Árvores Binárias

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará

2º semestre/2023

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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.

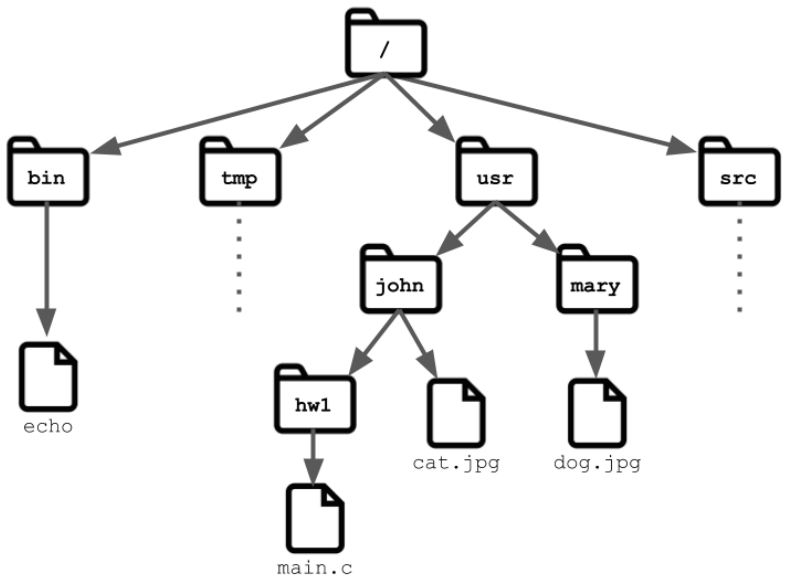


Figura: Hierarquia do sistema de arquivos de um PC Linux

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.

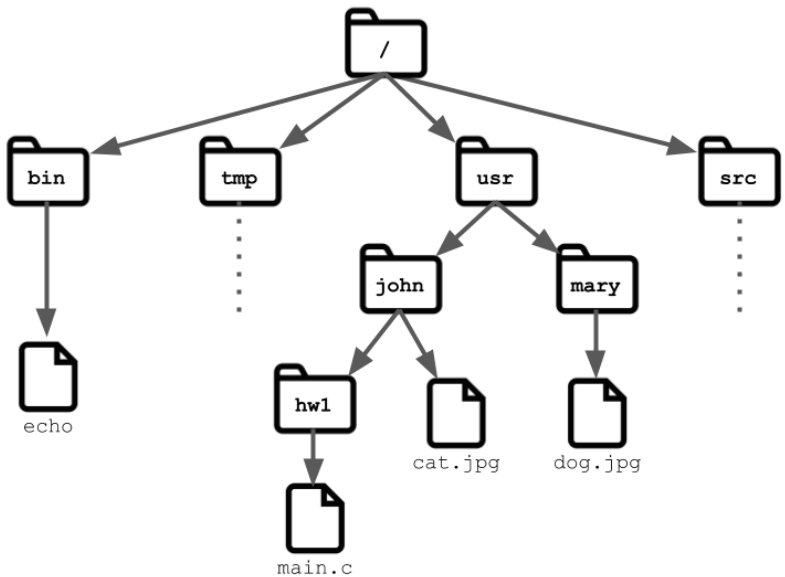


Figura: Hierarquia do sistema de arquivos de um PC Linux

- 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.

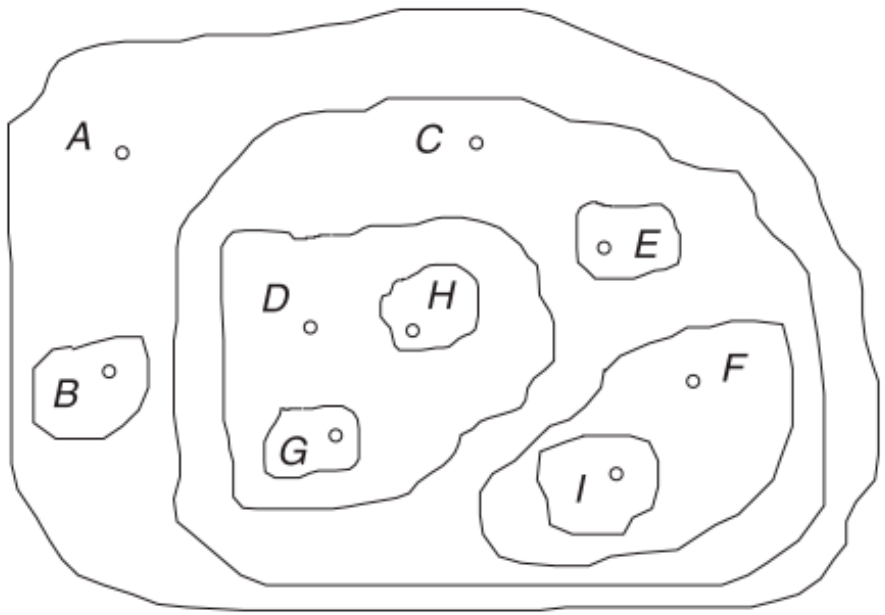
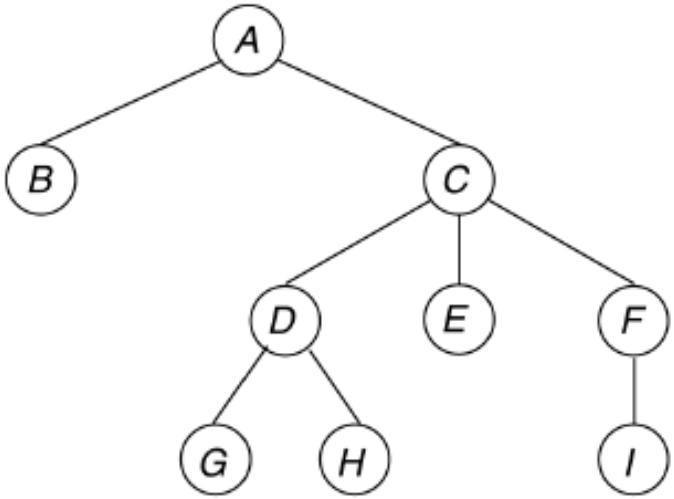


