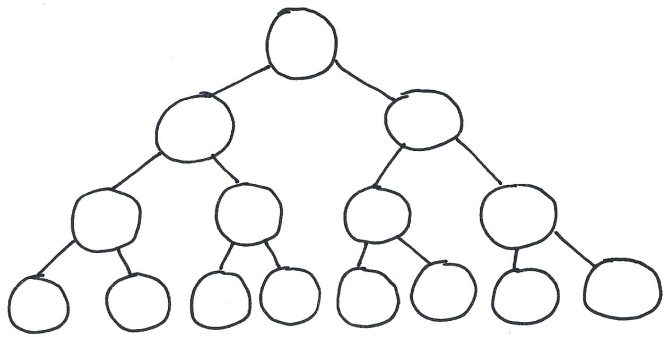


Uma árvore cheia

Tipos específicos de árvores binárias

Definição

Uma árvore binária é dita **cheia** se todos os seus nós internos têm dois filhos e todas as folhas estão no último nível da árvore.



Uma árvore cheia

Pergunta

Quantos nós tem uma árvore cheia de altura h?

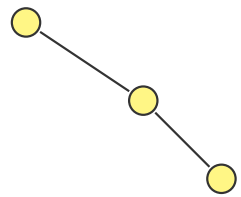
Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

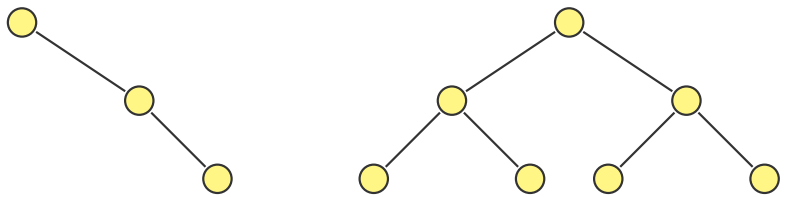
- tem no mínimo $h - 1$ nós



Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

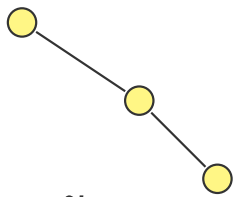
- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós

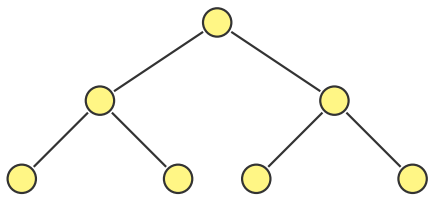


Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós



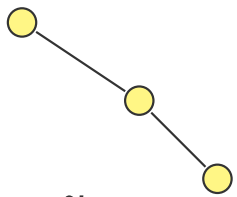


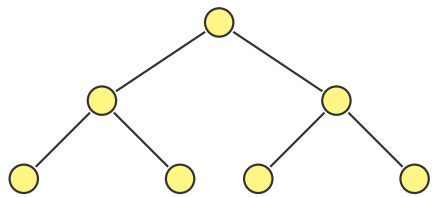
Se a árvore binária tem $n \geq 1$ nós, então:

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós





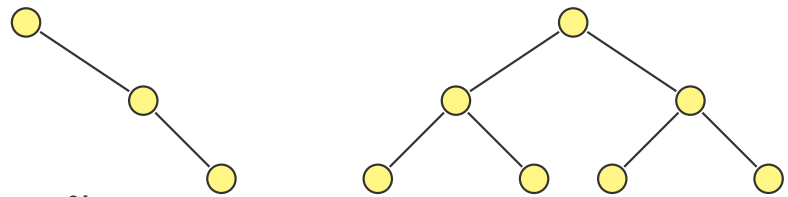
Se a árvore binária tem $n \geq 1$ nós, então:

- a altura é no mínimo $\lceil \log_2(n+1) \rceil - 1$

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós



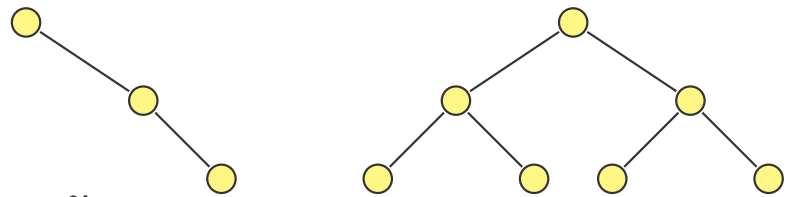
Se a árvore binária tem $n \geq 1$ nós, então:

- a altura é no mínimo $\lceil \log_2(n+1) \rceil - 1$
  - $n \leq 2^{h+1} - 1 \Rightarrow n + 1 \leq 2^{h+1} \Rightarrow \log_2(n+1) \leq \log_2 2^{h+1} \Rightarrow h \geq \log_2(n+1) - 1$

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós



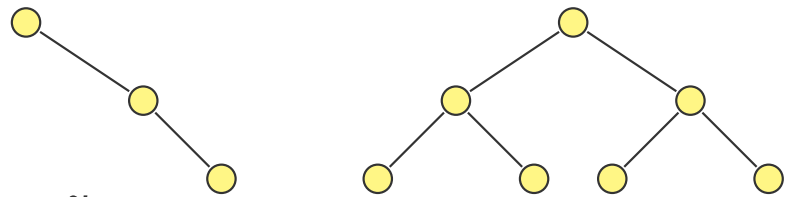
Se a árvore binária tem $n \geq 1$ nós, então:

- a altura é no mínimo $\lceil \log_2(n+1) \rceil - 1$
  - $n \leq 2^{h+1} - 1 \Rightarrow n + 1 \leq 2^{h+1} \Rightarrow \log_2(n+1) \leq \log_2 2^{h+1} \Rightarrow h \geq \log_2(n+1) - 1$
  - quando a árvore é completa

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós



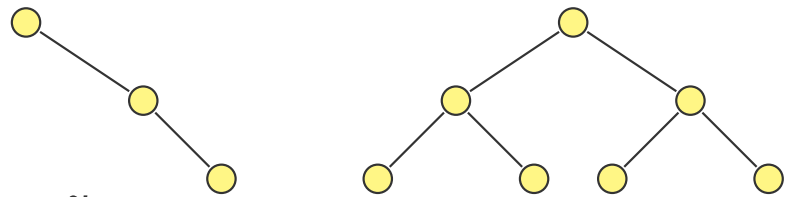
Se a árvore binária tem $n \geq 1$ nós, então:

- a altura é no mínimo $\lceil \log_2(n+1) \rceil - 1$
  - $n \leq 2^{h+1} - 1 \Rightarrow n + 1 \leq 2^{h+1} \Rightarrow \log_2(n+1) \leq \log_2 2^{h+1} \Rightarrow h \geq \log_2(n+1) - 1$
  - quando a árvore é completa
- a altura é no máximo $n - 1$

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

- tem no mínimo $h - 1$ nós
- tem no máximo $2^{h+1} - 1$ nós

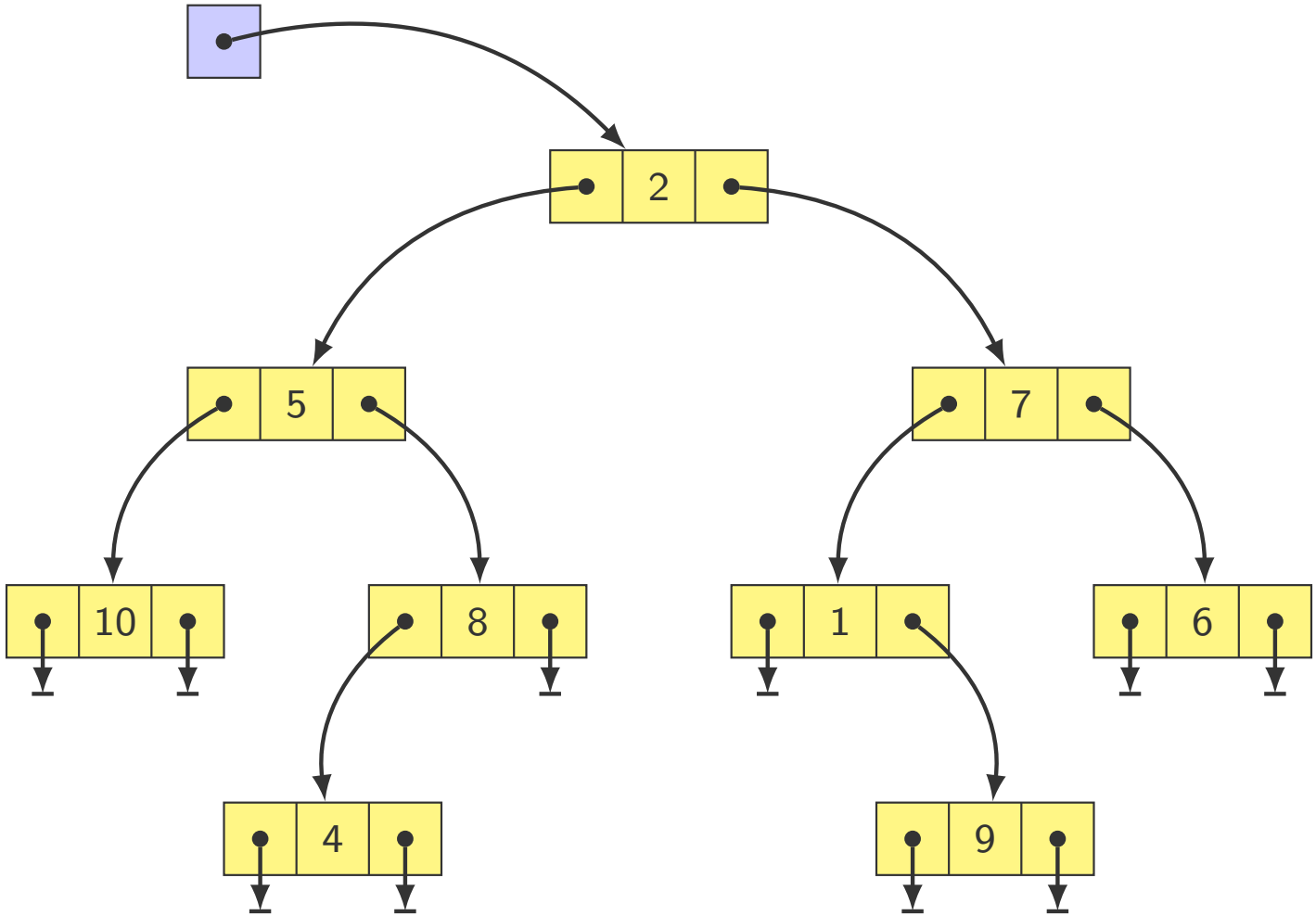


Se a árvore binária tem $n \geq 1$ nós, então:

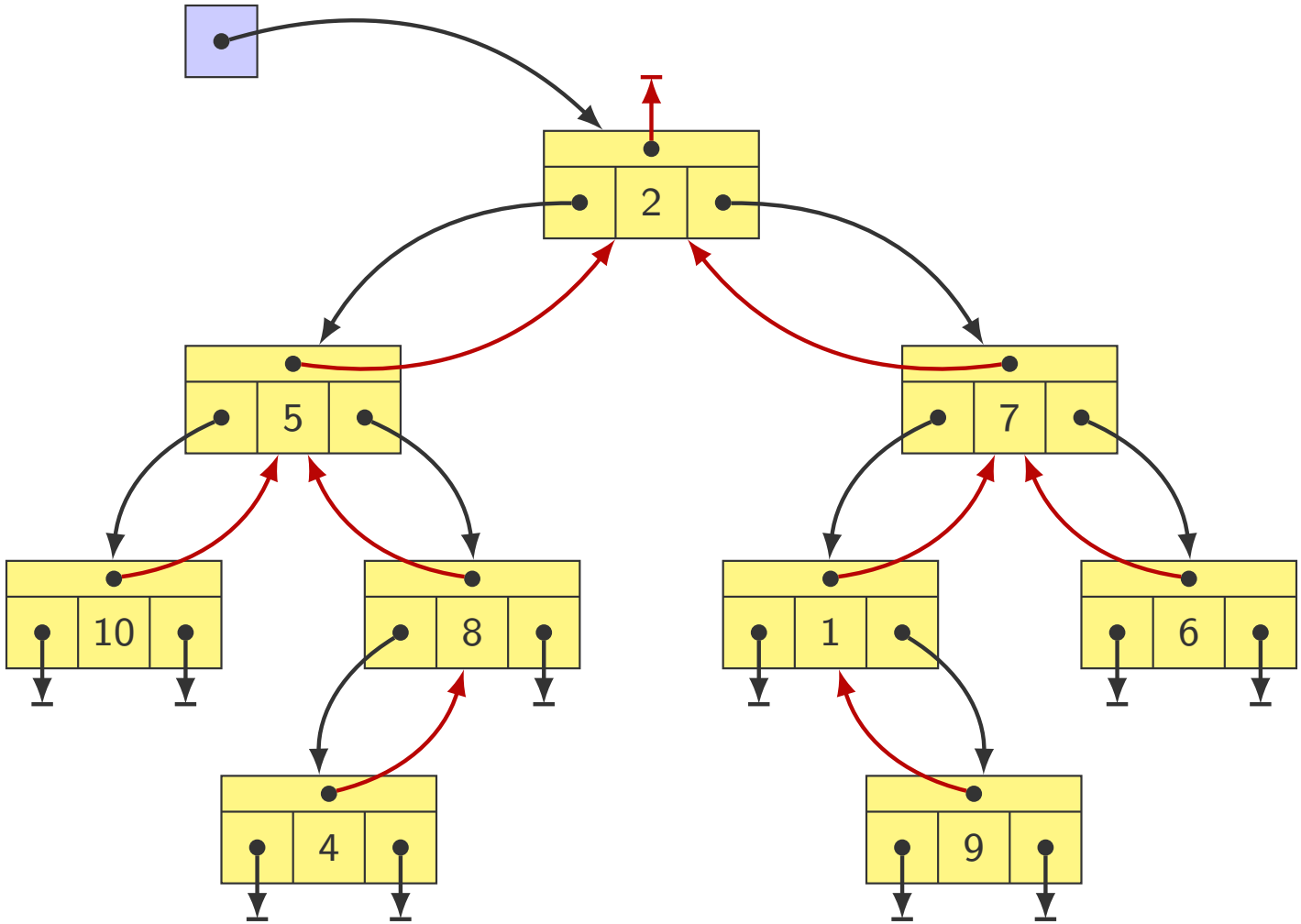
- a altura é no mínimo $\lceil \log_2(n+1) \rceil - 1$
  - $n \leq 2^{h+1} - 1 \Rightarrow n + 1 \leq 2^{h+1} \Rightarrow \log_2(n+1) \leq \log_2 2^{h+1} \Rightarrow h \geq \log_2(n+1) - 1$
  - quando a árvore é completa
- a altura é no máximo $n - 1$
  - quando cada nó interno tem apenas um filho (a árvore é um caminho)

Representação no Computador

Representação no Computador



Representação com ponteiro para pai



Representação — Decisões de projeto

- Em programação de computadores, os nós de uma árvore binária são definidos como um tipo de dado composto contendo pelo menos três atributos:
  - um valor (chave a ser guardada)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó

- Para acessarmos qualquer nó da árvore, basta termos o endereço do nó raiz. Pois podemos usar recursão para fazer todo o trabalho. Portanto, a única informação inicial necessária é um ponteiro para a raiz da árvore.

Implementação do Nó da Árvore em C

```c
struct arv_no {
    int chave;
    struct arv_no *esq;
    struct arv_no *dir;
};

typedef struct arv_no No;
```

Criando uma árvore

```
1  // Cria uma arvore vazia
2  No *arv_cria_vazia(void) {
3      return NULL;
4  }
5
6  // Cria uma arvore a partir de duas sub-arvores distintas
7  No *arv_cria(int valor, No *sae, No *sad) {
8      No *raiz = (No*) malloc(sizeof(No));
9      raiz->chave = valor;
10     raiz->esq = sae;
11     raiz->dir = sad;
12     return raiz;
13 }
```

Árvore vazia?

```
1 // Retorna true se e somente a arvore for vazia
2 bool arv_vazia(No *r) {
3     return r == NULL;
4 }
```

Imprime os nós

```
1  // Imprime as chaves de todos os nos da arvore
2  void arv_imprime(No *r) {
3      printf("(");
4      if(!arv_vazia(r)) {
5          printf("%d ", r->chave);
6          arv_imprime(r->esq);
7          arv_imprime(r->dir);
8      }
9      printf(")");
10 }
```

Libera os nós

```
1 // Libera todos os nos da arvore
2 No* arv_libera(No *r) {
3     if(!arv_vazia(r)) {
4         arv_libera(r->esq);
5         arv_libera(r->dir);
6         free(r);
7     }
8     return NULL;
9 }
```

Verificar pertencimento

```
1 // Verifica se um valor esta contido na arvore
2 bool arv_pertence(No *r, int valor) {
3     if(arv_vazia(r))
4         return false;
5     else
6         return r->chave == valor ||
7                arv_pertence(r->esq, valor) ||
8                arv_pertence(r->dir, valor);
9 }
```

Programa cliente

```c
#include <stdio.h> // main.c
#include "Tree.h"

int main() {
    No *a = arv_cria(12,
        arv_cria(23,
            arv_cria_vazia(),
            arv_cria(45, arv_cria_vazia(), arv_cria_vazia())
        ),
        arv_cria(77,
            arv_cria(99, arv_cria_vazia(), arv_cria_vazia()),
            arv_cria(88, arv_cria_vazia(), arv_cria_vazia())
        )
    );

    arv_imprime(a);
    printf("\n");
    arv_libera(a);
    return 0;
}
```

Percursos em Árvores Binárias

Percursos em Árvores Binárias

- Muitas operações em árvores binárias envolvem o percurso de todas as subárvores, com execução de alguma ação de tratamento em cada nó.

Percursos em Árvores Binárias

- Muitas operações em árvores binárias envolvem o percurso de todas as subárvores, com execução de alguma ação de tratamento em cada nó.

- É comum percorrer uma árvore em uma das seguintes ordens:
  - pré-ordem:
    - **visita raiz**, percorre `r->left`, percorre `r->right`

Percursos em Árvores Binárias

- Muitas operações em árvores binárias envolvem o percurso de todas as subárvores, com execução de alguma ação de tratamento em cada nó.

- É comum percorrer uma árvore em uma das seguintes ordens:
  - pré-ordem:
    - **visita raiz**, percorre `r->left`, percorre `r->right`
  - ordem simétrica:
    - percorre `r->left`, **visita raiz**, percorre `r->right`

Percursos em Árvores Binárias

- Muitas operações em árvores binárias envolvem o percurso de todas as subárvores, com execução de alguma ação de tratamento em cada nó.

- É comum percorrer uma árvore em uma das seguintes ordens:
  - pré-ordem:
    - **visita raiz**, percorre `r->left`, percorre `r->right`
  - ordem simétrica:
    - percorre `r->left`, **visita raiz**, percorre `r->right`
  - pós-ordem:
    - percorre `r->left`, percorre `r->right`, **visita raiz**

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

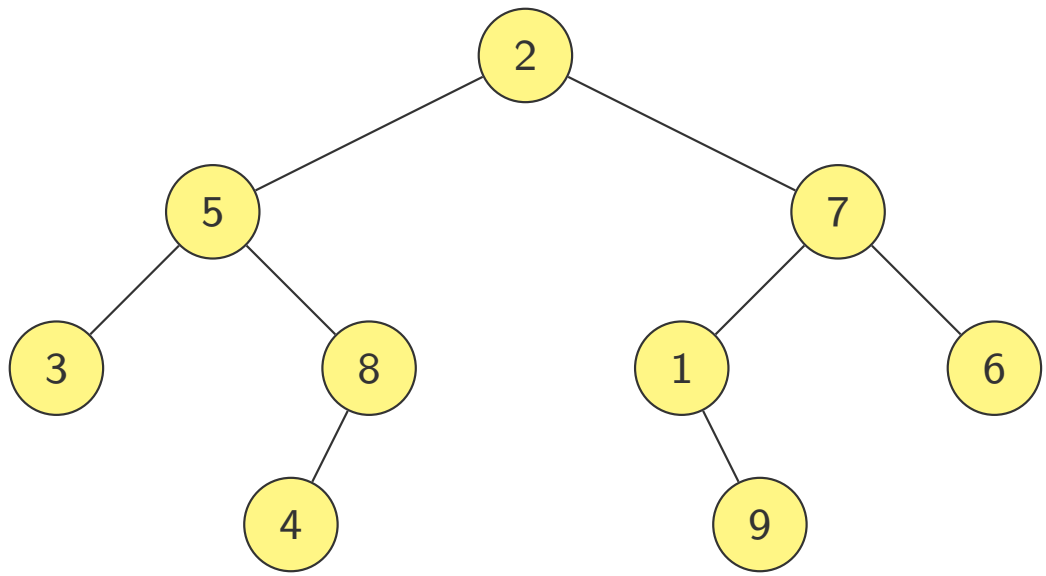
A pré-ordem

- 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

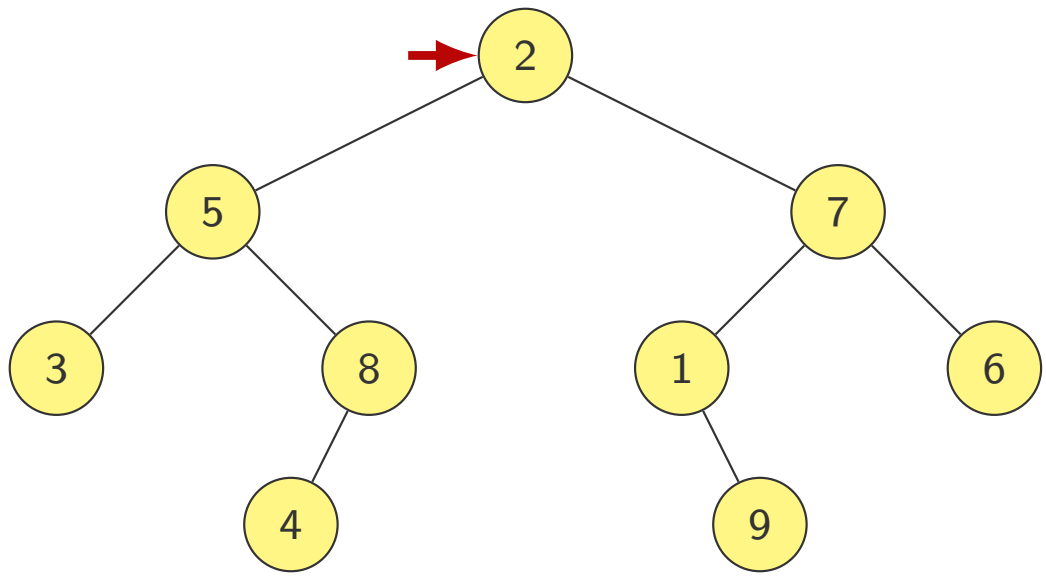


Ex:

Percorrendo os nós — Pré-ordem

A pré-ordem

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

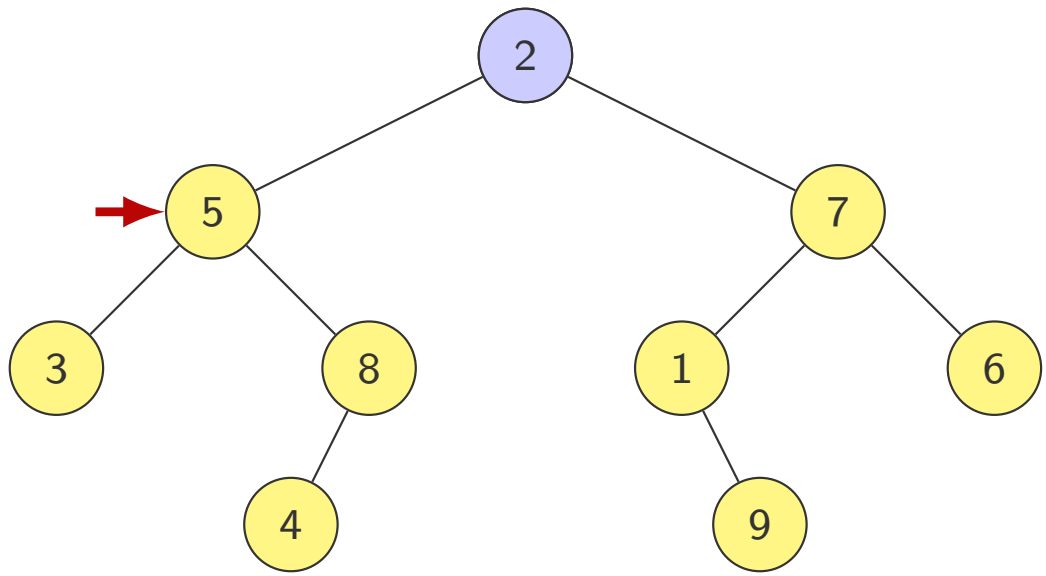


Ex:

Percorrendo os nós — Pré-ordem

A pré-ordem

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

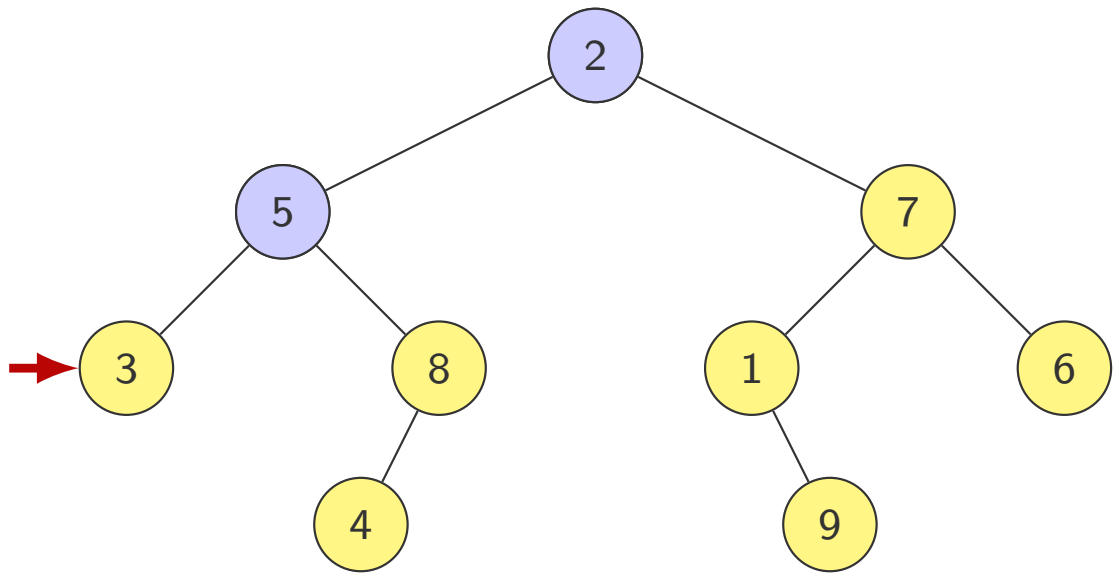


Ex: 2,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

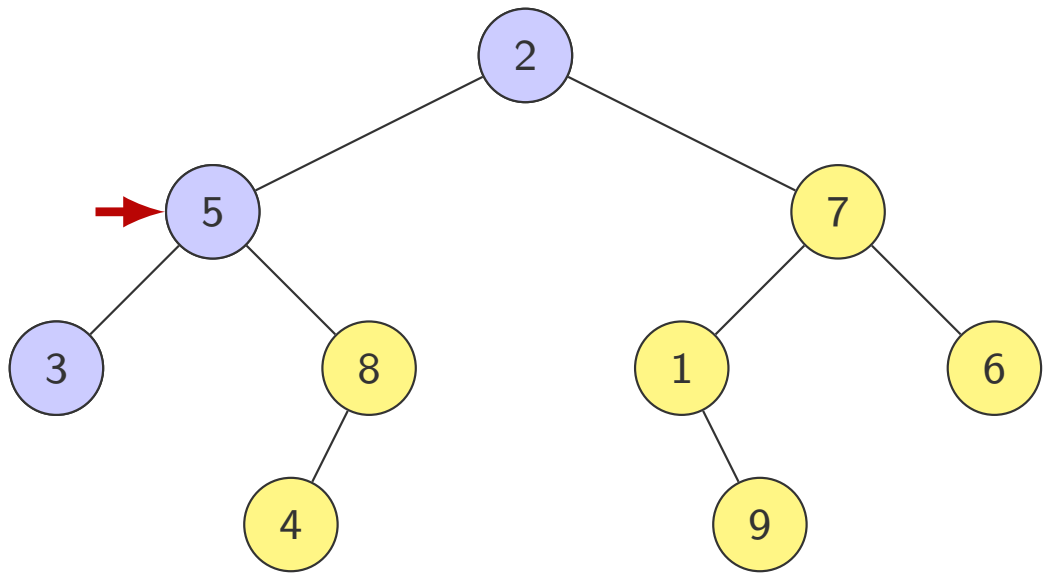


Ex: 2, 5,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

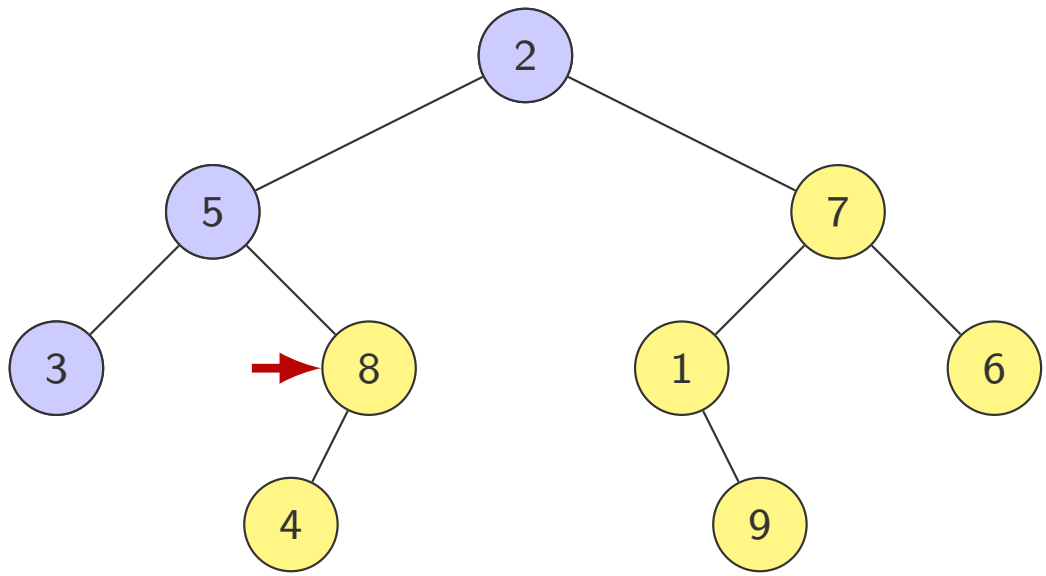


Ex: 2, 5, 3,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

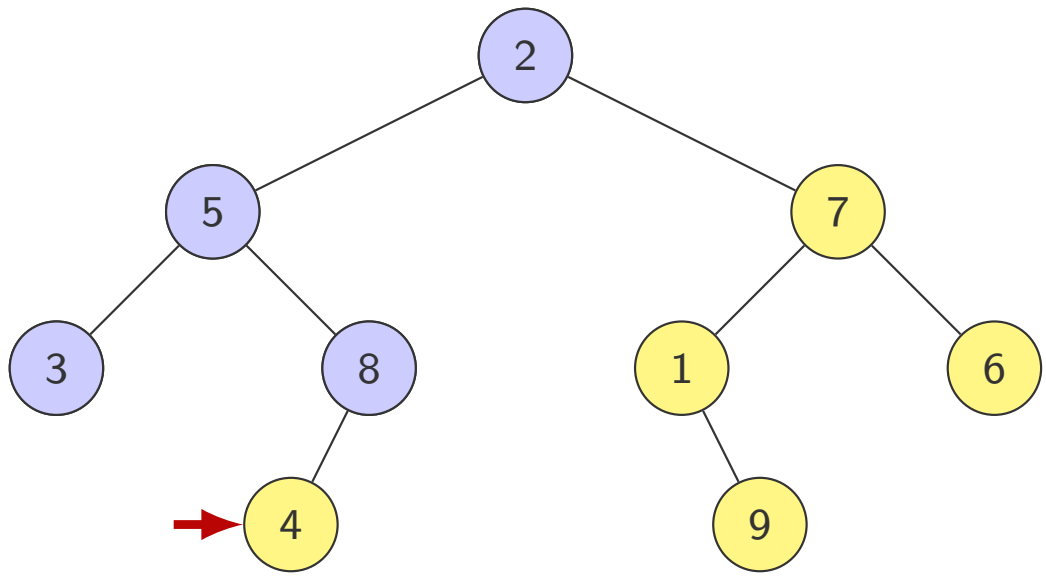


Ex: 2, 5, 3,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

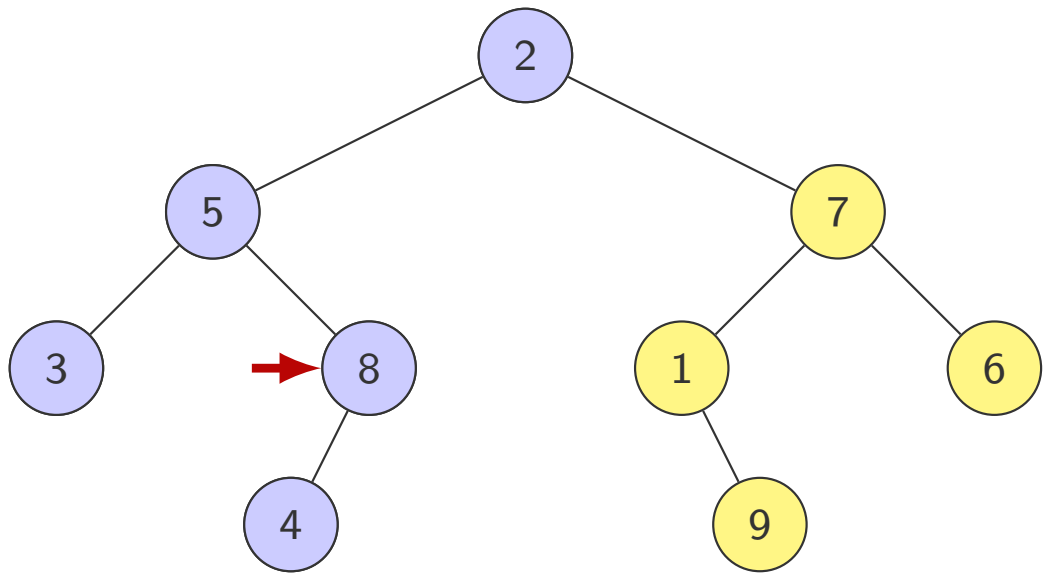


Ex: 2, 5, 3, 8,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

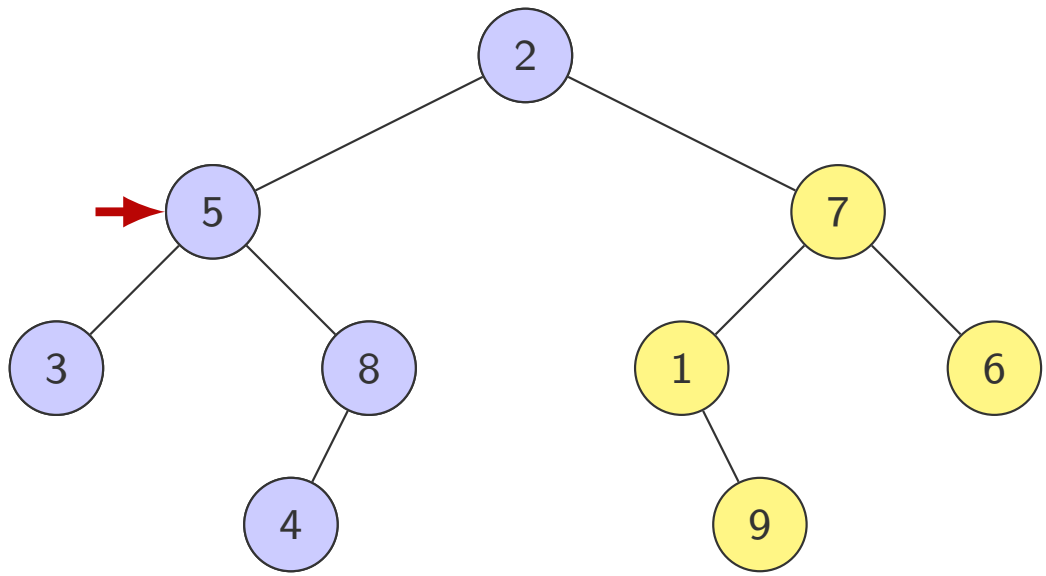


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

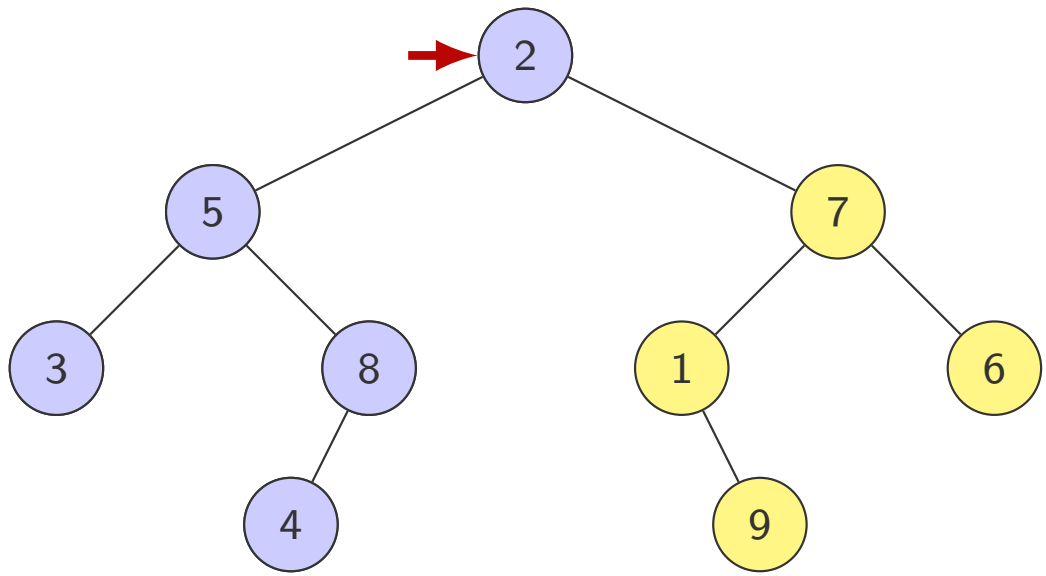


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

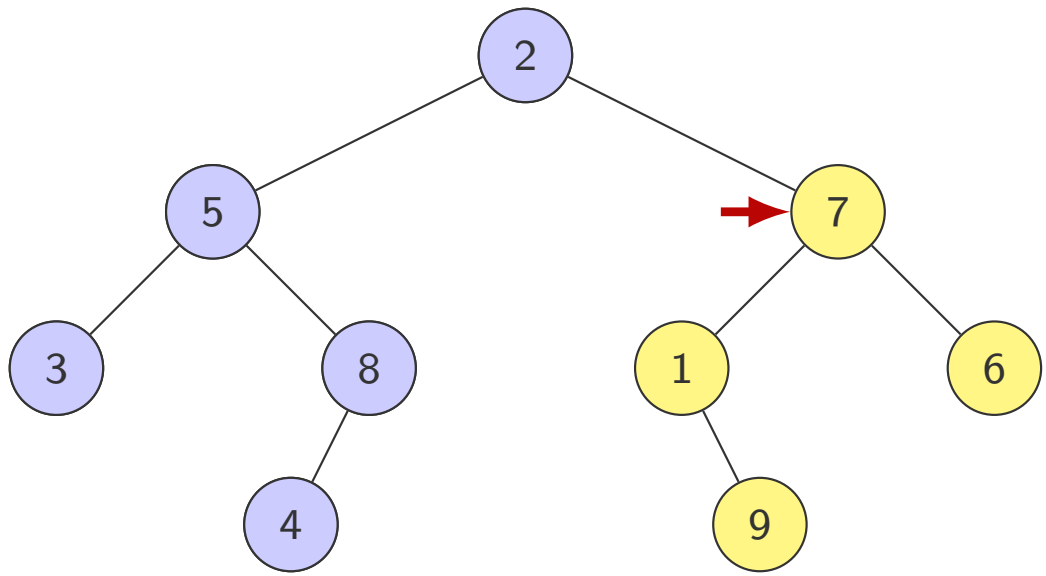


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

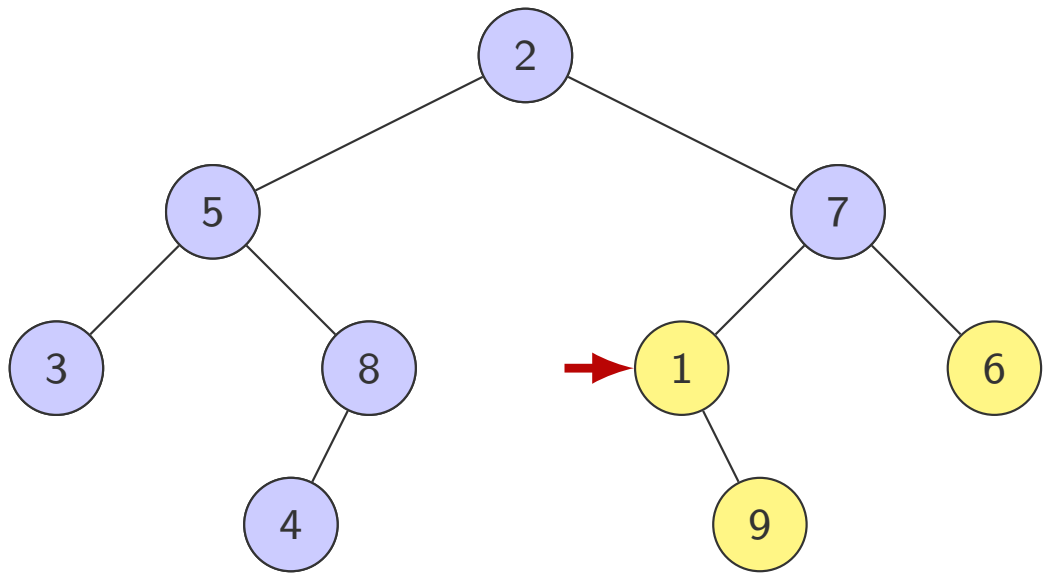


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

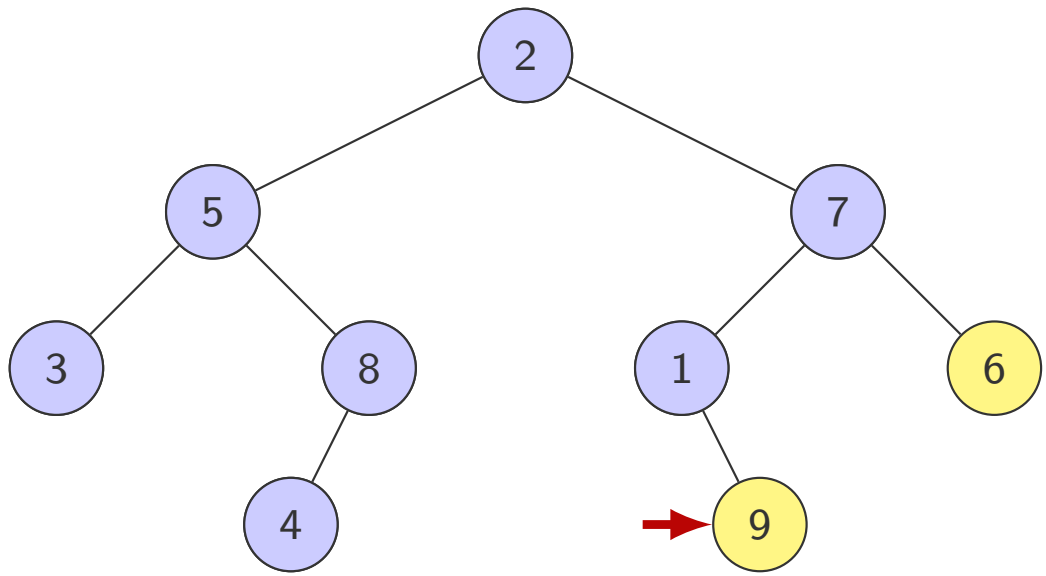


Ex: 2, 5, 3, 8, 4, 7,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

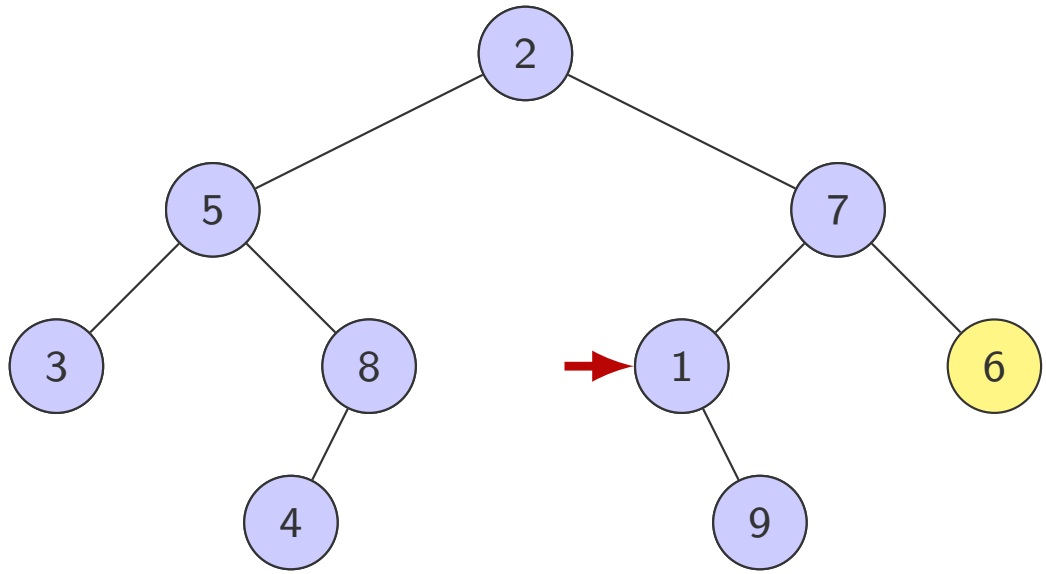


Ex: 2, 5, 3, 8, 4, 7, 1,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

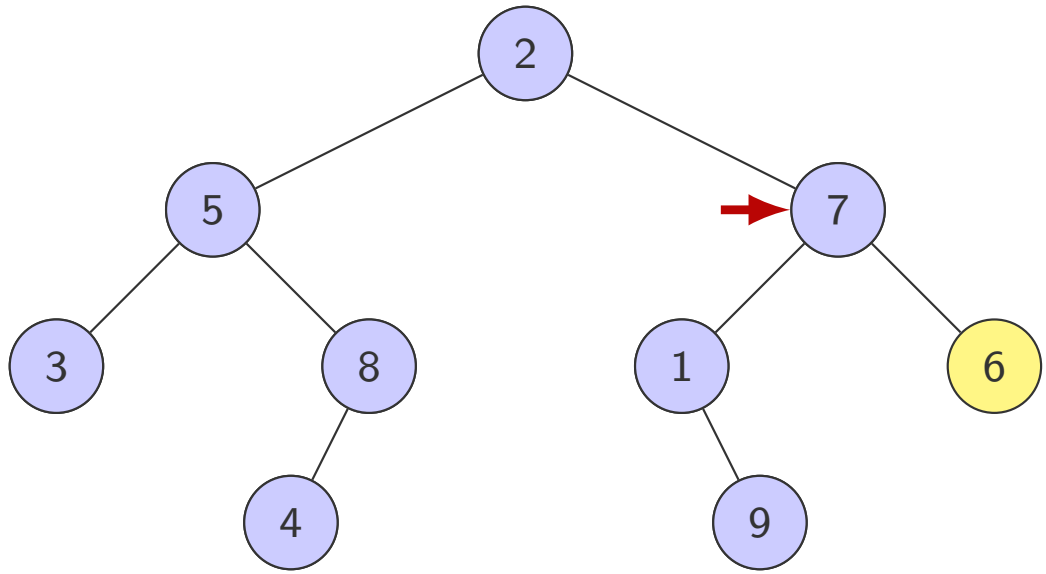


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

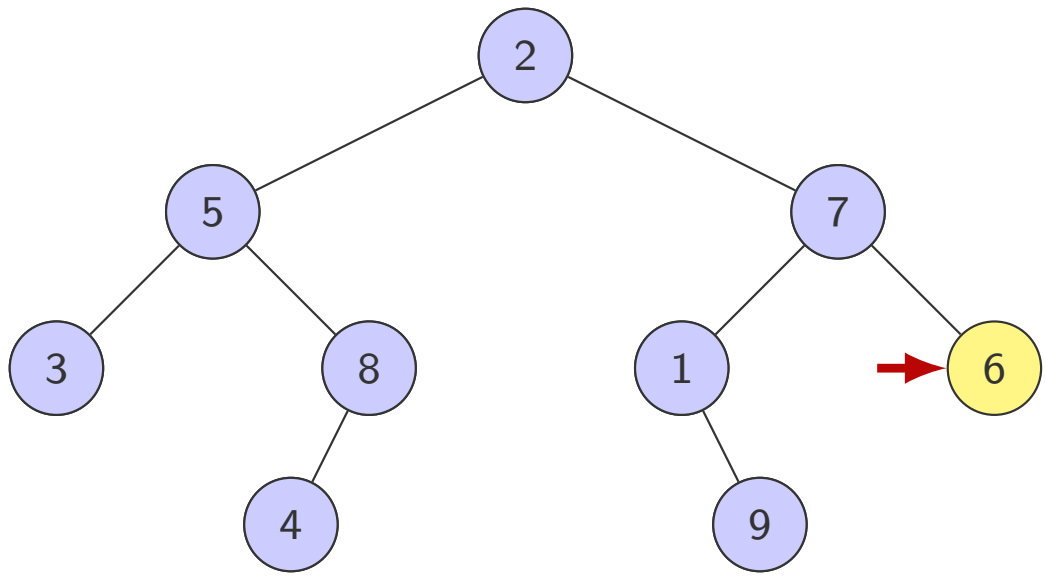


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

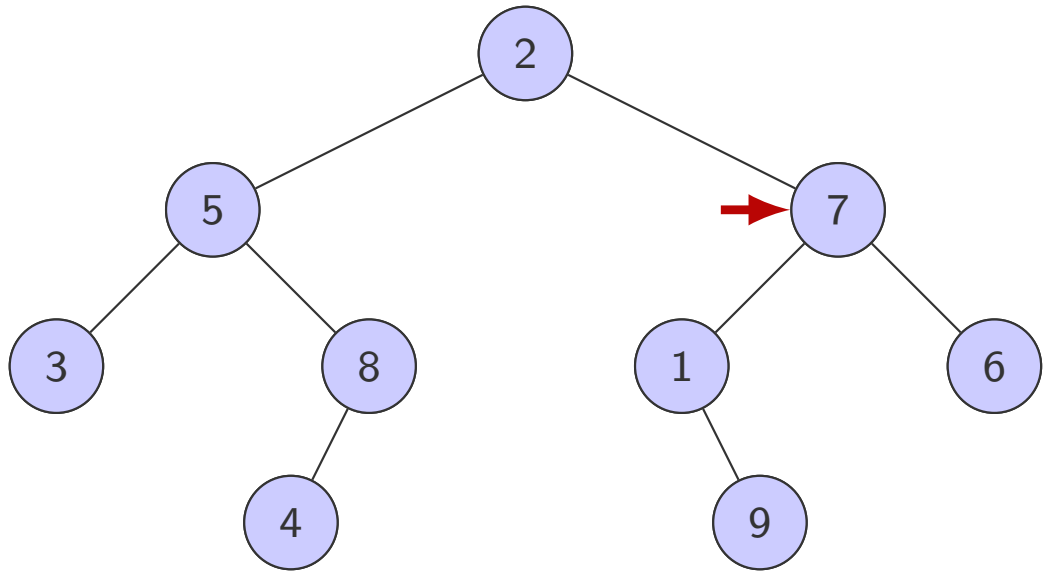


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

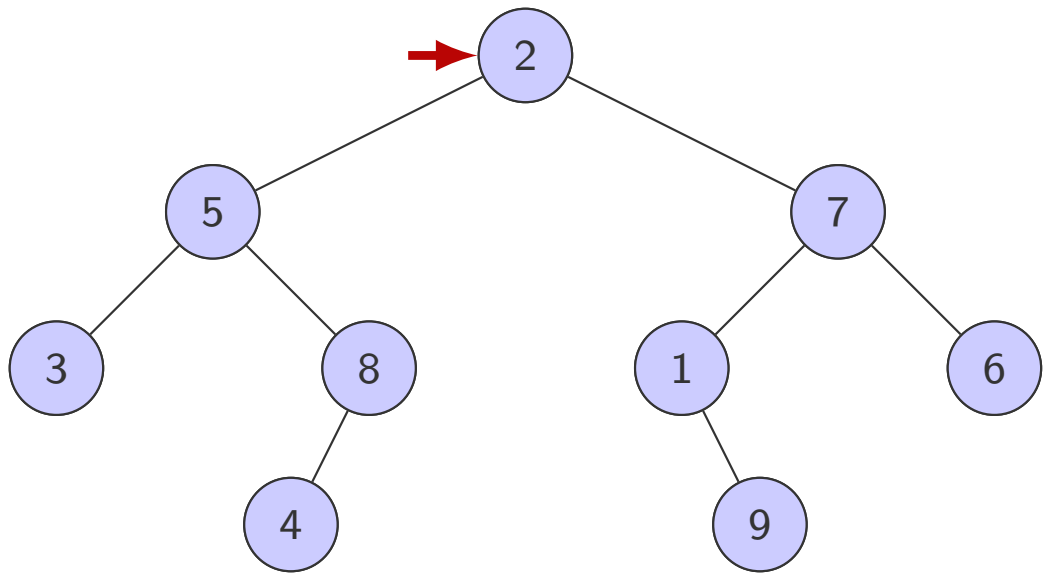


Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

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

Pré-ordem

```
1 // Percurso em pre-ordem
2 void arv_preordem(No *r) {
3     if(!arv_vazia(r)) {
4         printf("%d ", r->chave);
5         arv_preordem(r->esq);
6         arv_preordem(r->dir);
7     }
8 }
```