Diagrama de inclusão

Árvore — Outras Representações



Representação hierárquica

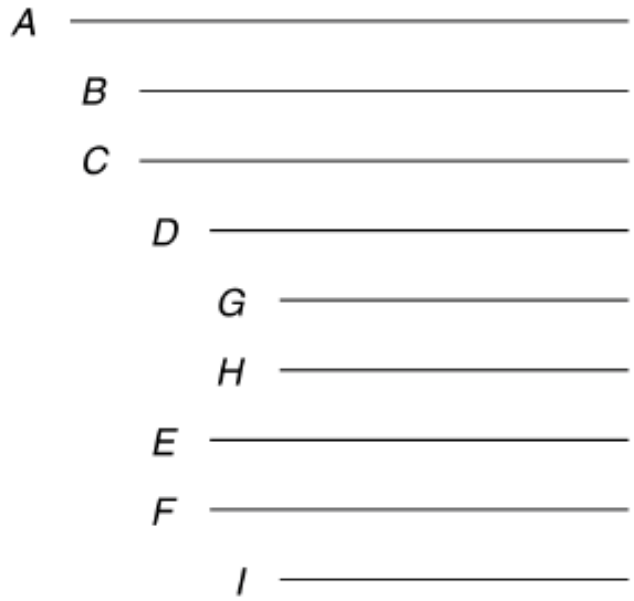
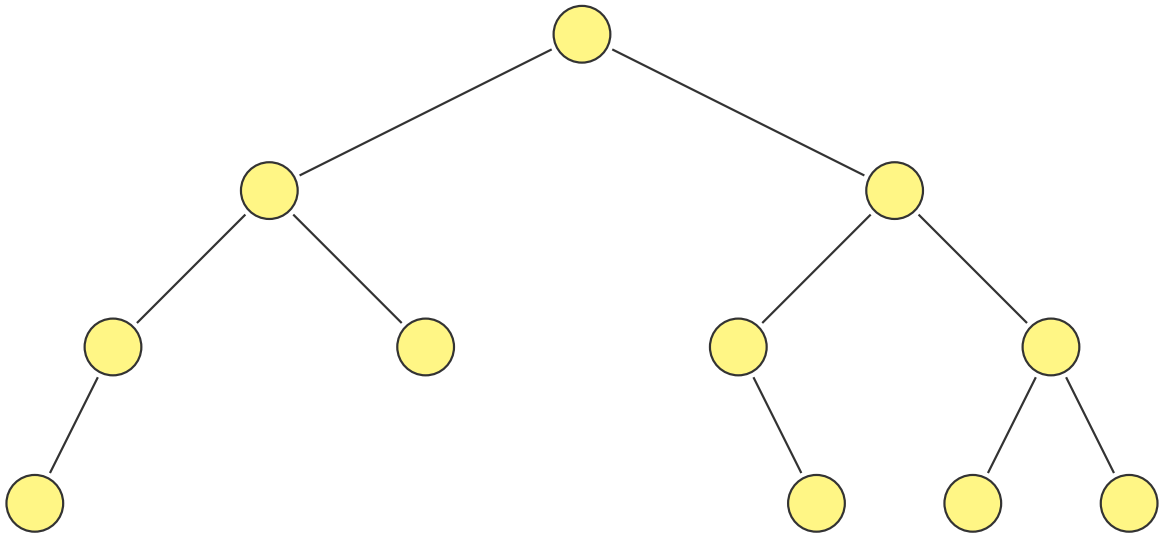


Diagrama de barras

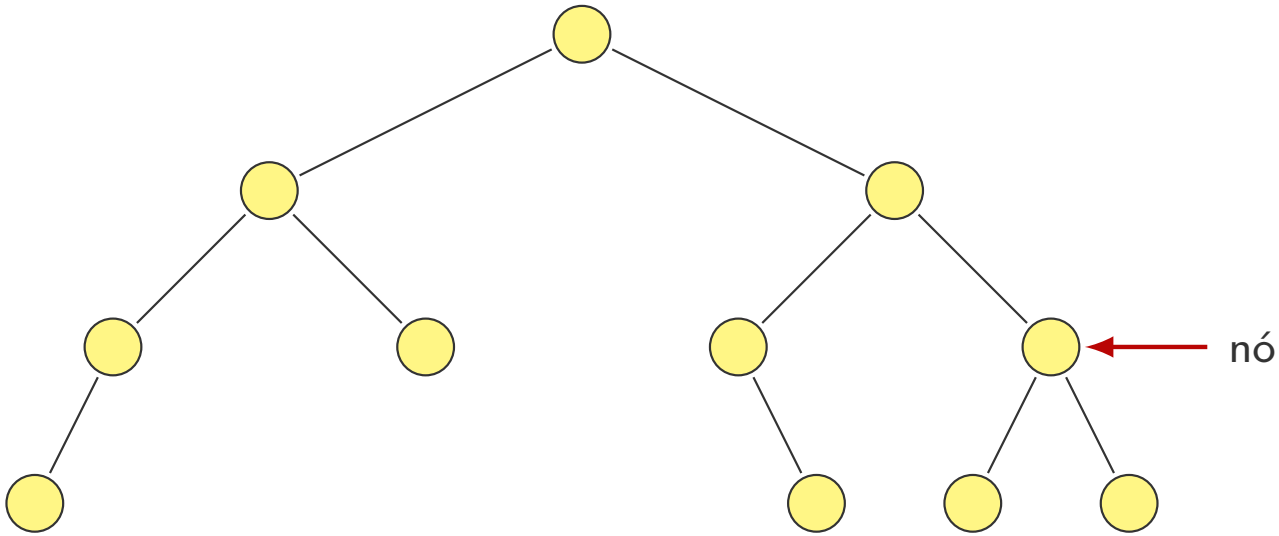
(A (B) (C (D (G) (H)) (E) (F (I))))