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

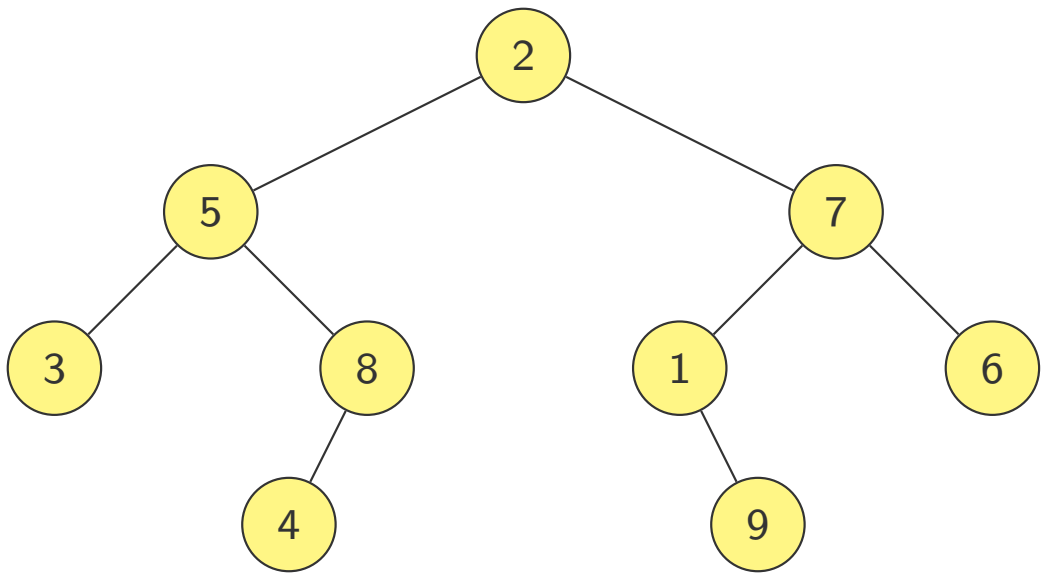
A pós-ordem

- 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

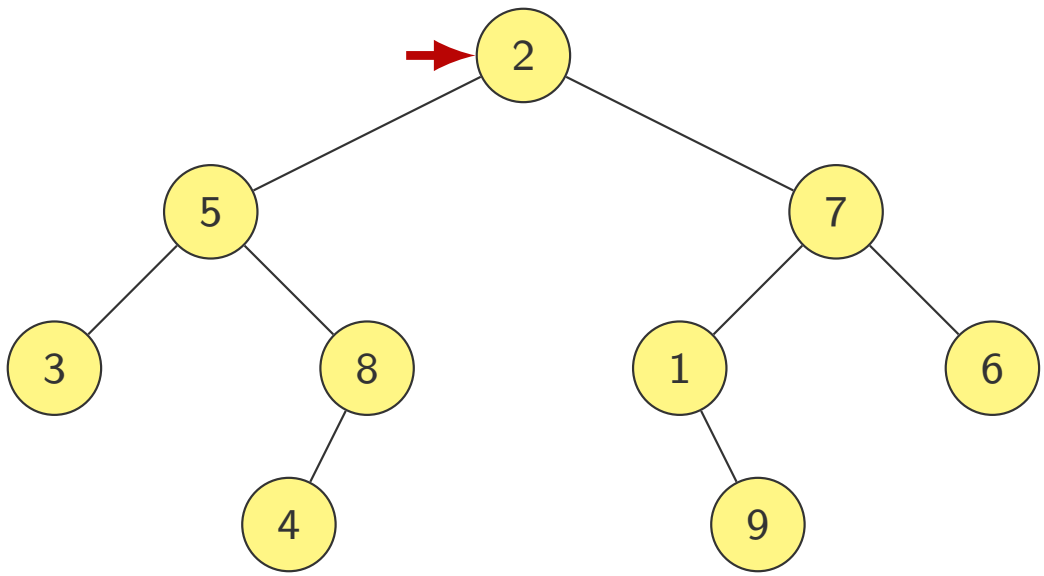


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

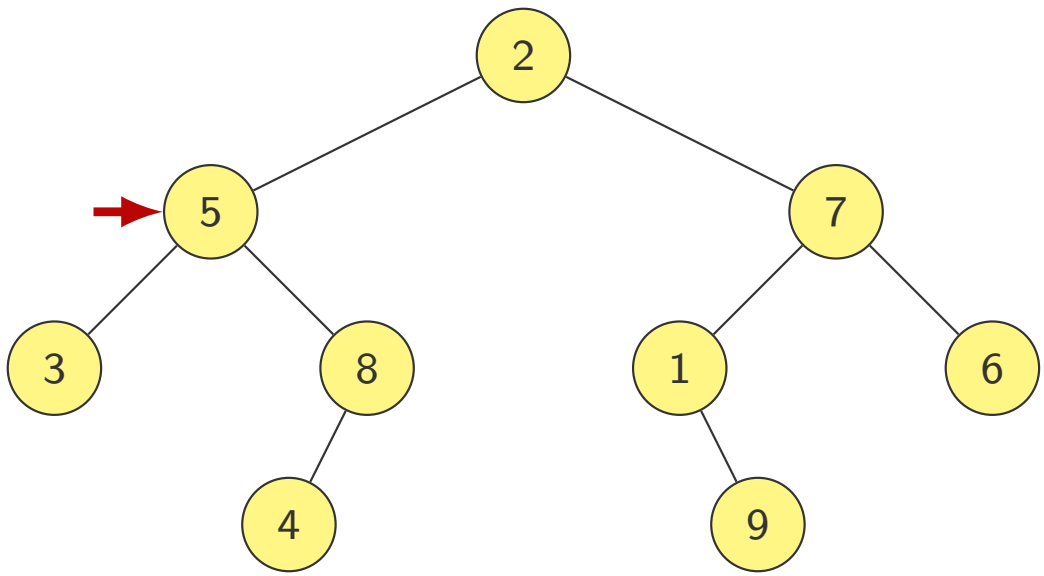


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

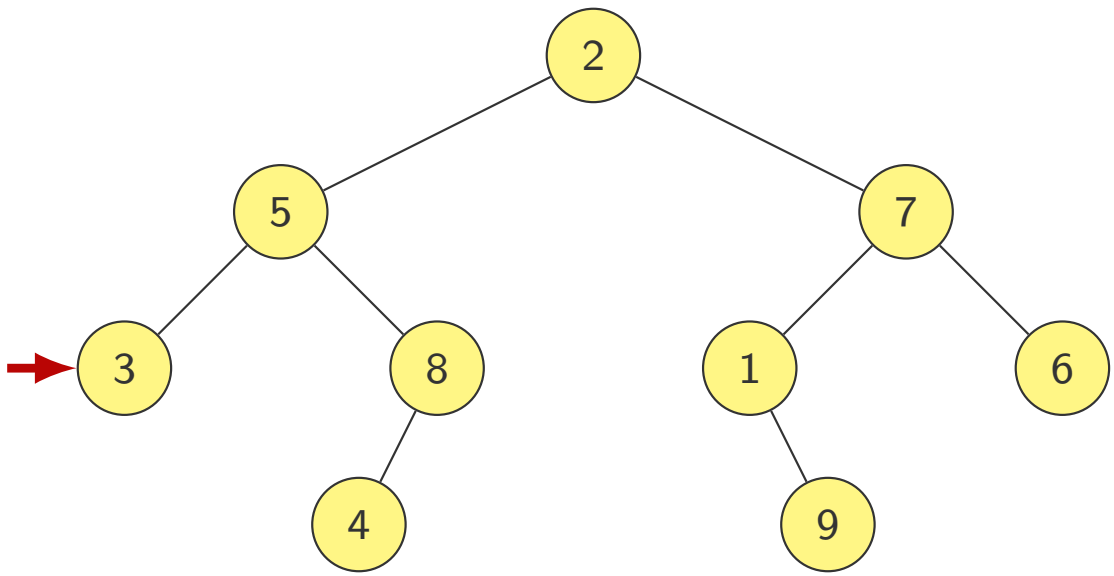


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

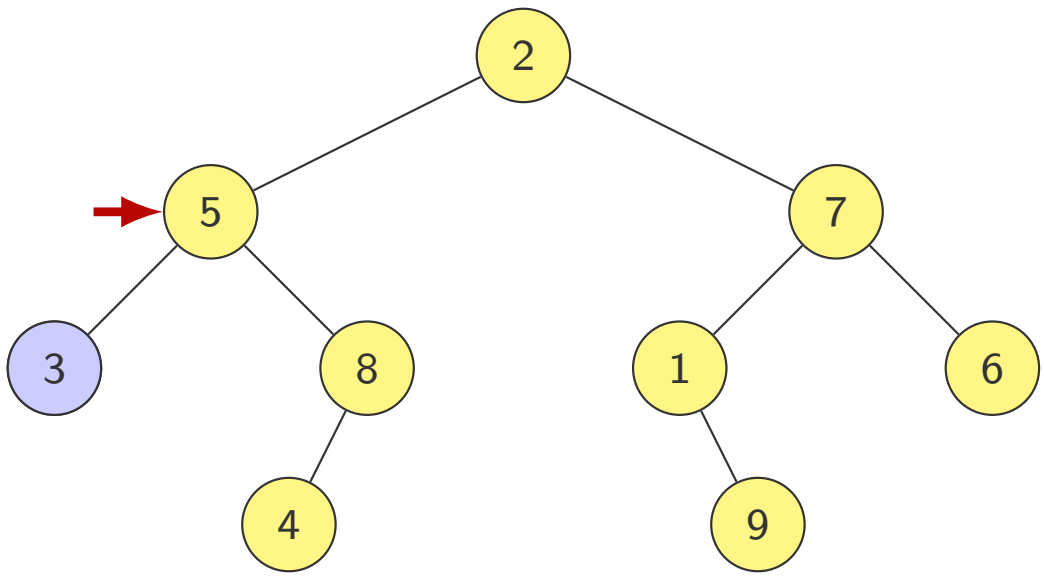


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

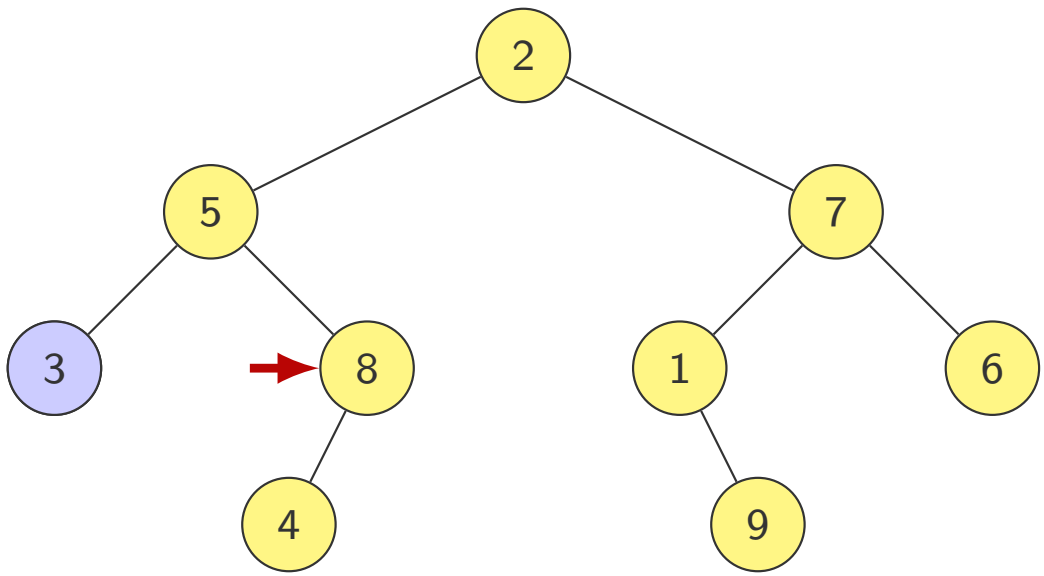


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

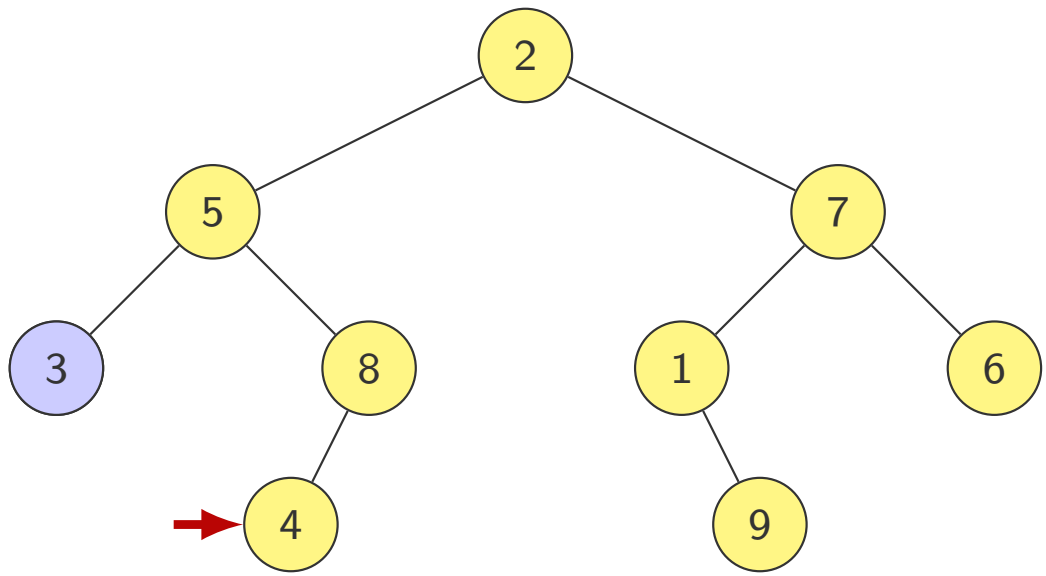


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

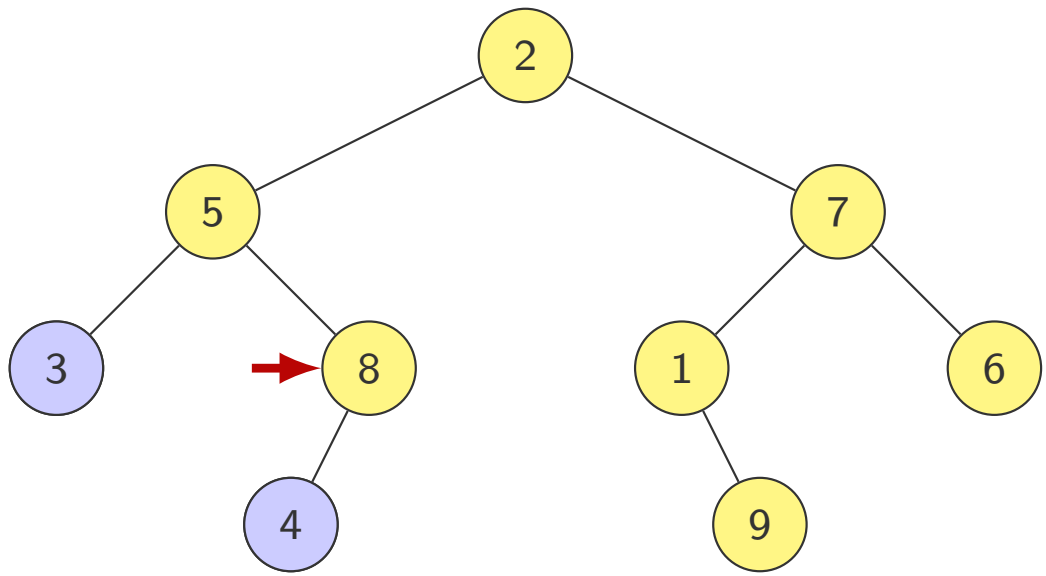


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

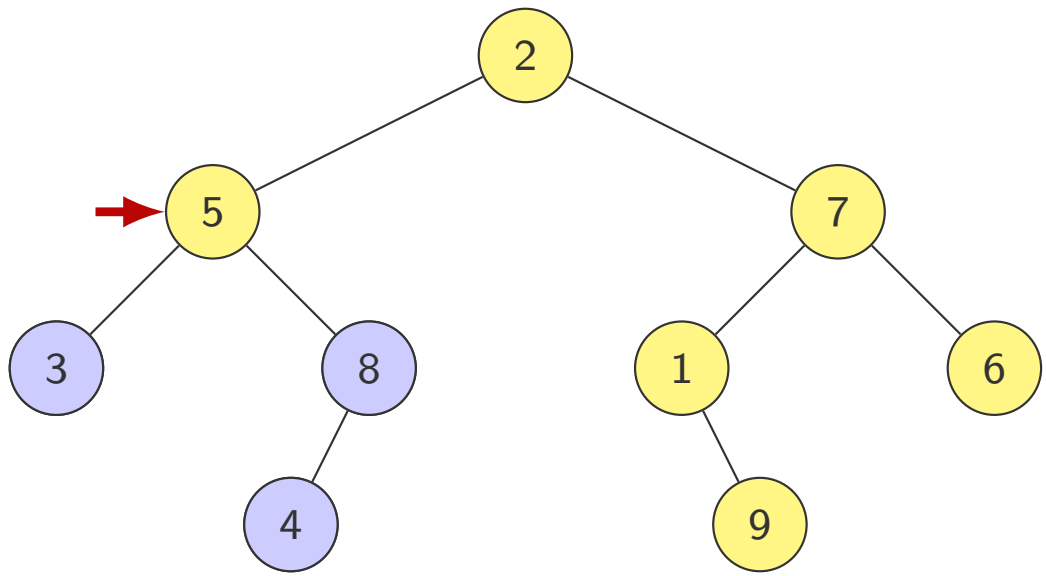


Ex: 3, 4,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

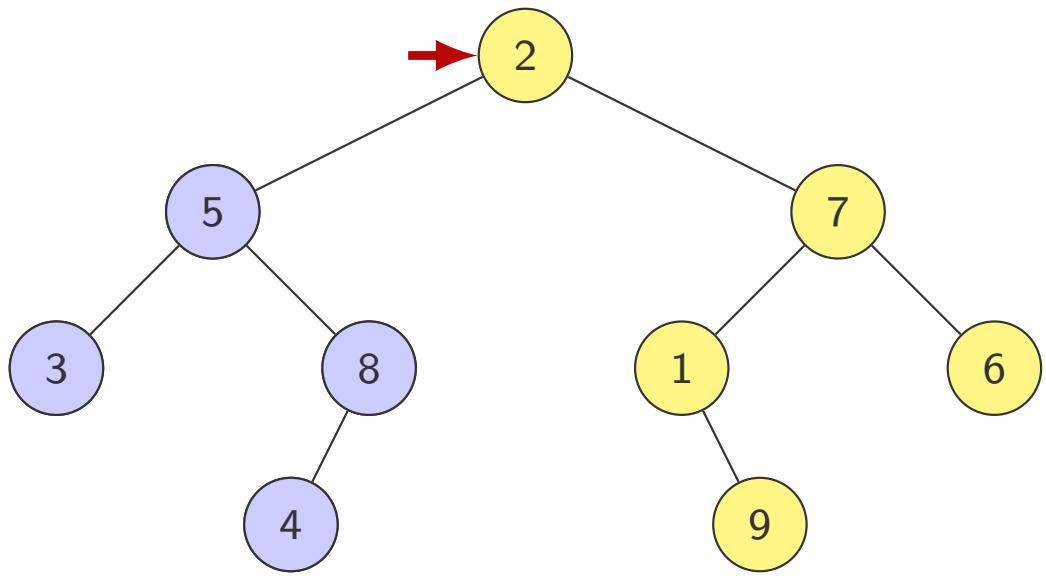


Ex: 3, 4, 8,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

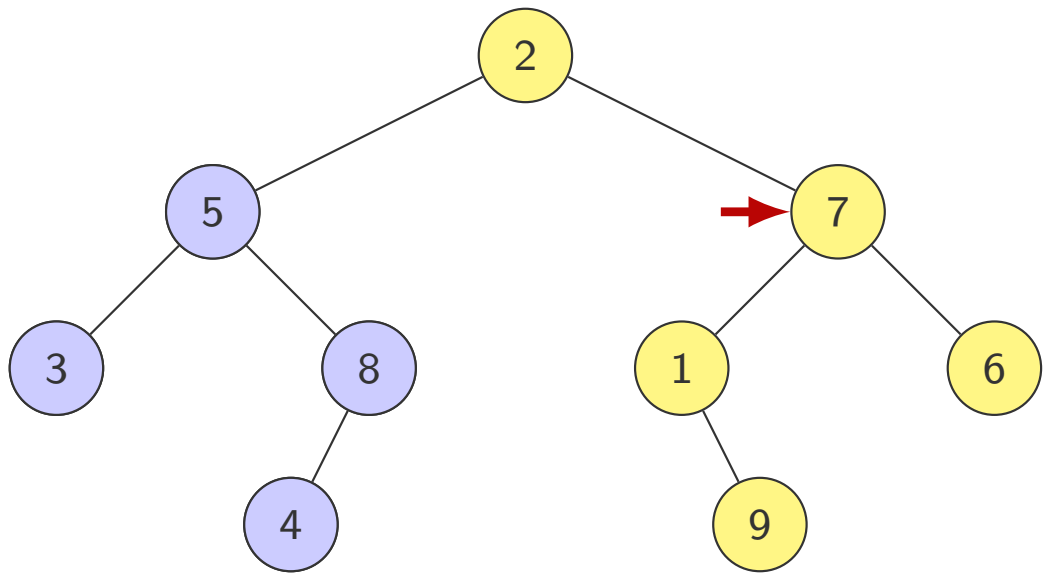


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

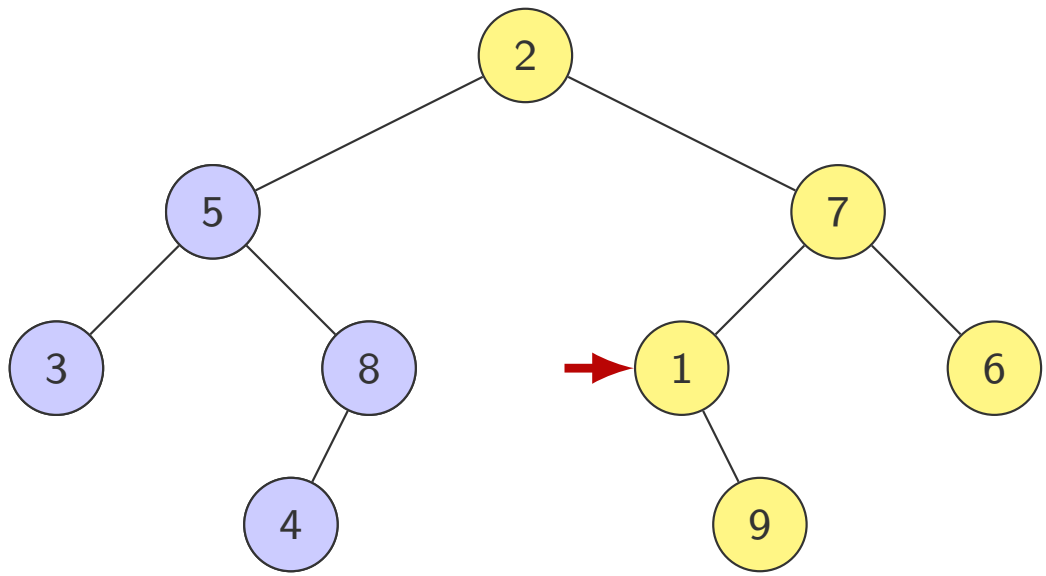


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

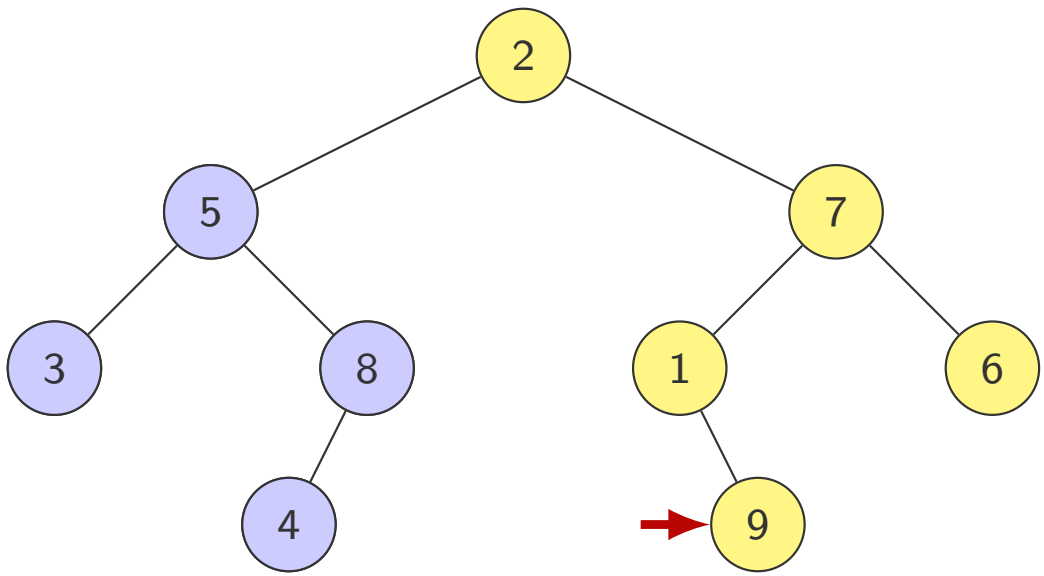


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

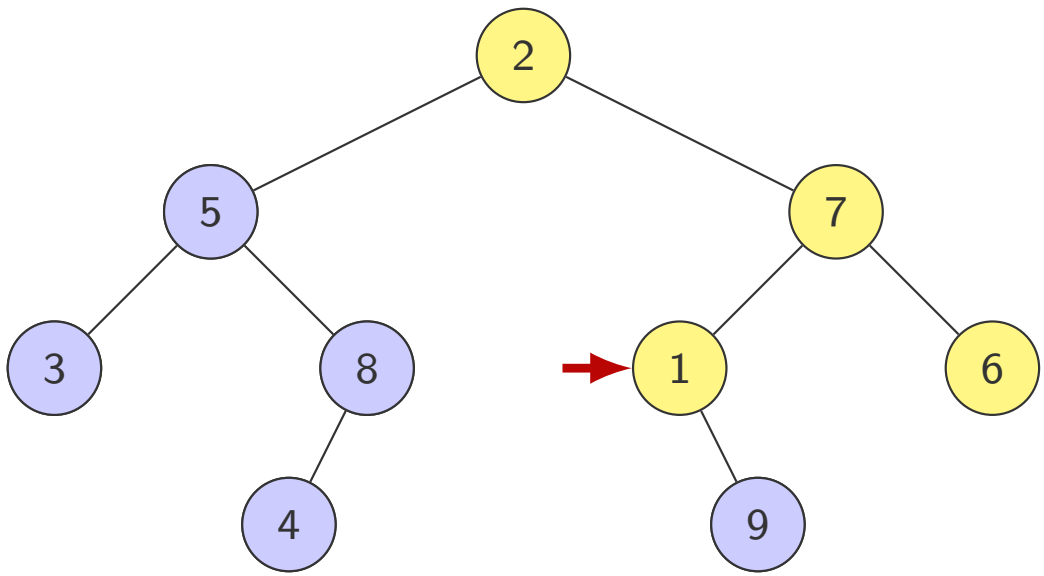


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

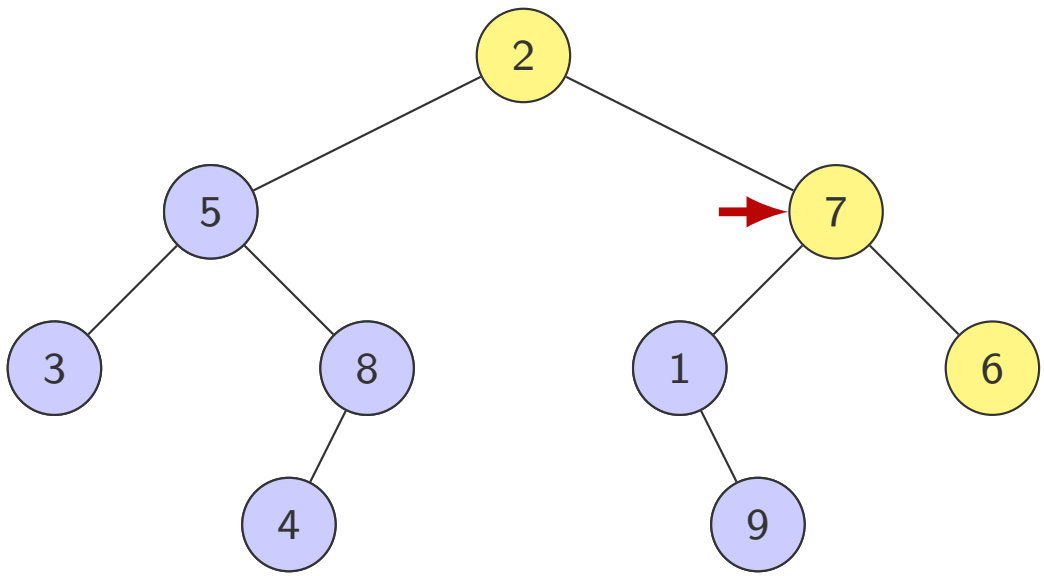


Ex: 3, 4, 8, 5, 9,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

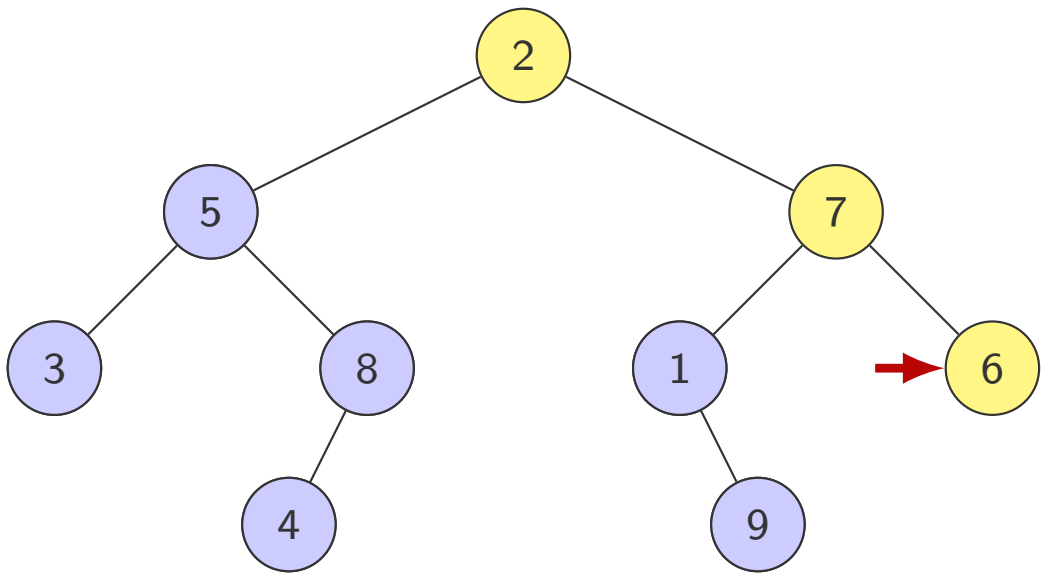


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

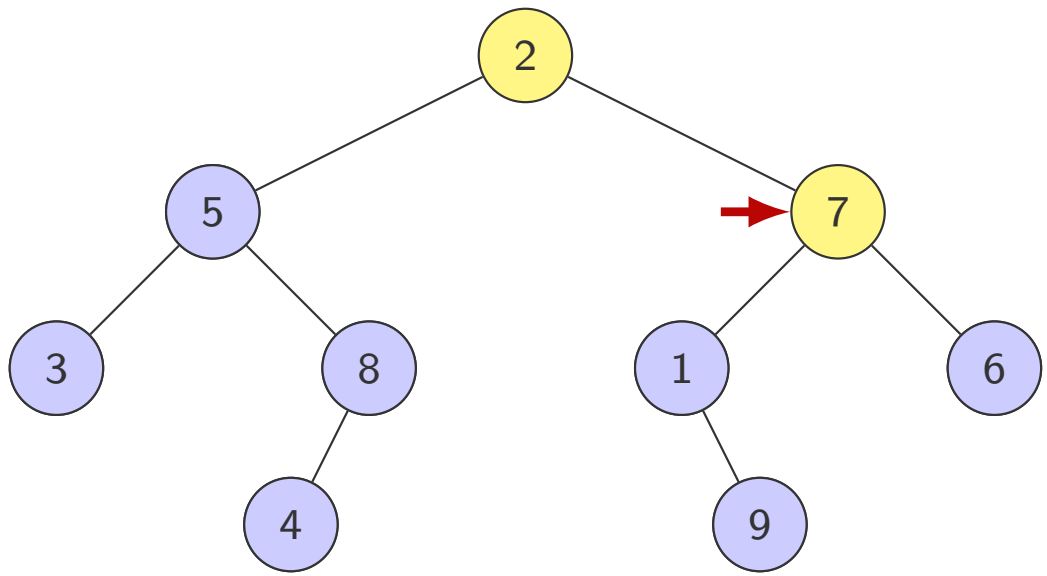


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

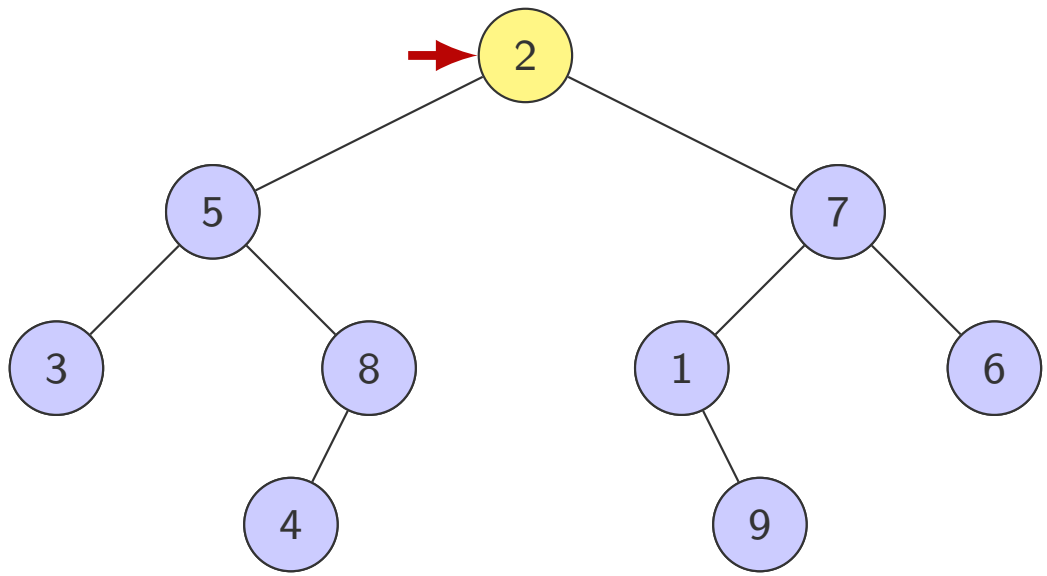


Ex: 3, 4, 8, 5, 9, 1, 6,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

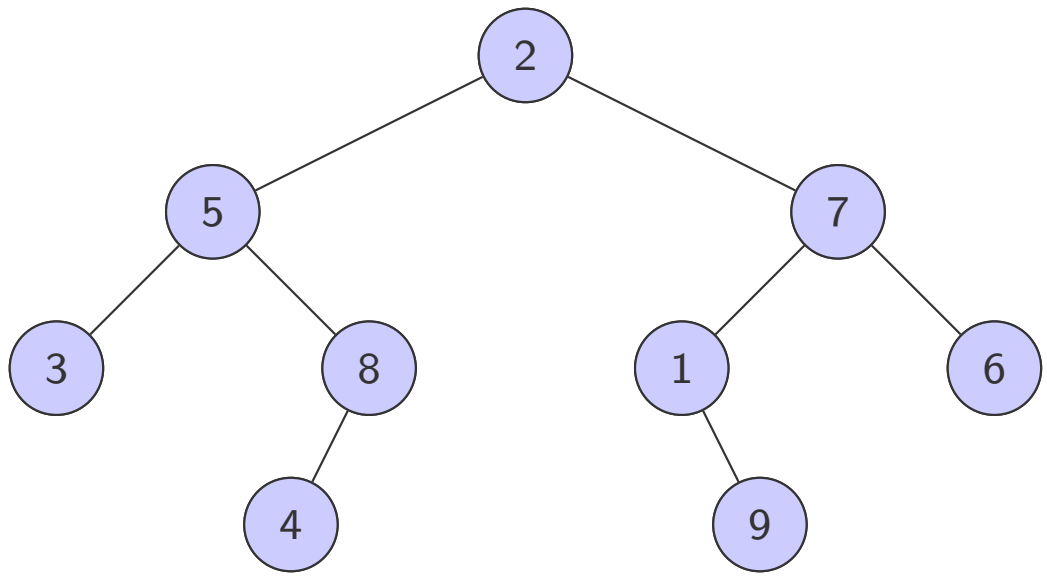


Ex: 3, 4, 8, 5, 9, 1, 6, 7,

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

Pós-ordem

```
1 // Percurso pos-ordem
2 void arv_posordem(No *r) {
3     if(!arv_vazia(r)) {
4         arv_posordem(r->esq);
5         arv_posordem(r->dir);
6         printf("%d ", r->chave);
7     }
8 }
```

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

- primeiro visita a subárvore esquerda

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

- primeiro visita a subárvore esquerda
- depois visita a raiz

Percorrendo os nós — Ordem Simétrica

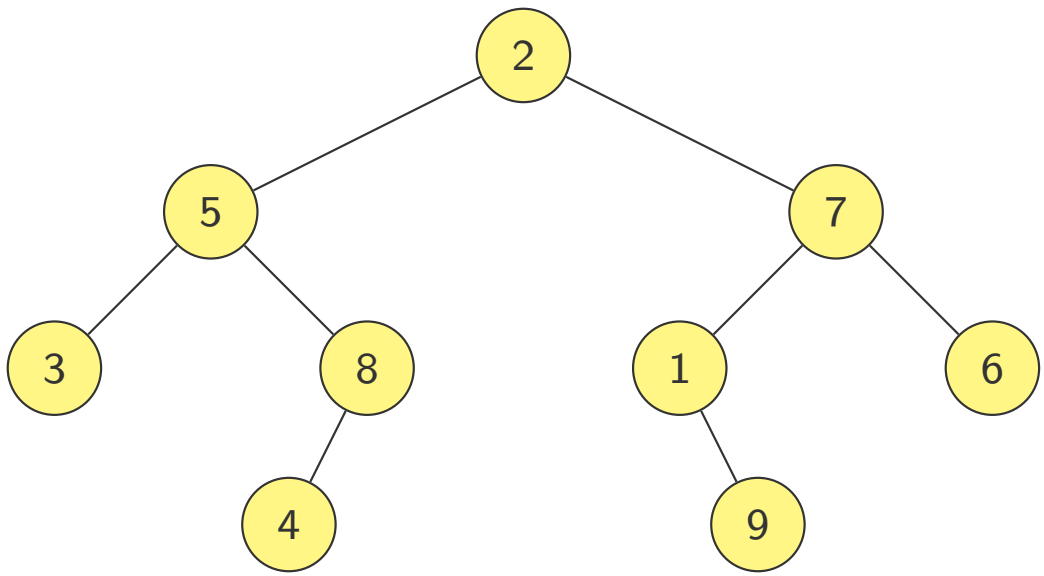
A ordem simétrica

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

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

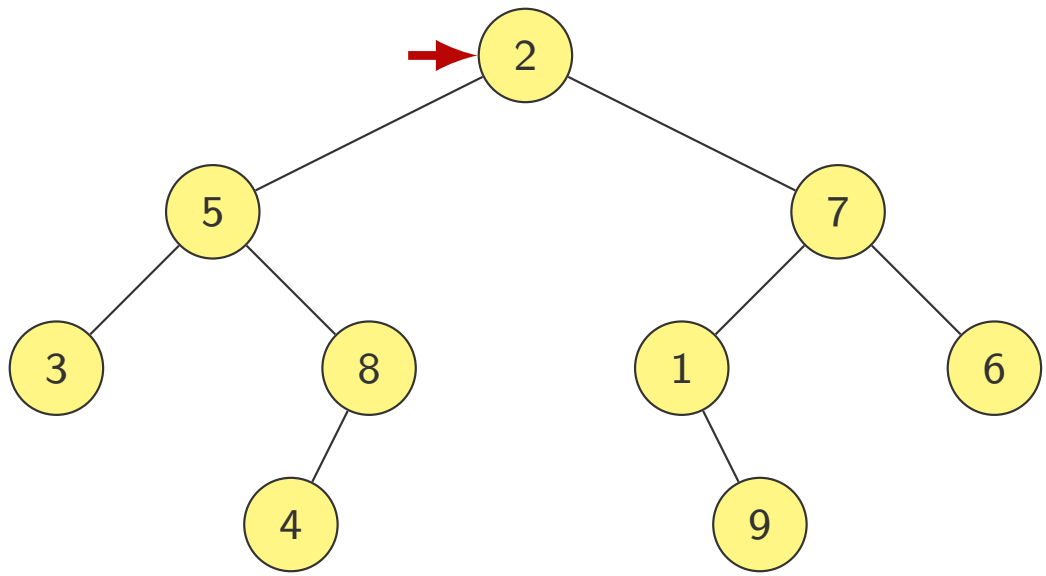


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

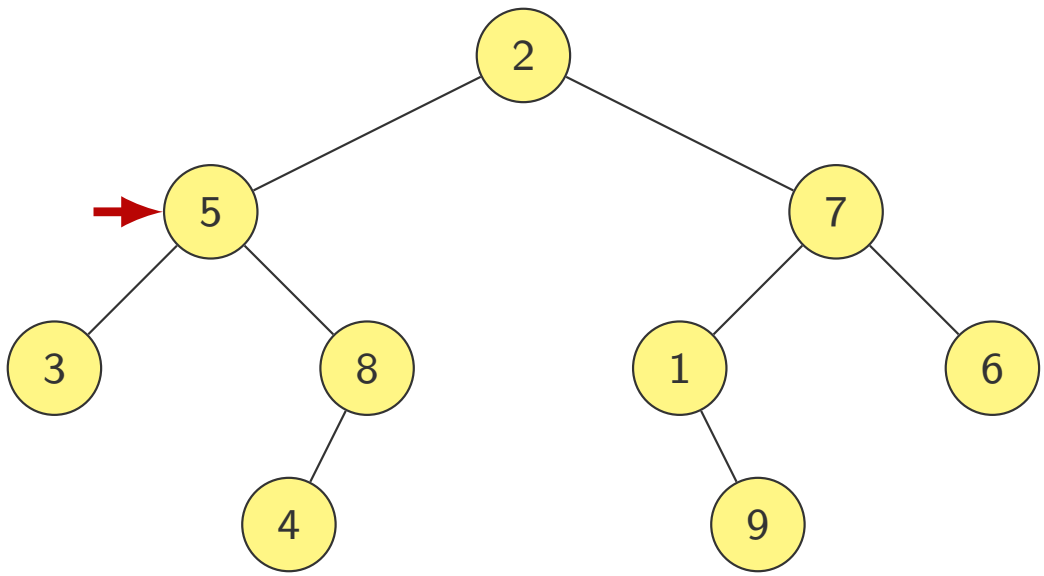


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

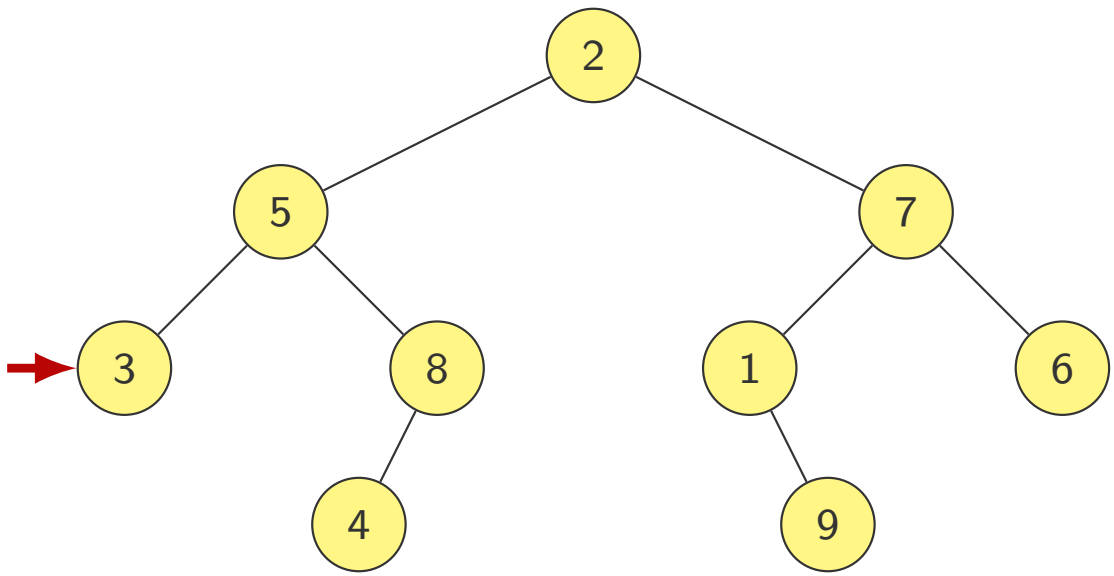


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

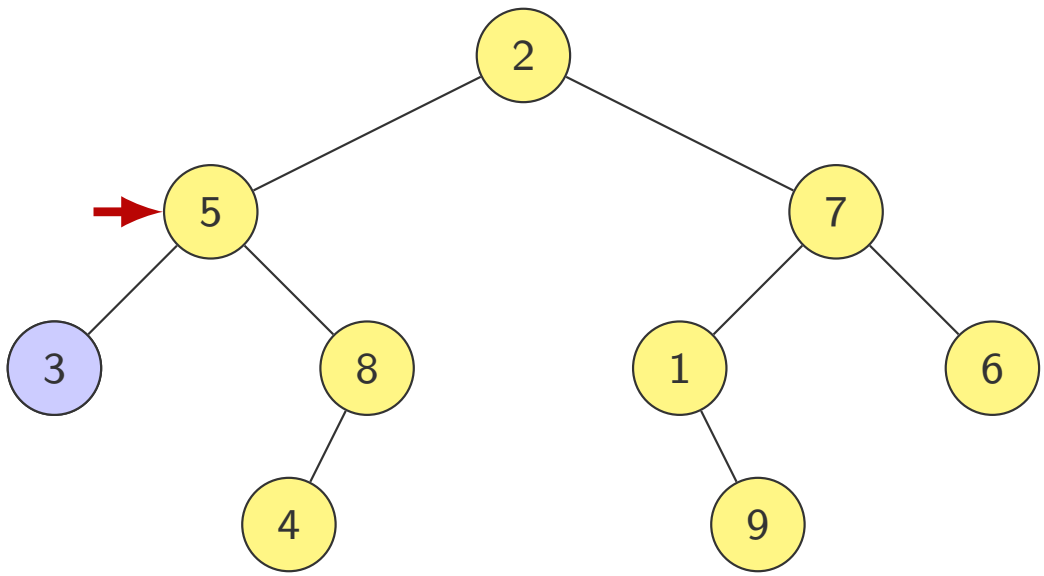


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

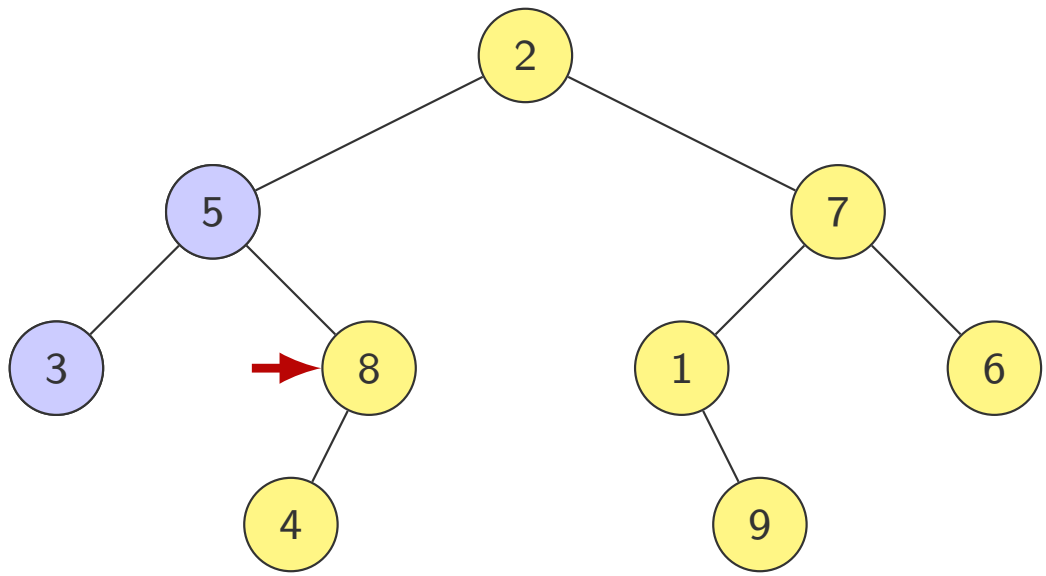


Ex: 3,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

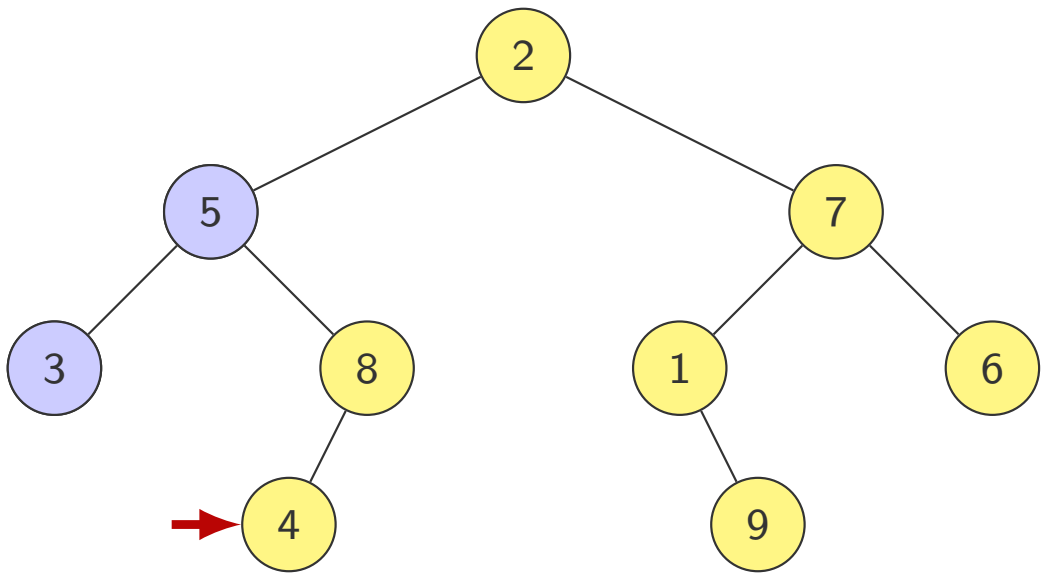


Ex: 3, 5,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

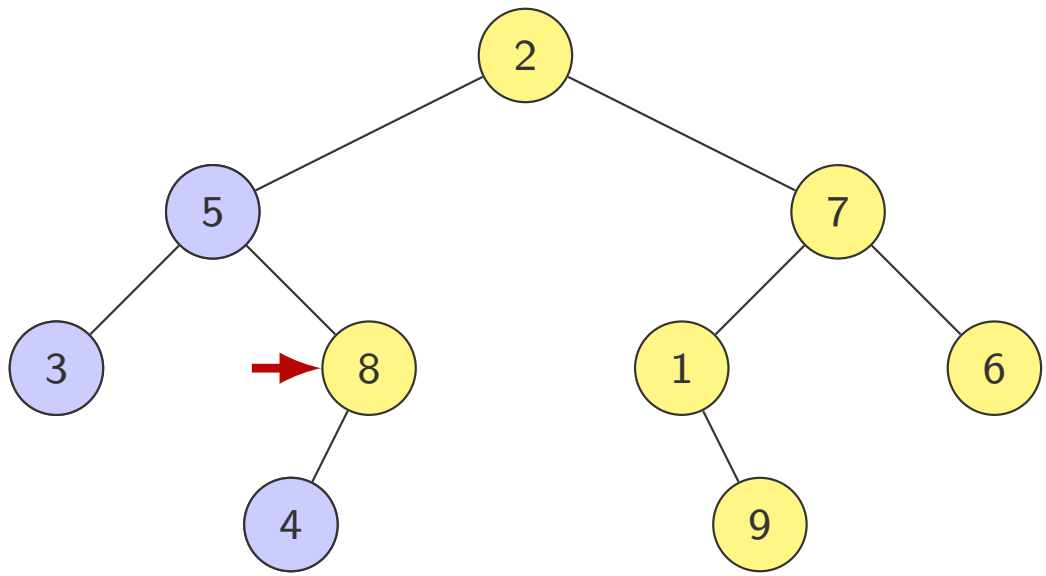


Ex: 3, 5,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

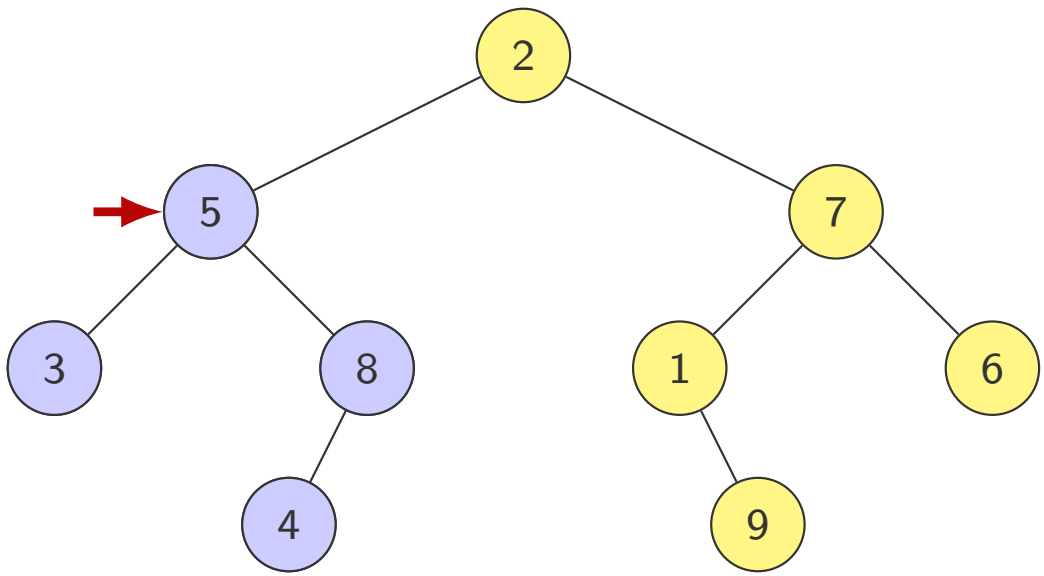


Ex: 3, 5, 4,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

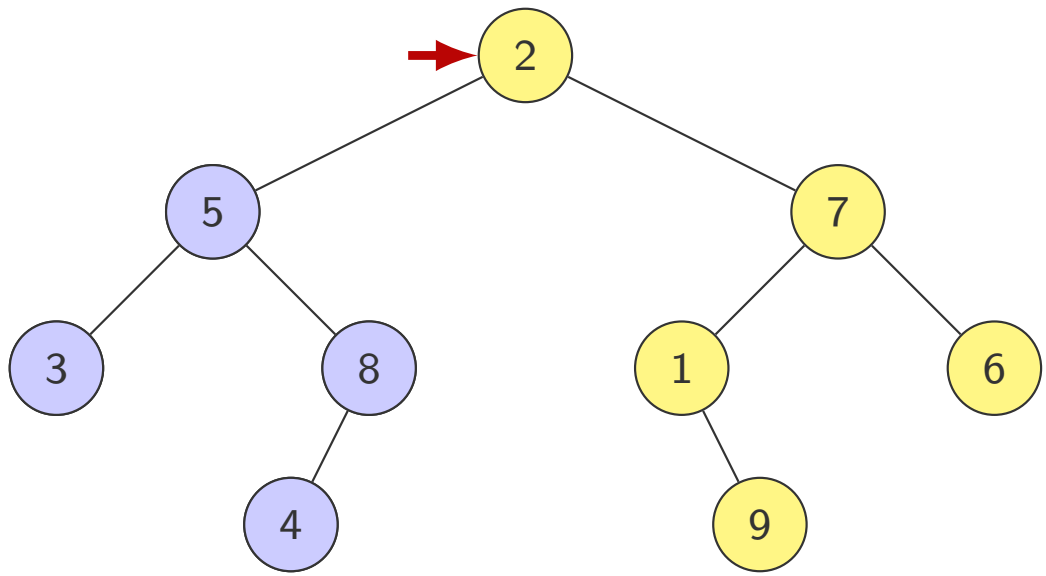


Ex: 3, 5, 4, 8,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

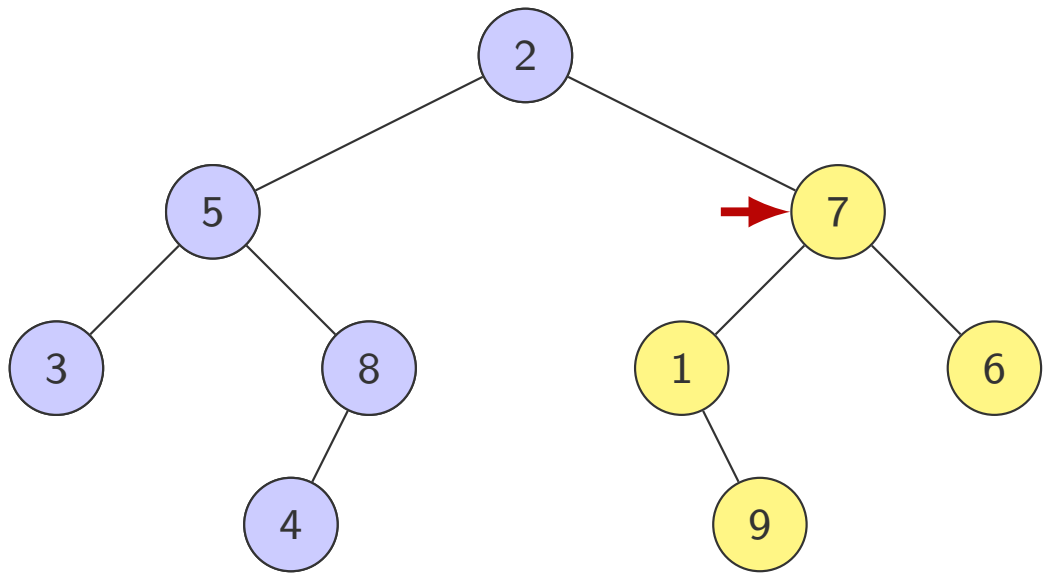


Ex: 3, 5, 4, 8,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

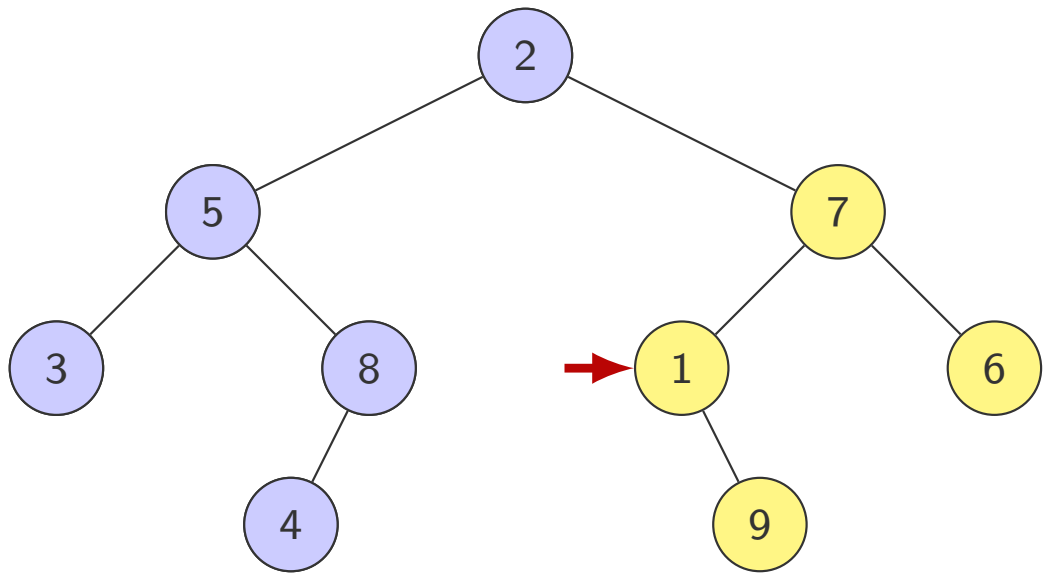


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

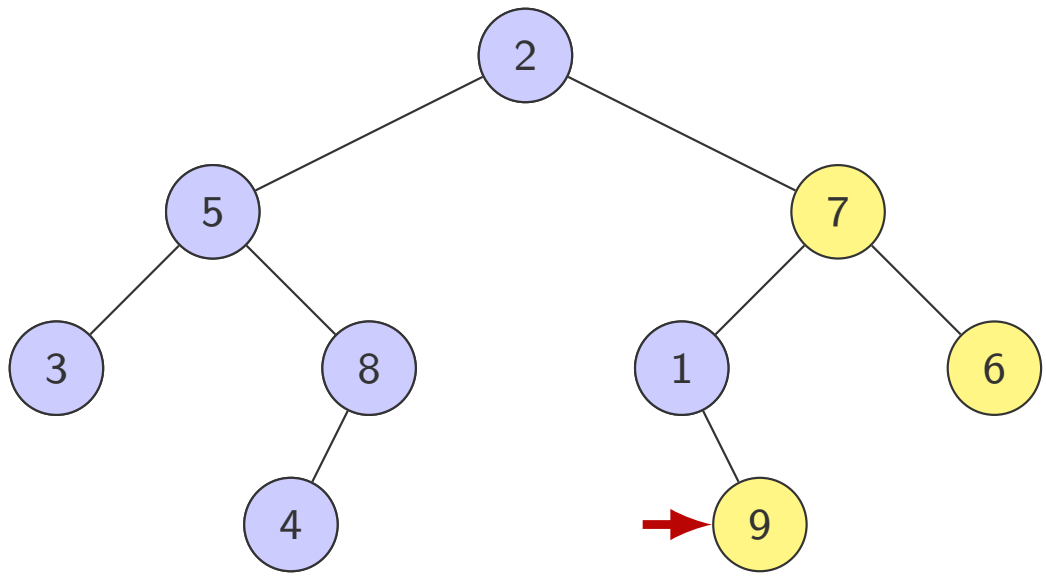


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

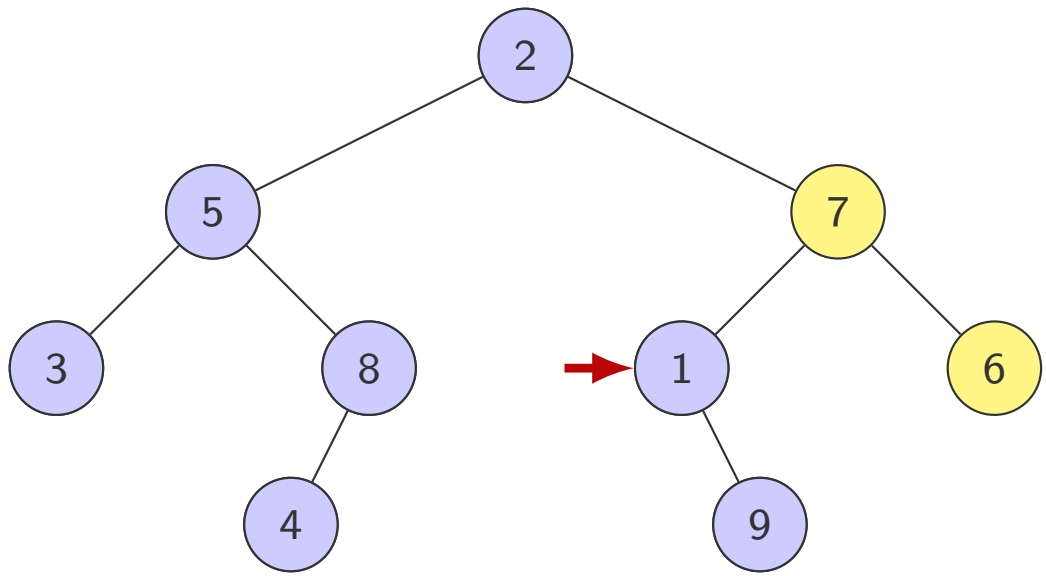


Ex: 3, 5, 4, 8, 2, 1,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

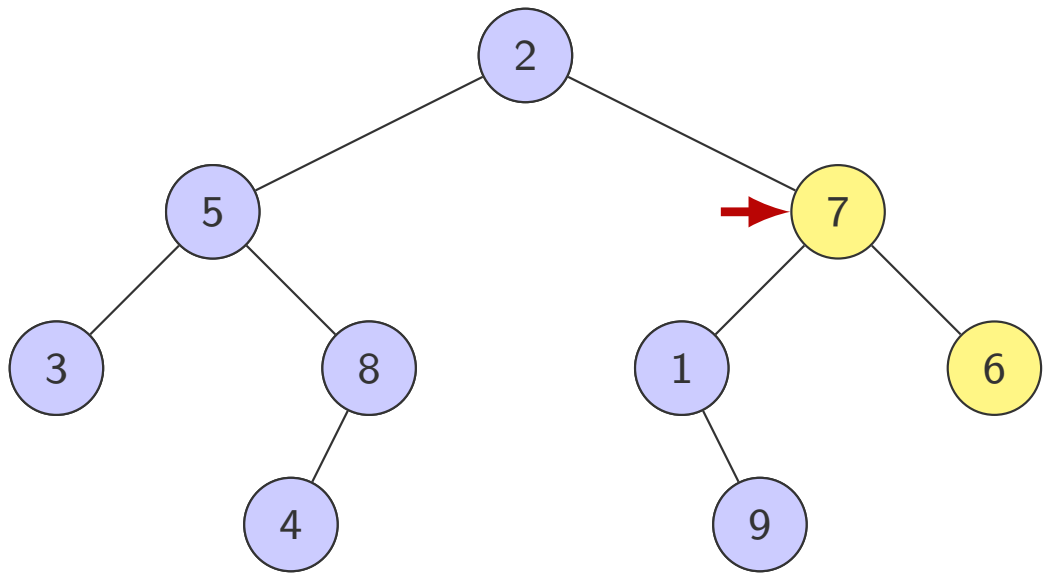


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

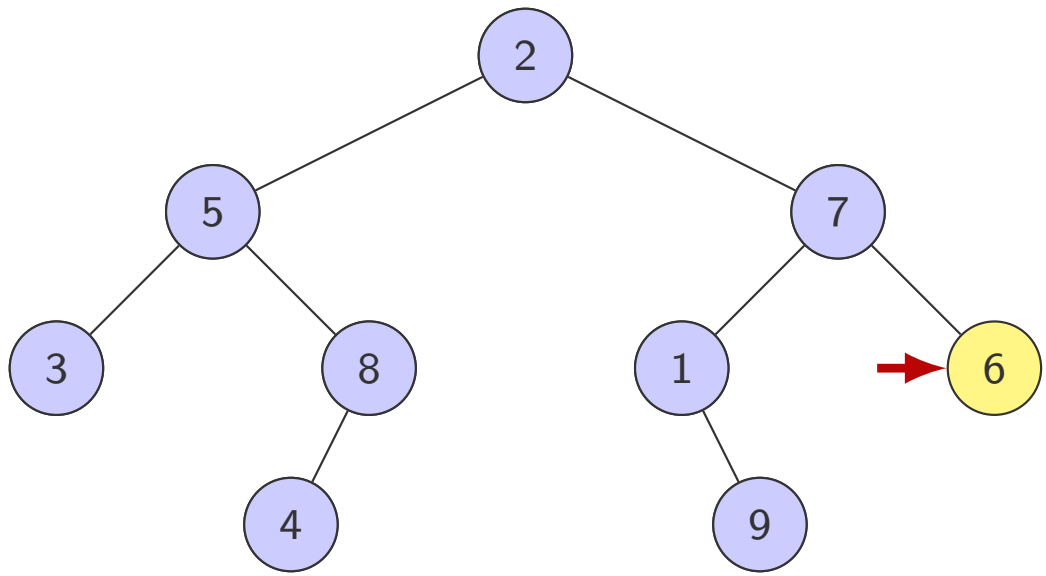


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

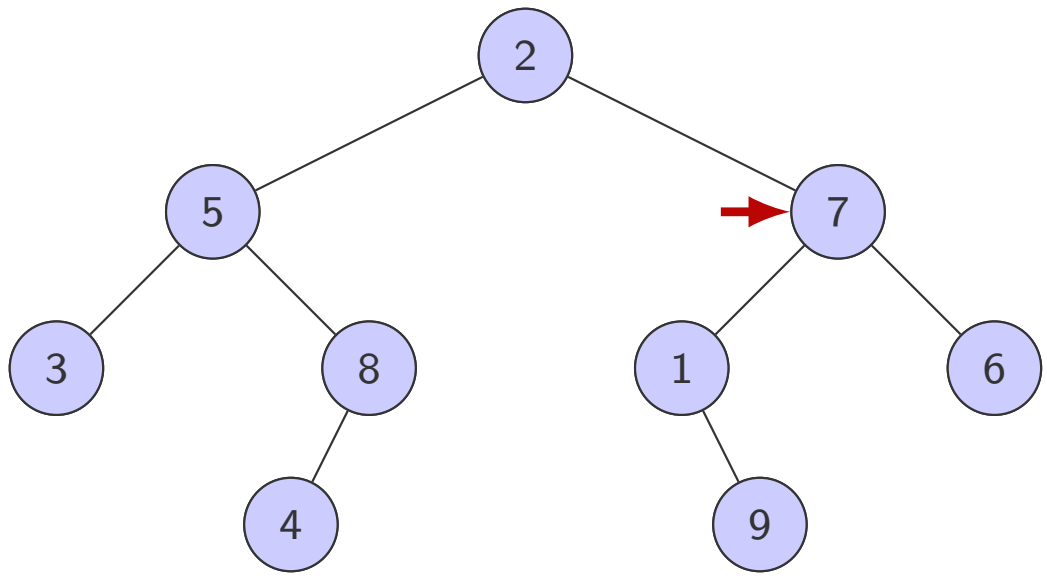


Ex: 3, 5, 4, 8, 2, 1, 9, 7,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

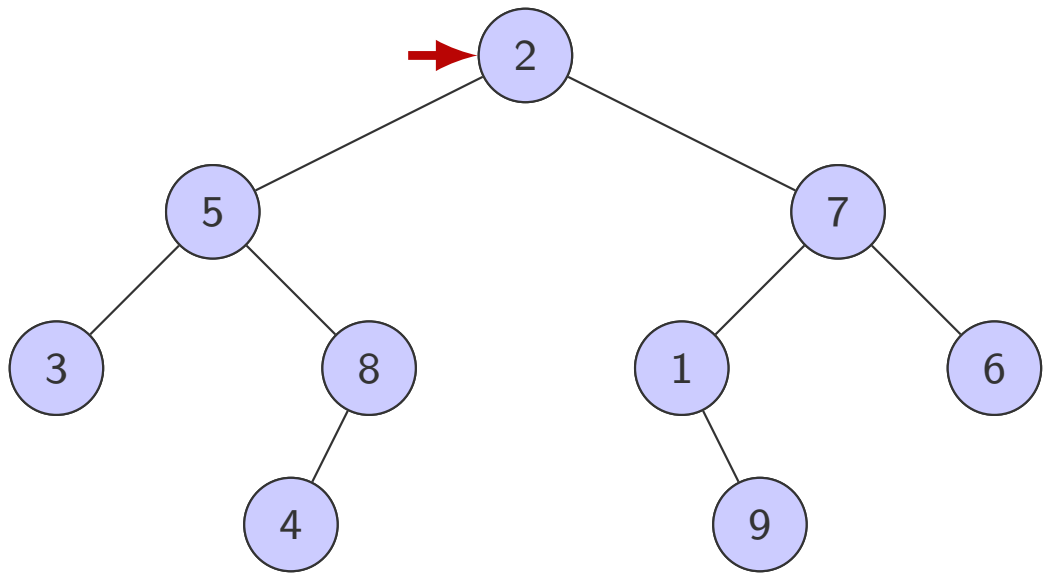


Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

- 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

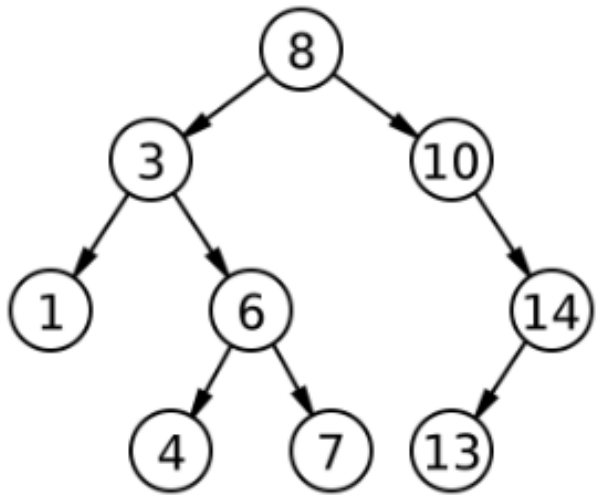
Ordem Simétrica (inorder)

```
1 // Percurso em-ordem
2 void arv_emordem(No *r) {
3     if(!arv_vazia(r)) {
4         arv_emordem(r->esq);
5         printf("%d ", r->chave);
6         arv_emordem(r->dir);
7     }
8 }
```

Serialização de árvores

Serialização de Árvores

- A serialização de uma árvore binária é um processo pelo qual percorremos a árvore em pré-ordem e adicionamos o valor de cada chave encontrada ao final de uma string que inicialmente começa vazia, sendo que, para cada filho nulo encontrado, seu valor é representado pelo caractere '#'.
  Exemplo:



A serialização da árvore acima consiste na string:

```
8 3 1 # # 6 4 # # 7 # # 10 # 14 13 # # #
```

Serializar uma árvore

```
1  // Cria uma string serial da arvore binaria
2  void arv_serializar(No *r, char *str) {
3      if(arv_vazia(r)) {
4          int n = strlen(str);
5          str[n] = ' ';
6          str[n+1] = '#';
7          str[n+2] = '\0';
8      } else {
9          char numero[25] = " ";
10         snprintf(numero + 1, sizeof(numero)-strlen(numero),
11                  "%d", r->chave);
12         strcat(str, numero);
13         arv_serializar(r->esq, str);
14         arv_serializar(r->dir, str);
15     }
16 }
```

Deserializar uma árvore

```
1  // Cria uma arvore binaria a partir duma string serial valida
2  No* arv_deserializar(char *str, int *idx) {
3      if(str[*idx] == '#') {
4          *idx = *idx + 2;
5          return NULL;
6      } else {
7          char token[50];
8          int i = 0;
9          while(str[*idx] != ' ' && str[*idx] != '\0') {
10             token[i++] = str[(*idx)++];
11         }
12         *idx = *idx + 1;
13         token[i] = '\0';
14
15         No *raiz = (No*) malloc(sizeof(No));
16         raiz->chave = (int) atoi(token);
17         raiz->esq = arv_deserializar(str, idx);
18         raiz->dir = arv_deserializar(str, idx);
19         return raiz;
20     }
21 }
```

Percurso em largura

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)