Representação por parênteses aninhados

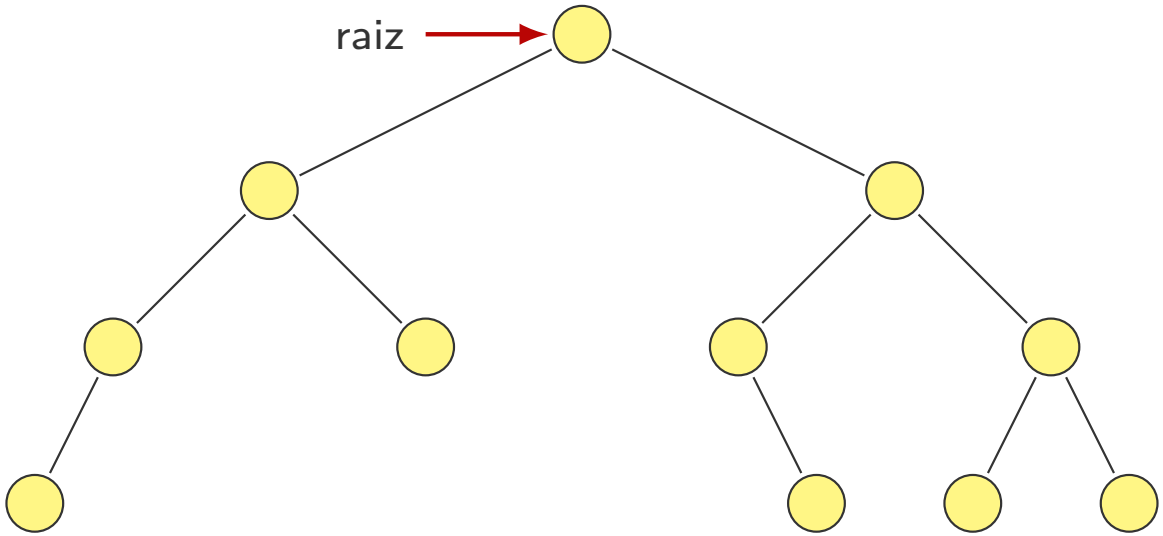
Definições Adicionais



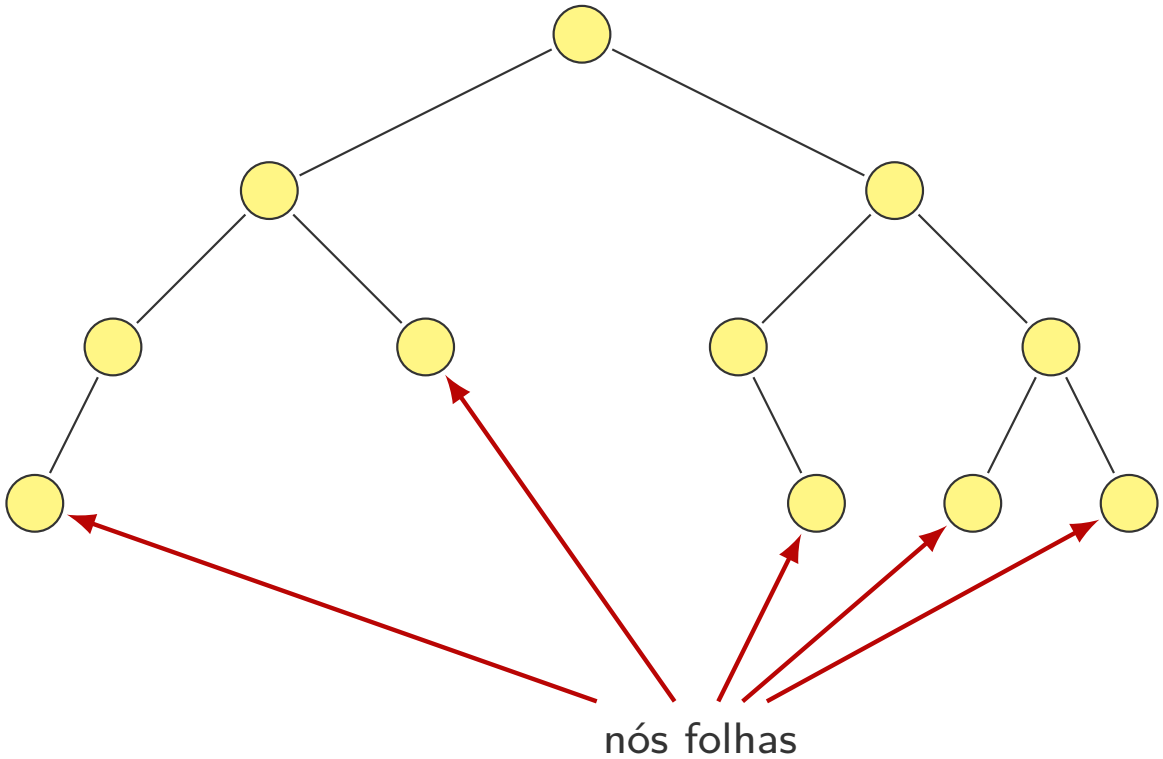
Definições Adicionais



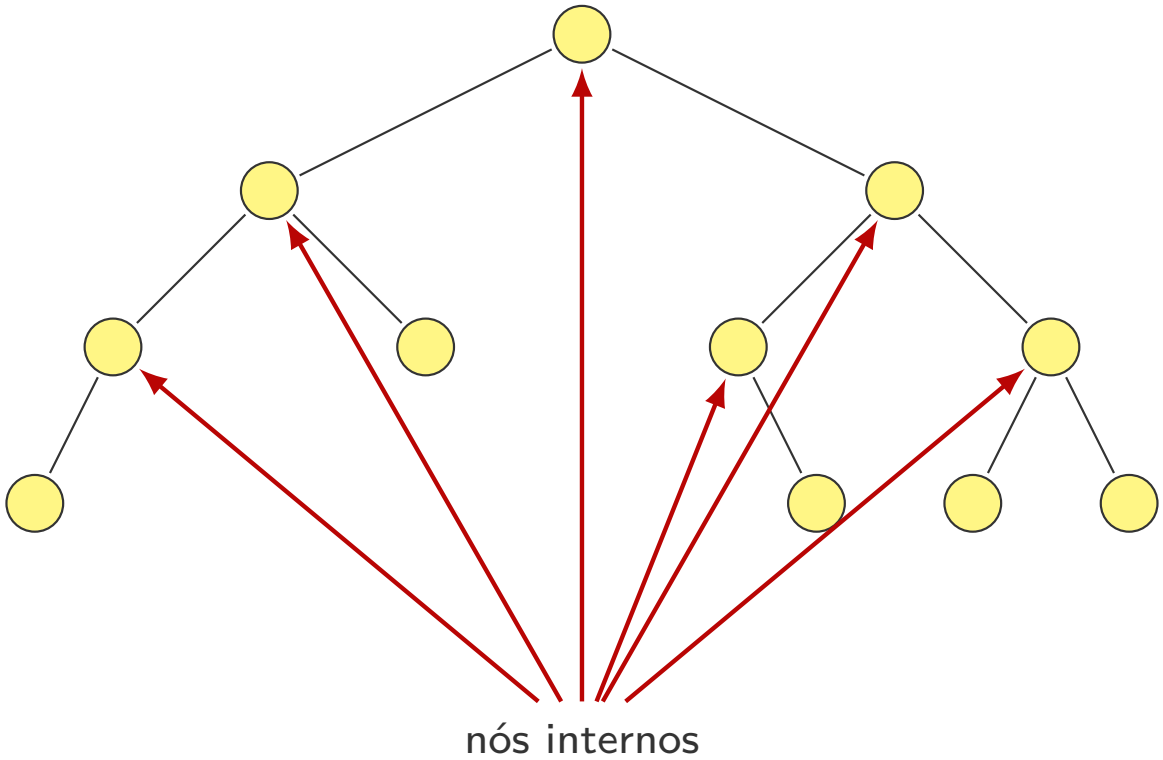
Definições Adicionais



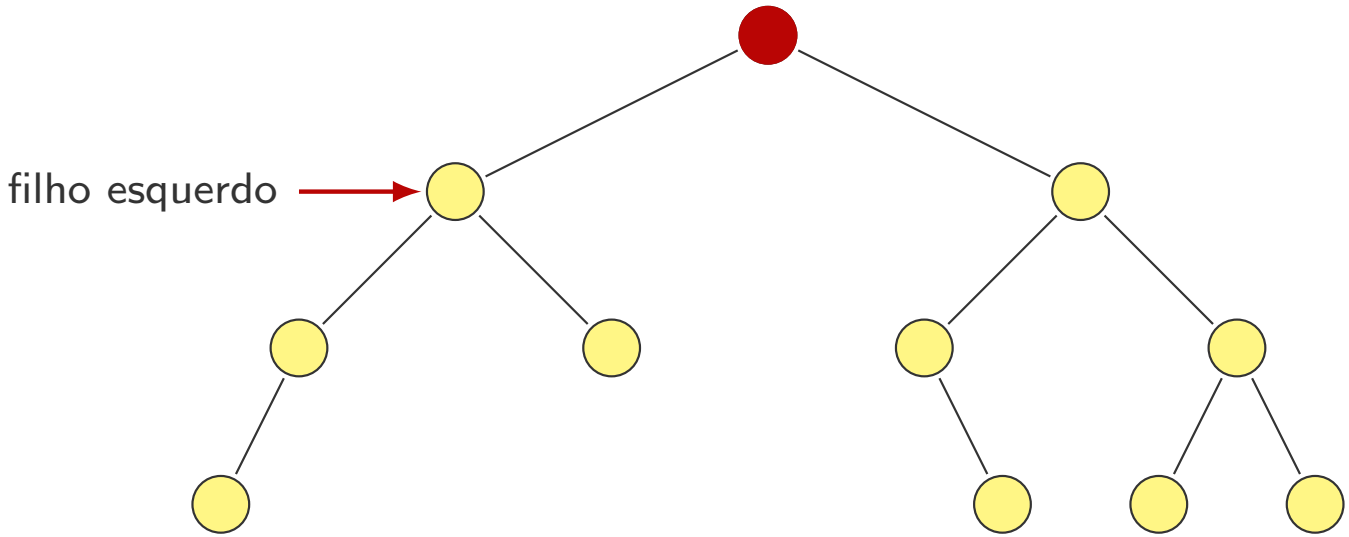