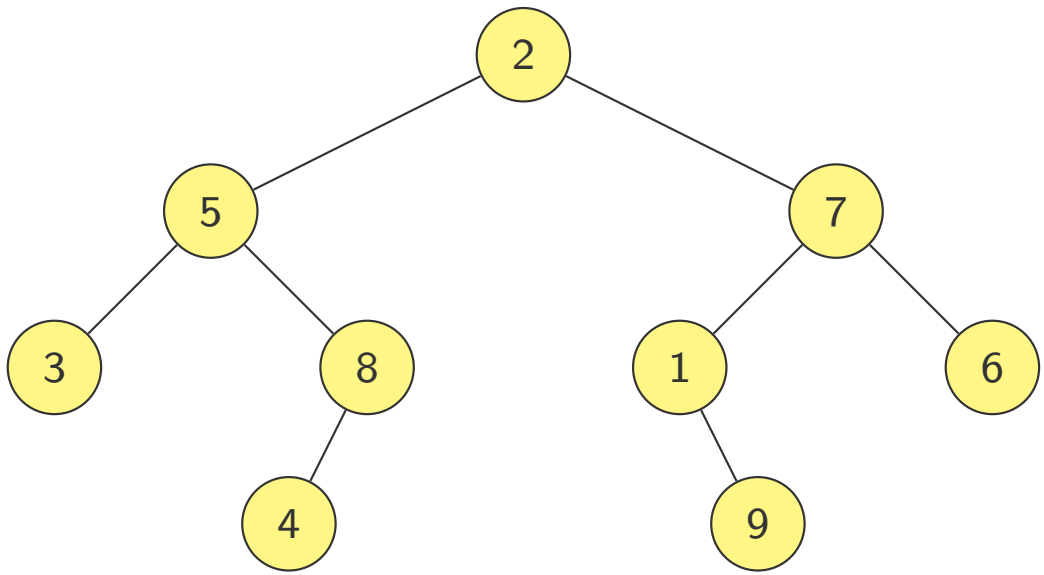
O percurso em largura

- 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

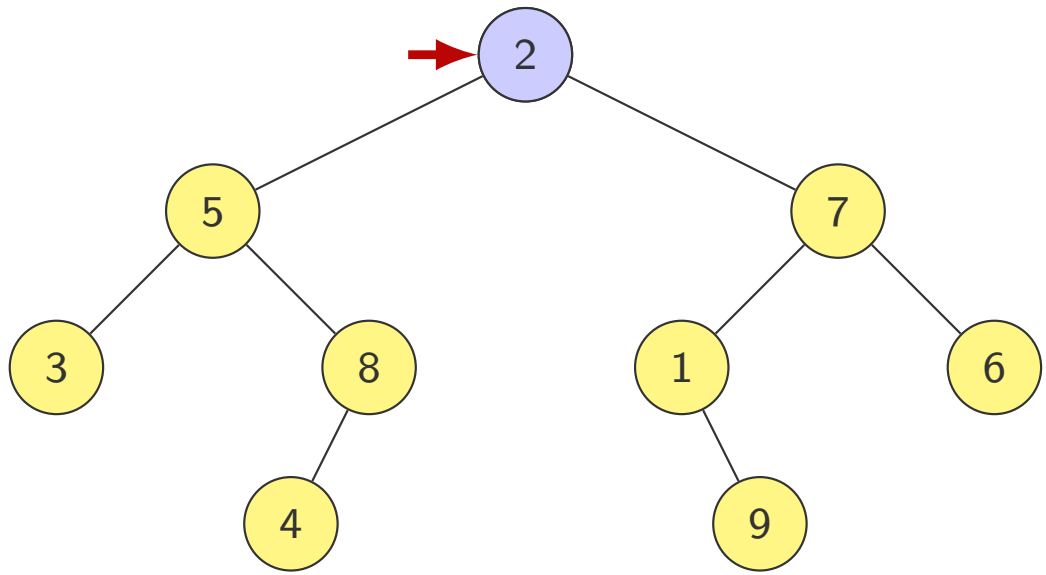


Ex:

Percorrendo os nós (em largura)

O percurso em largura

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

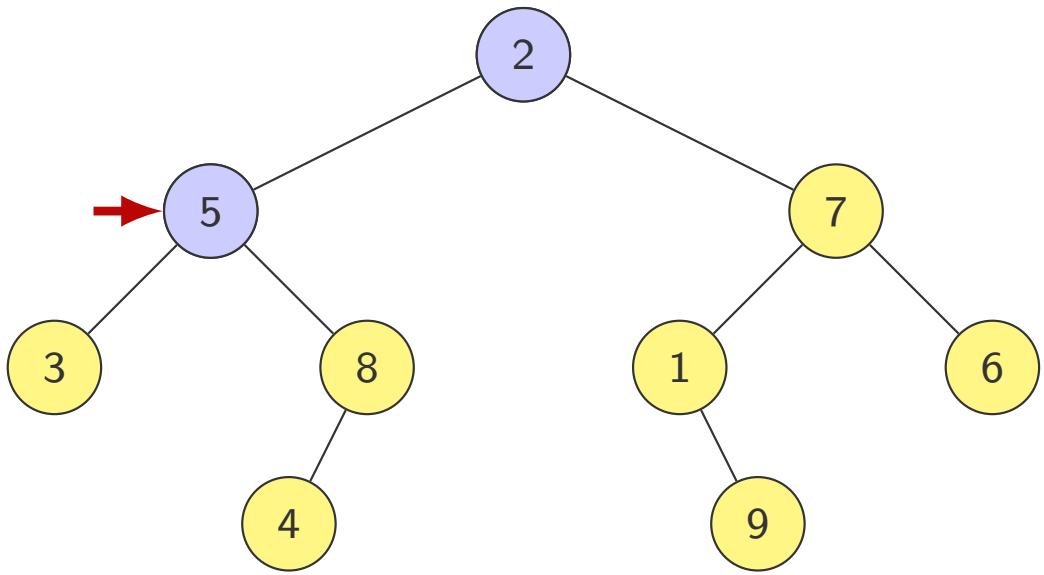


Ex: 2,

Percorrendo os nós (em largura)

O percurso em largura

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

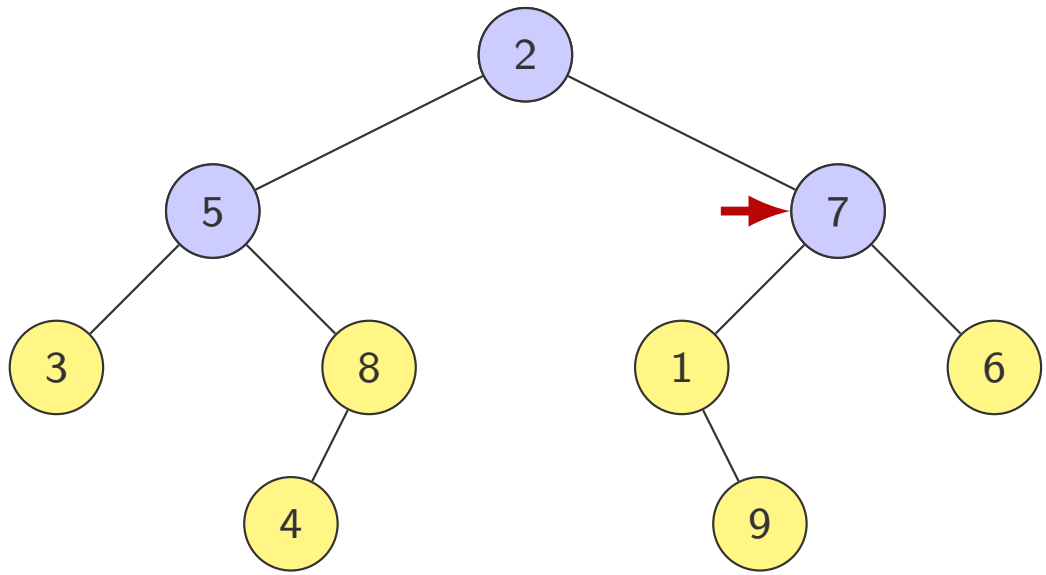


Ex: 2, 5,

Percorrendo os nós (em largura)

O percurso em largura

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

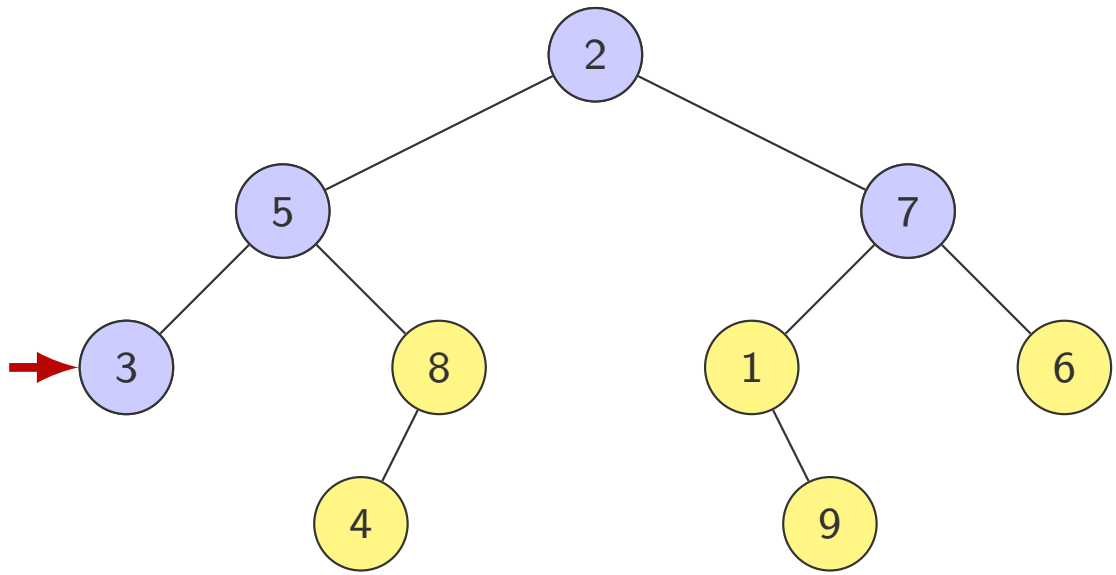


Ex: 2, 5, 7,

Percorrendo os nós (em largura)

O percurso em largura

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

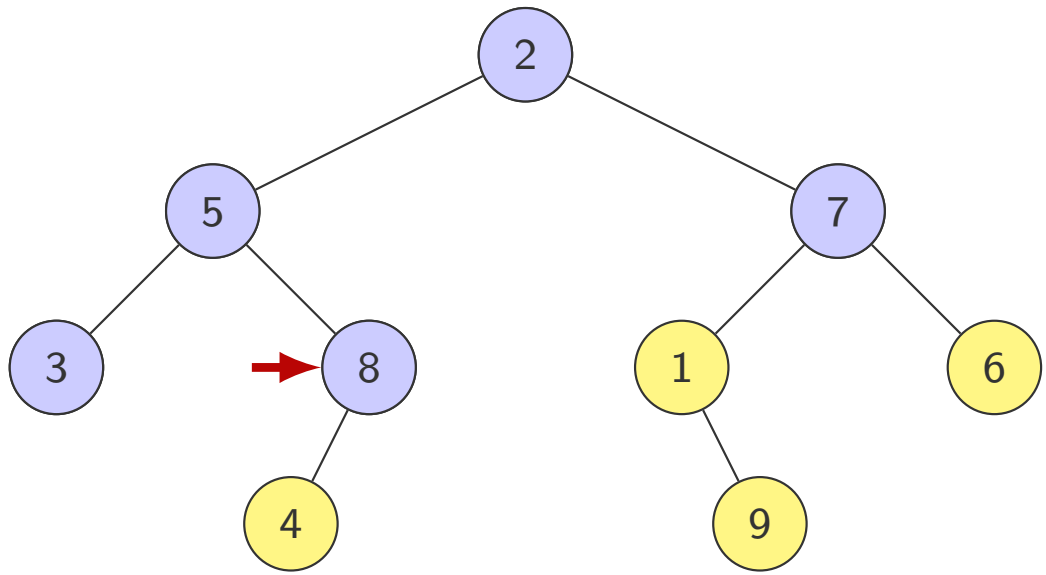


Ex: 2, 5, 7, 3,

Percorrendo os nós (em largura)

O percurso em largura

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

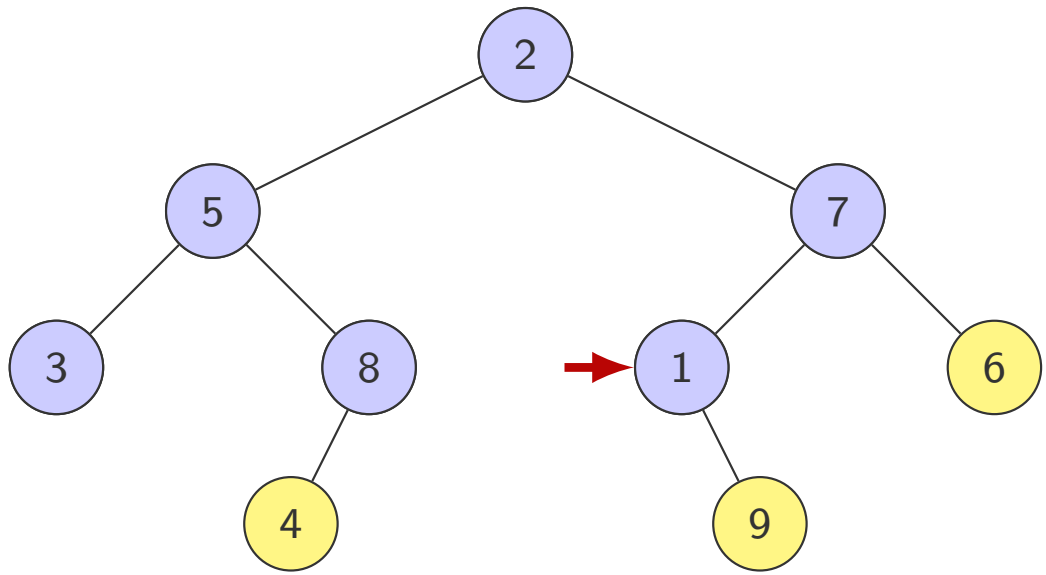


Ex: 2, 5, 7, 3, 8,

Percorrendo os nós (em largura)

O percurso em largura

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

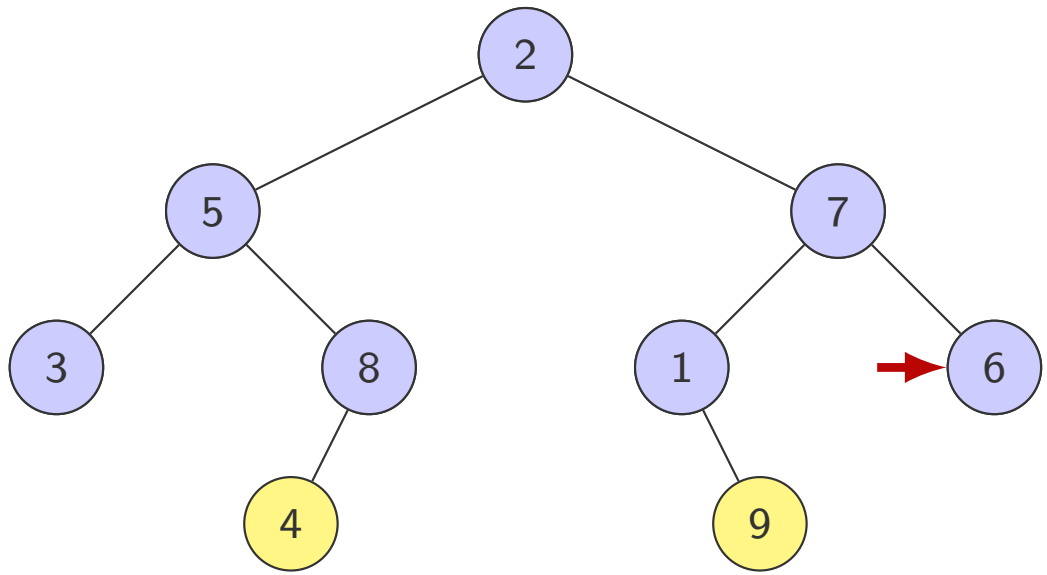


Ex: 2, 5, 7, 3, 8, 1,

Percorrendo os nós (em largura)

O percurso em largura

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

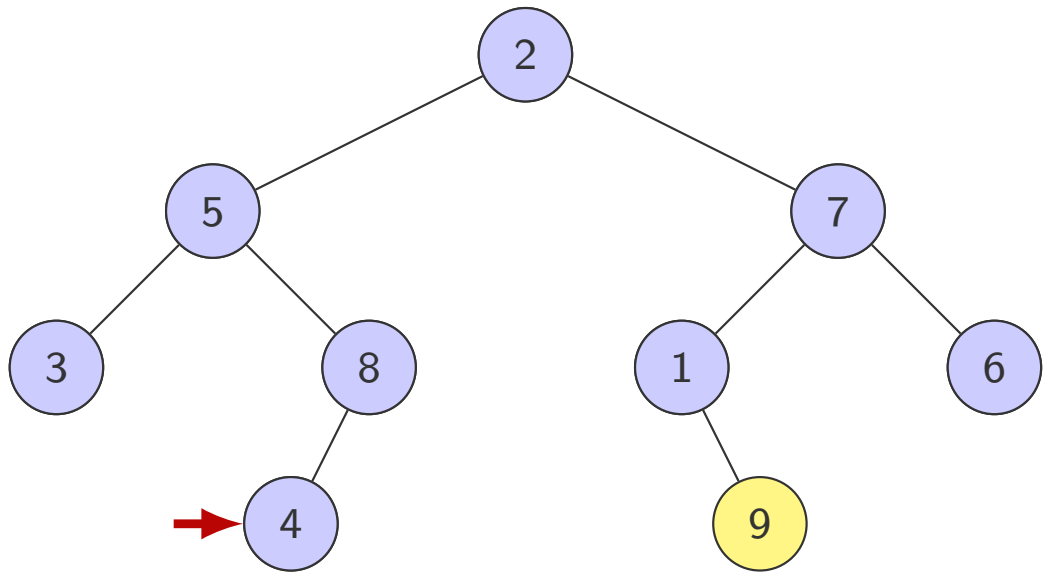


Ex: 2, 5, 7, 3, 8, 1, 6,

Percorrendo os nós (em largura)

O percurso em largura

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

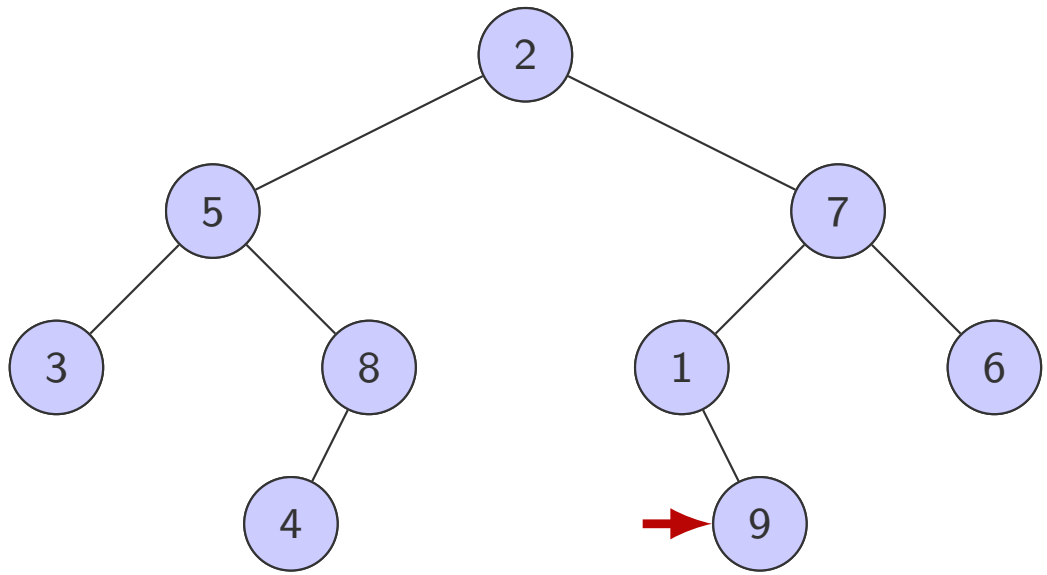


Ex: 2, 5, 7, 3, 8, 1, 6, 4,

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 a busca em largura?

Implementação do percurso em largura

Como implementar a busca em largura?

- Usamos uma fila

Implementação do percurso em largura

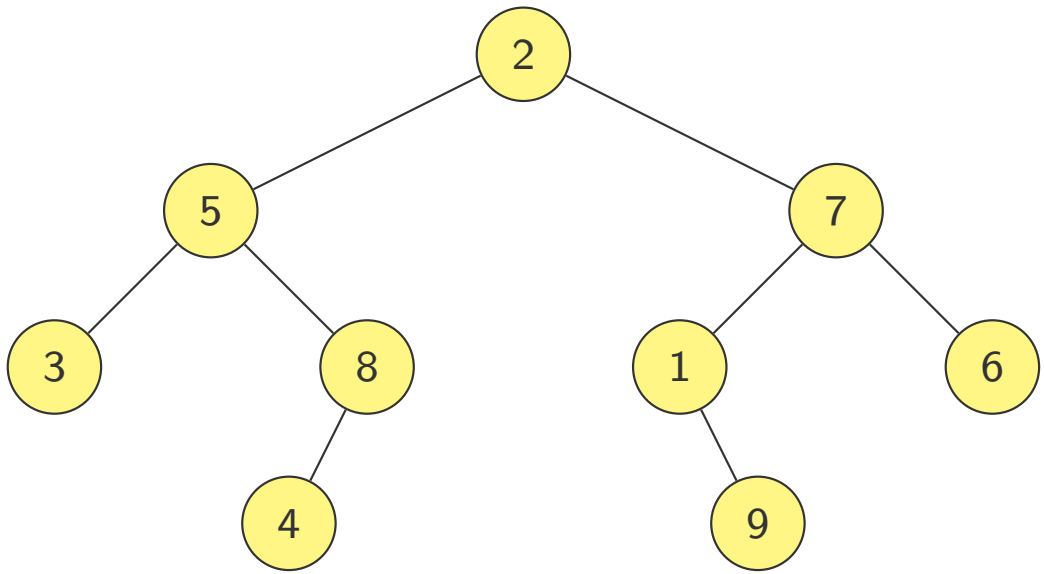
Como implementar a busca em largura?

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

Implementação do percurso em largura

Como implementar a busca em largura?

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

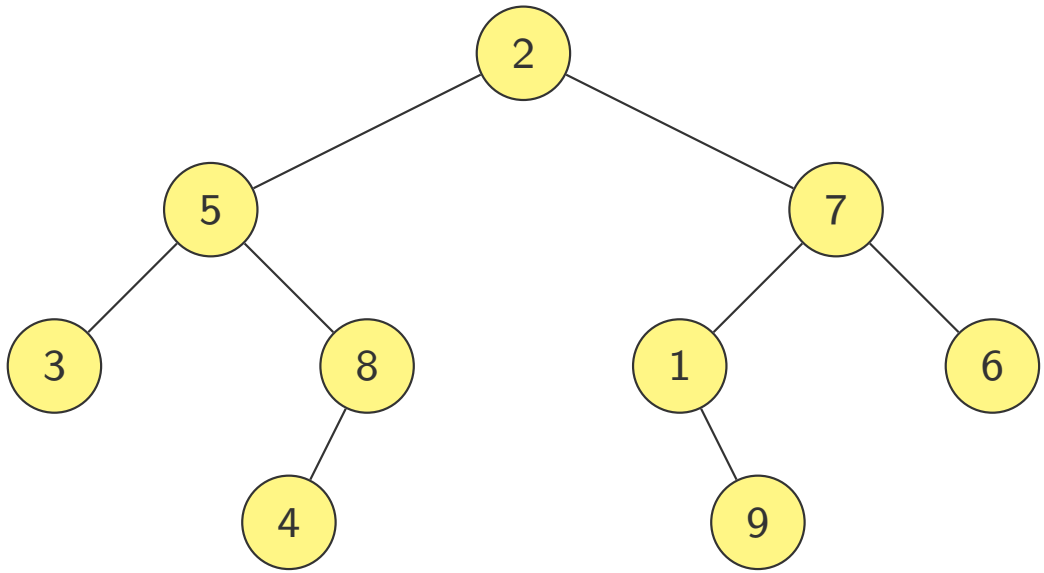


Fila

Implementação do percurso em largura

Como implementar a busca em largura?