Definições Adicionais



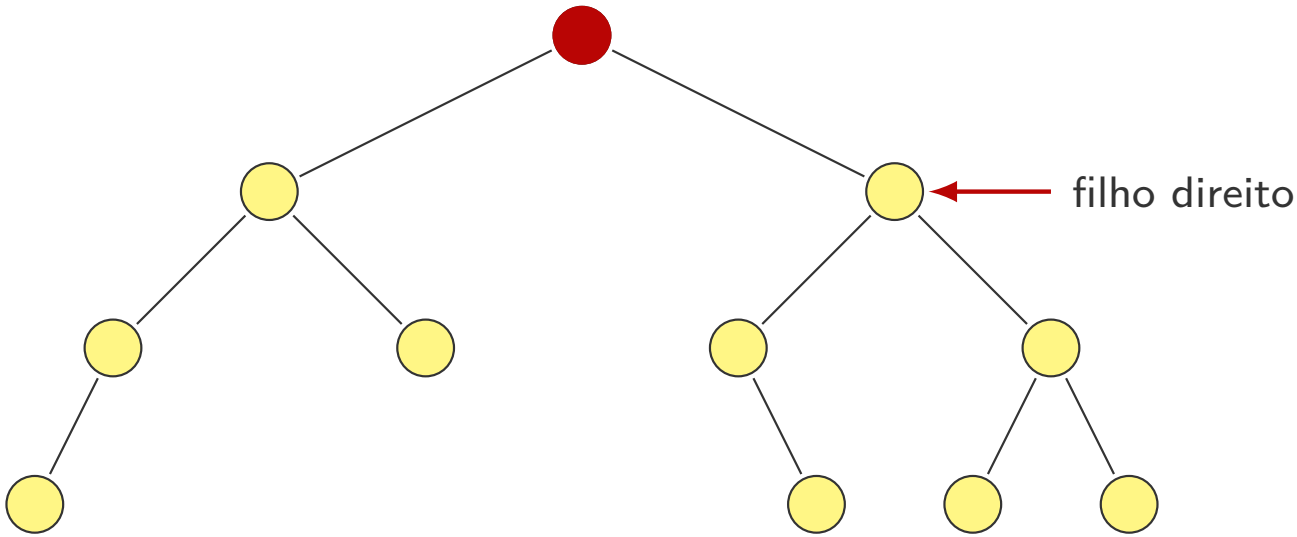
Definições Adicionais



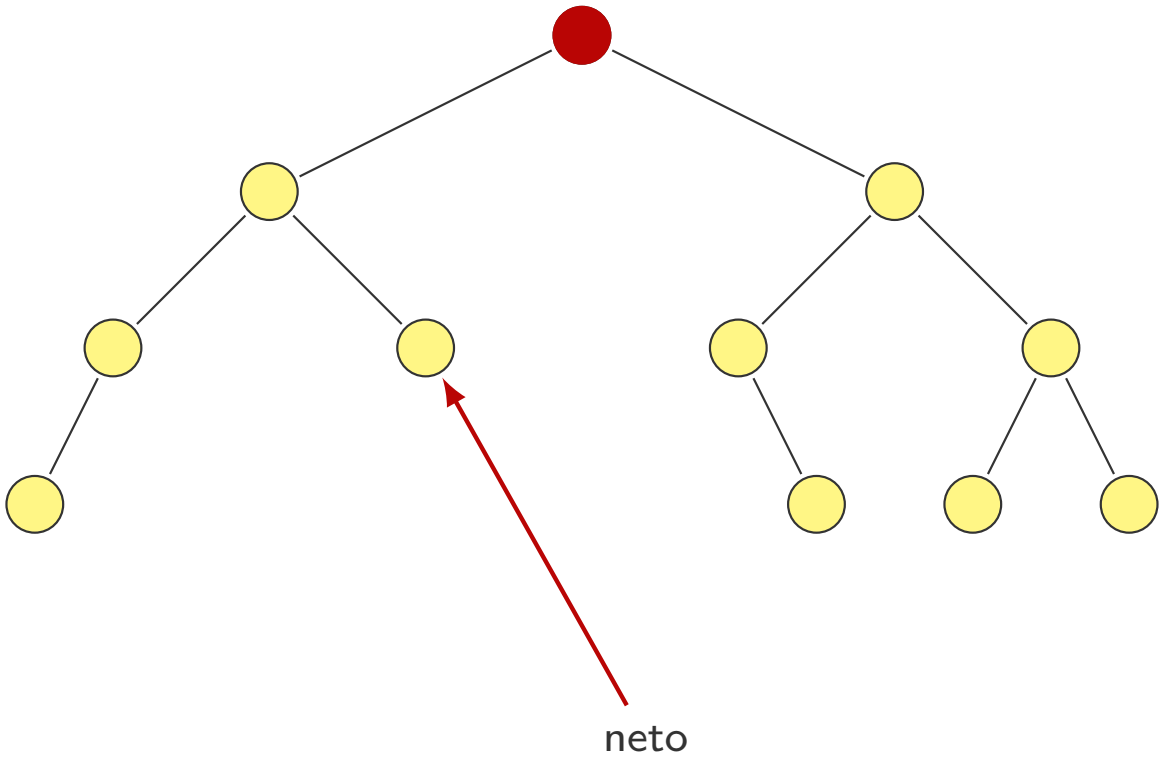
Definições Adicionais



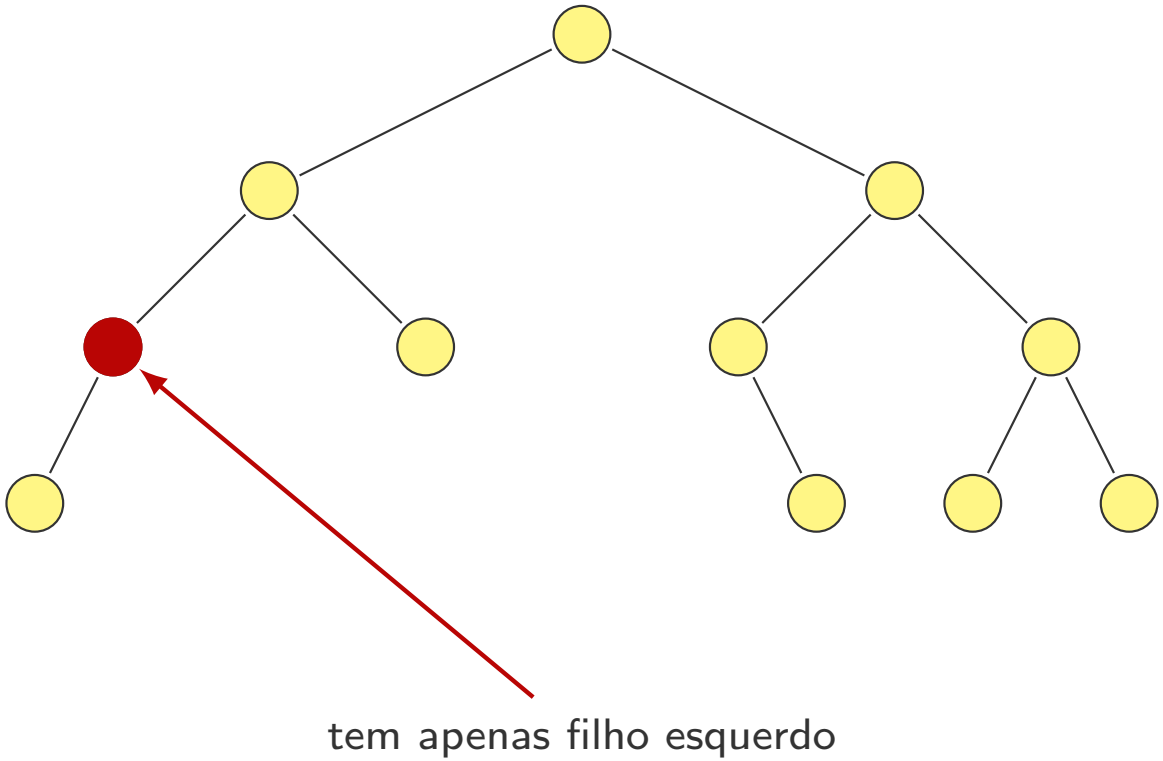
Definições Adicionais



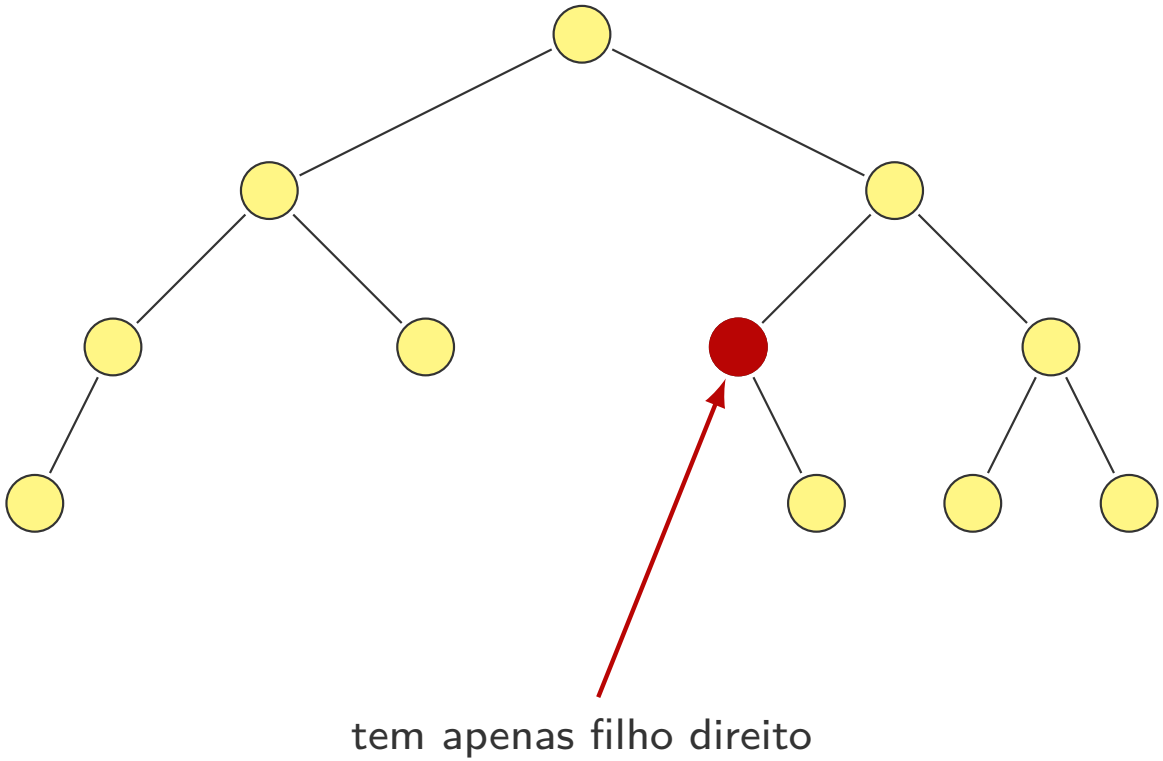
Definições Adicionais