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

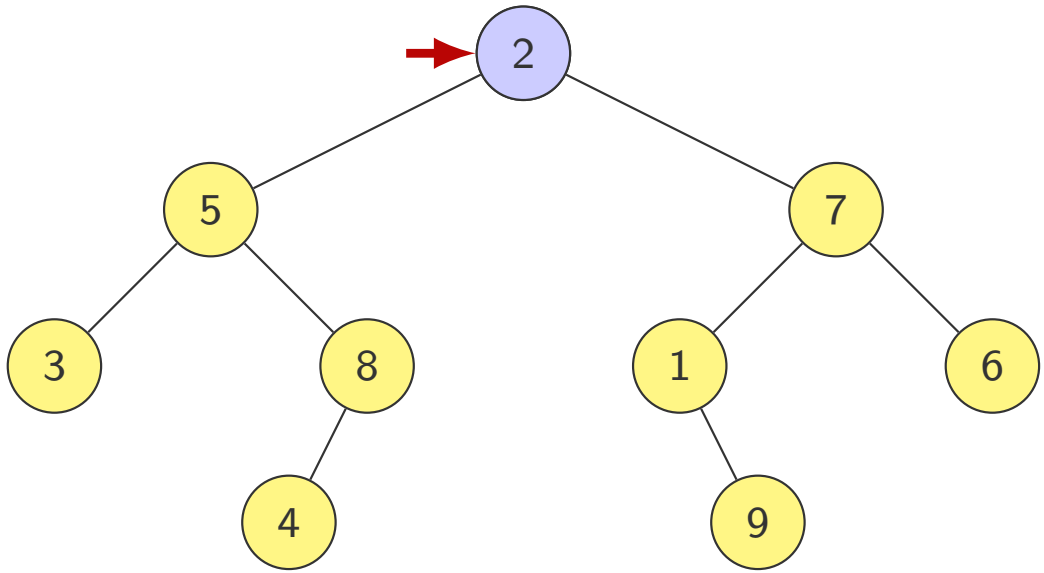


Fila 

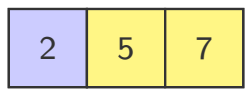
Implementação do percurso em largura

Como implementar a busca em largura?

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



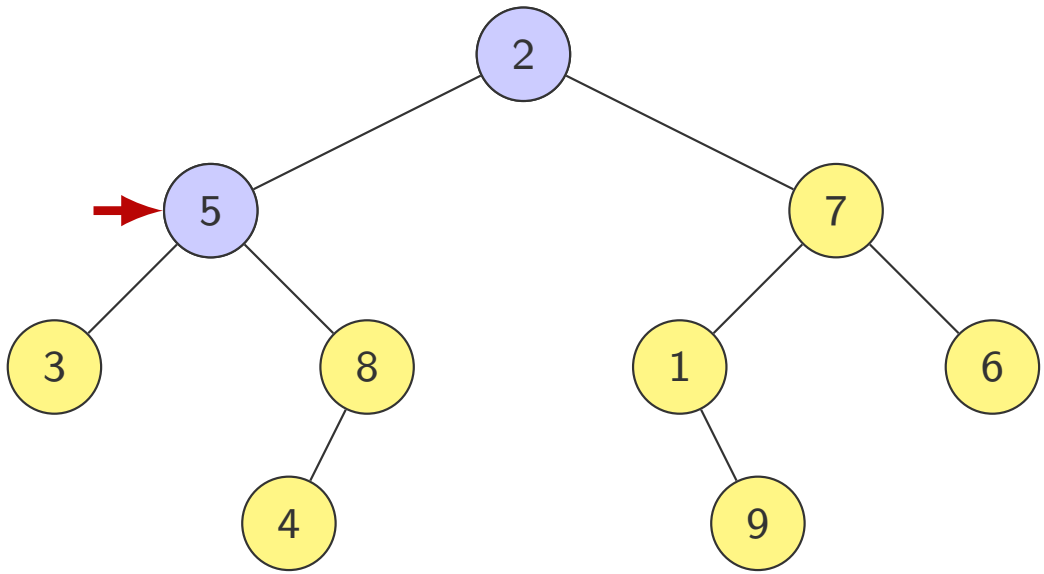
Fila



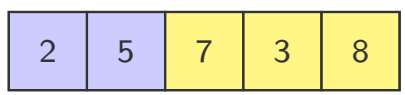
Implementação do percurso em largura

Como implementar a busca em largura?

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



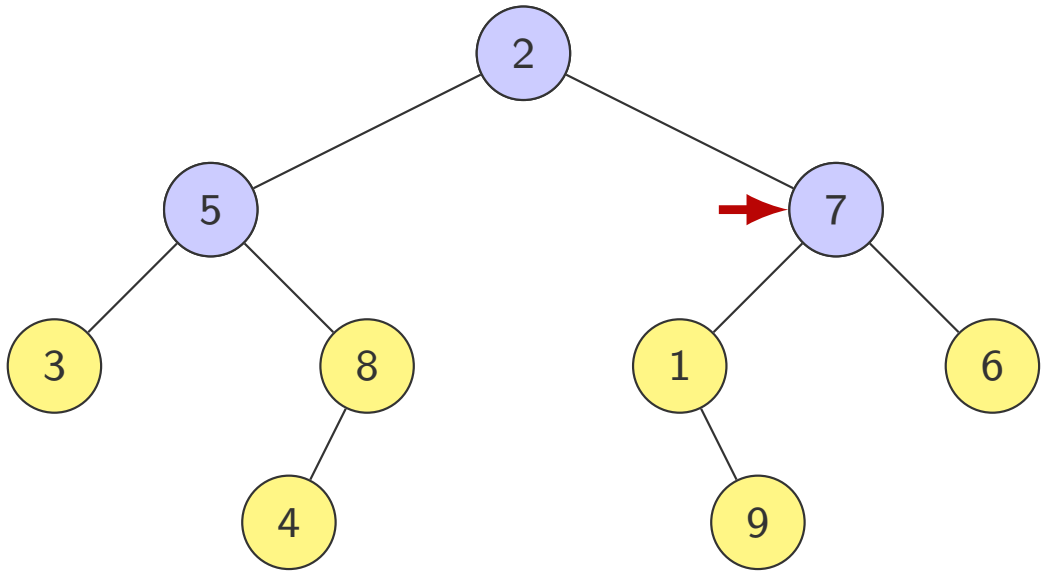
Fila



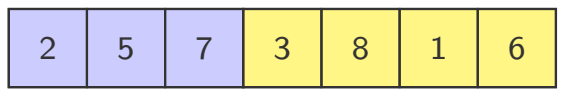
Implementação do percurso em largura

Como implementar a busca em largura?

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



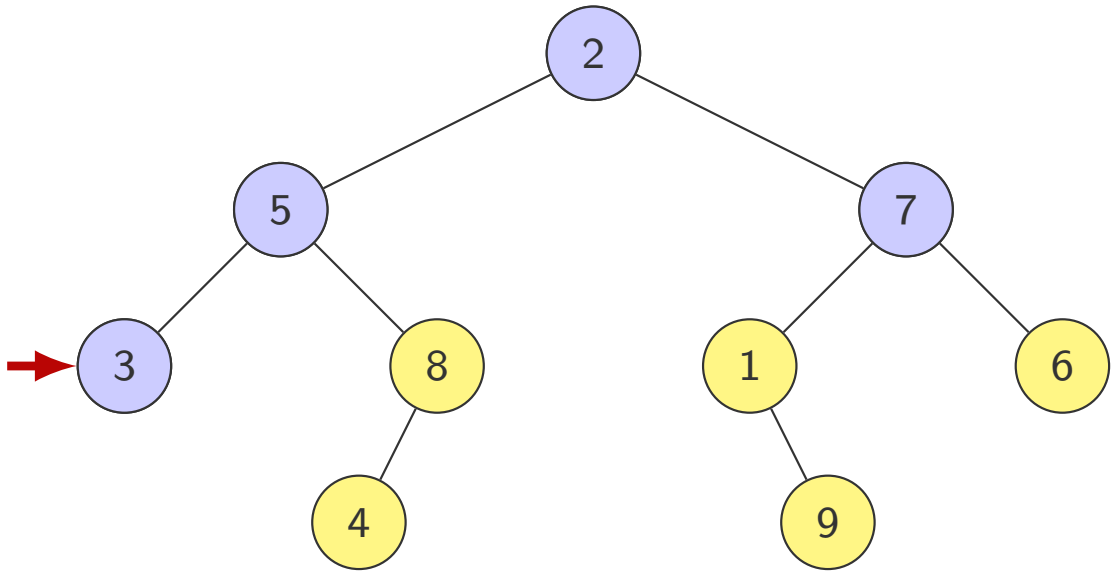
Fila



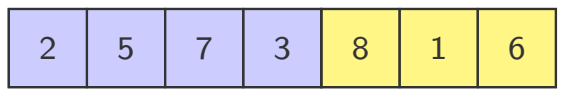
Implementação do percurso em largura

Como implementar a busca em largura?

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



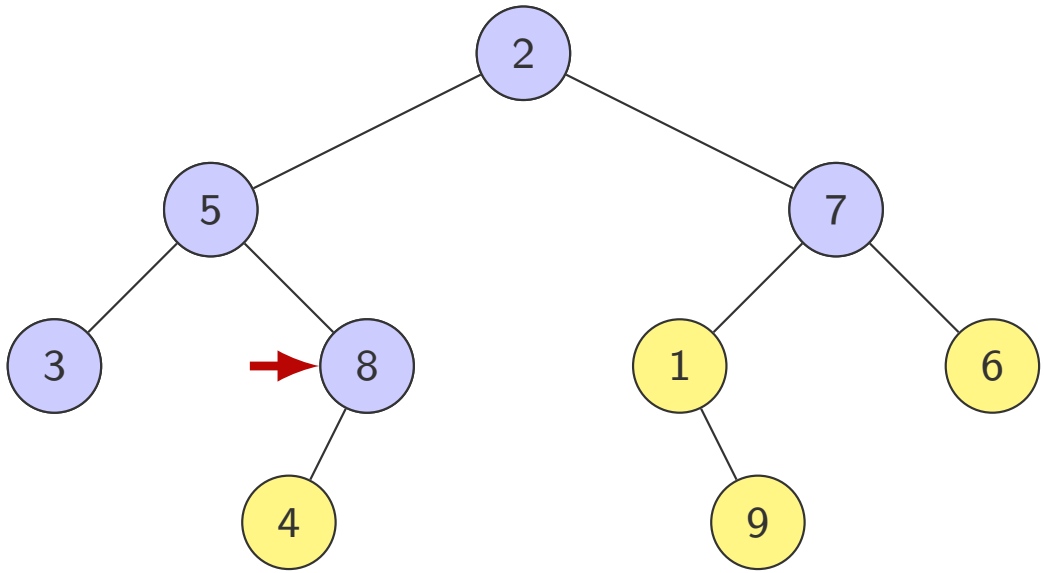
Fila



Implementação do percurso em largura

Como implementar a busca em largura?

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

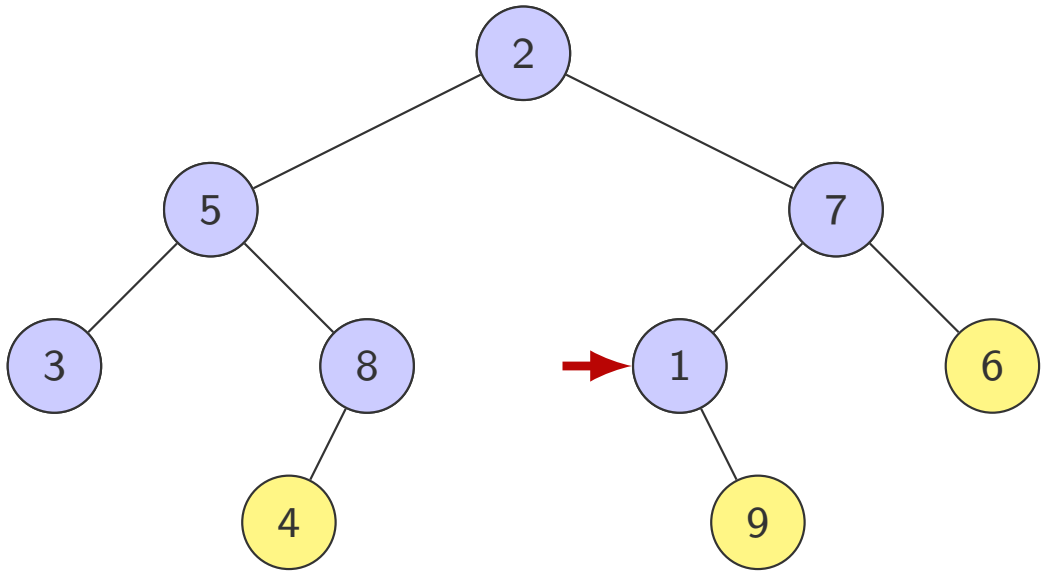


Fila

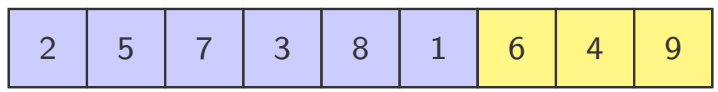
Implementação do percurso em largura

Como implementar a busca em largura?

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



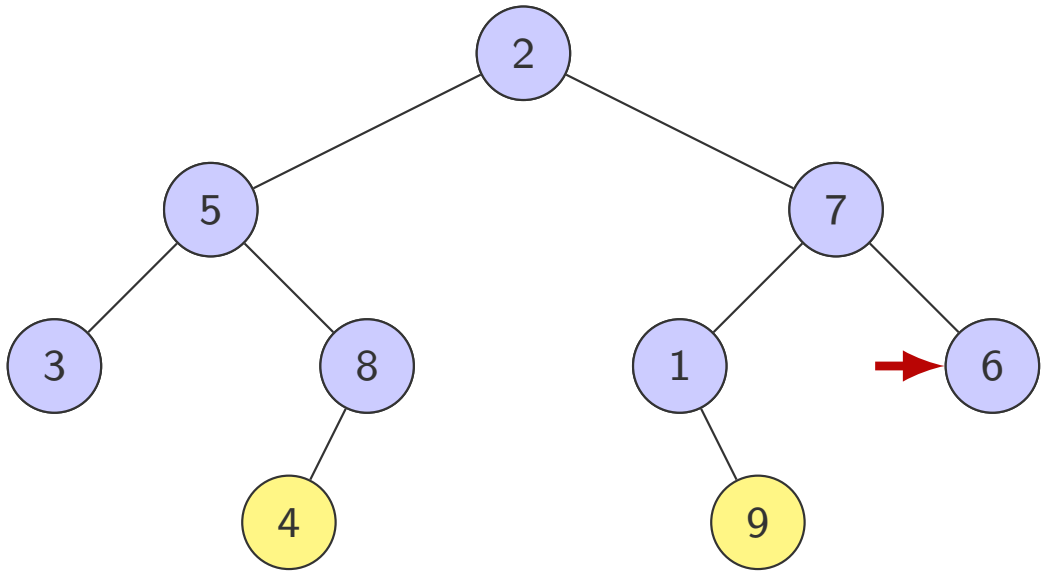
Fila



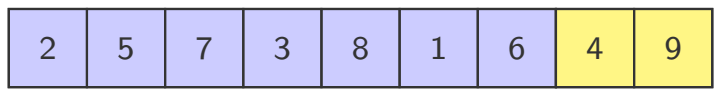
Implementação do percurso em largura

Como implementar a busca em largura?

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



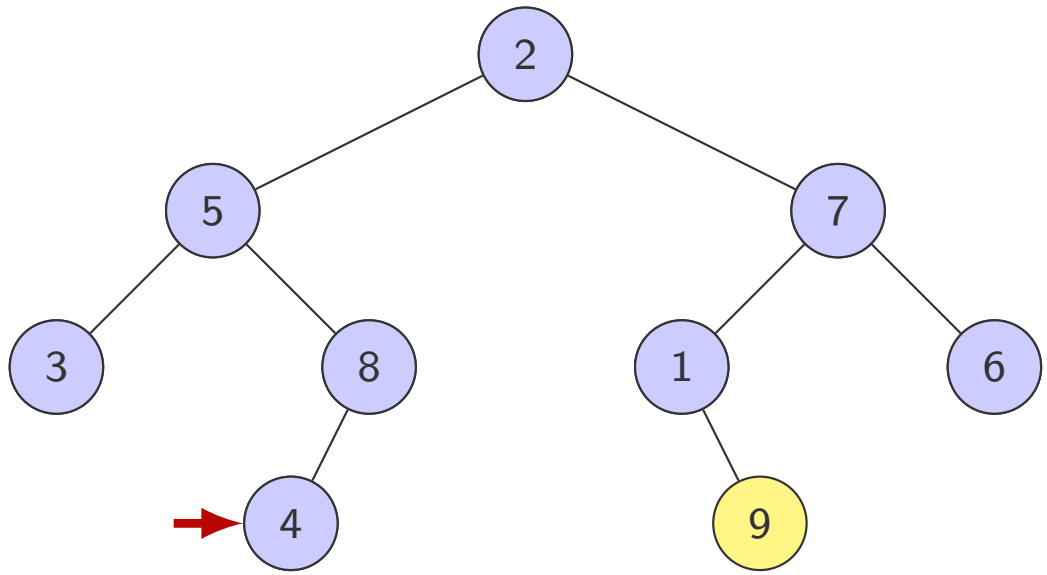
Fila



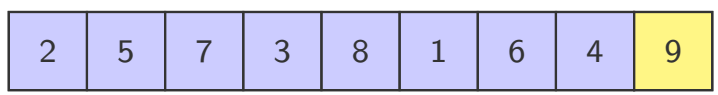
Implementação do percurso em largura

Como implementar a busca em largura?

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



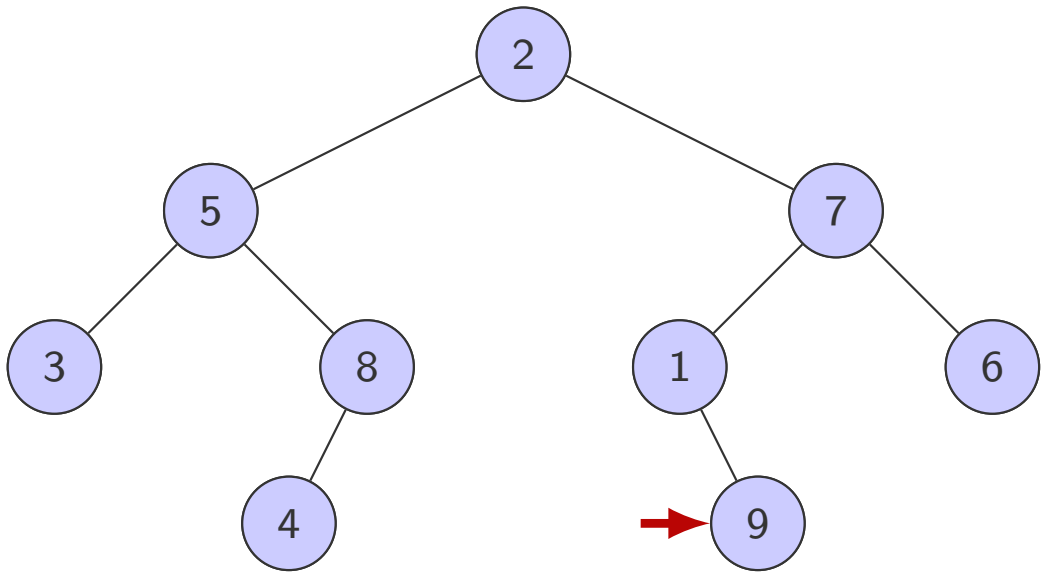
Fila



Implementação do percurso em largura

Como implementar a busca 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 — pseudocódigo

Algorithm PercursoEmLargura(No *root)

Require: root (ponteiro para a raiz da árvore)

```
1:  Cria uma fila vazia Q de ponteiros para nós
2:  queue_push(Q, root)
3:  while fila Q não for vazia do
4:      node = queue_front(Q)
5:      queue_pop(Q)
6:      if node ≠ NULL then
7:          visit(node)
8:          queue_push(Q, node→left)
9:          queue_push(Q, node→right)
10:     end if
11: end while
```

Exercícios

Exercícios

- Escreva uma função que calcula o número de nós de uma árvore. A função deve obedecer o seguinte protótipo:
  `int bt_size(Node* node);`

Exercícios

- Escreva uma função que calcula o número de nós de uma árvore. A função deve obedecer o seguinte protótipo:
  `int bt_size(Node* node);`

- Escreva uma função que calcula a altura de uma árvore. A função deve obedecer o seguinte protótipo:
  `int bt_height(Node* node);`

Exercícios

- Escreva uma função que calcula o número de nós de uma árvore. A função deve obedecer o seguinte protótipo:
  `int bt_size(Node* node);`

- Escreva uma função que calcula a altura de uma árvore. A função deve obedecer o seguinte protótipo:
  `int bt_height(Node* node);`

- Adicione o campo `height` ao struct `Node`. O campo `height` deve ser do tipo `int`. Implemente a função `bt_height(Node* node)` de modo que ela preencha o campo `height` de cada nó com a altura do nó.

Exercícios

- Um caminho que vai da raiz de uma árvore até um nó qualquer pode ser representado por uma sequência de 0s e 1s, do seguinte modo:
  - toda vez que o caminho "desce para a esquerda" temos um 0; toda vez que "desce para a direita" temos um 1.
  - Diremos que essa sequência de 0s e 1s é o código do nó.

- Suponha agora que todo nó de nossa árvore tem um campo adicional `code`, do tipo `string`, capaz de armazenar uma cadeia de caracteres de tamanho variável. Escreva uma função que preencha o campo `code` de cada nó com o código do nó.

FIM