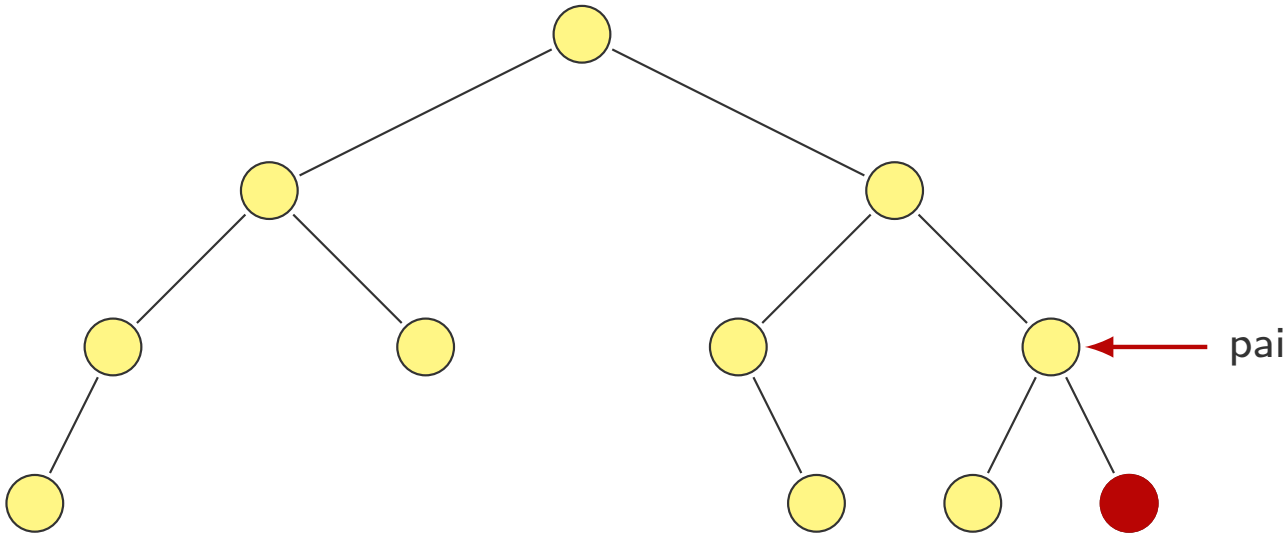
Definições Adicionais



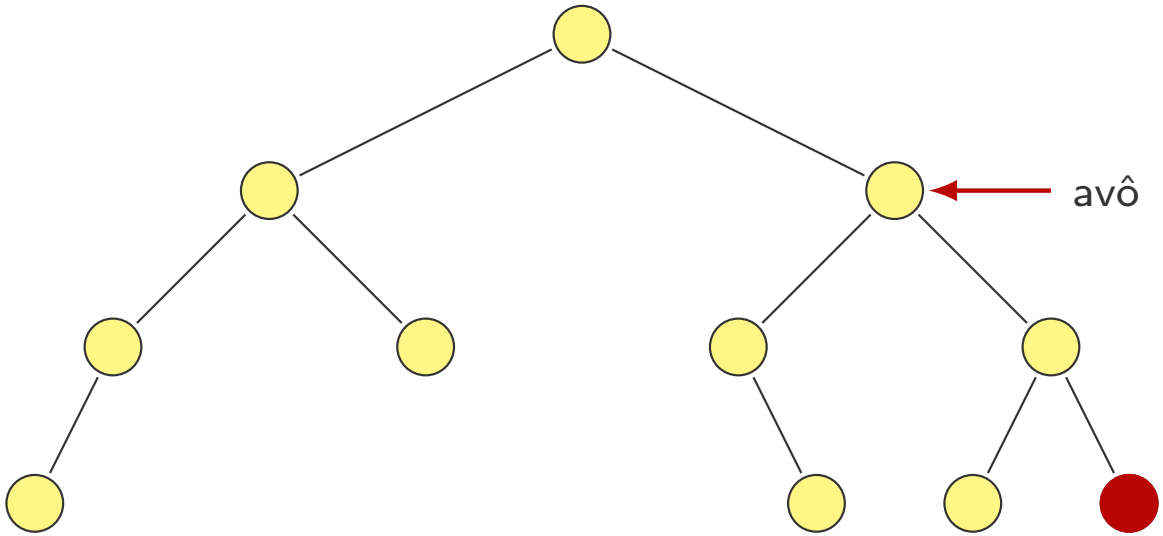
Definições Adicionais



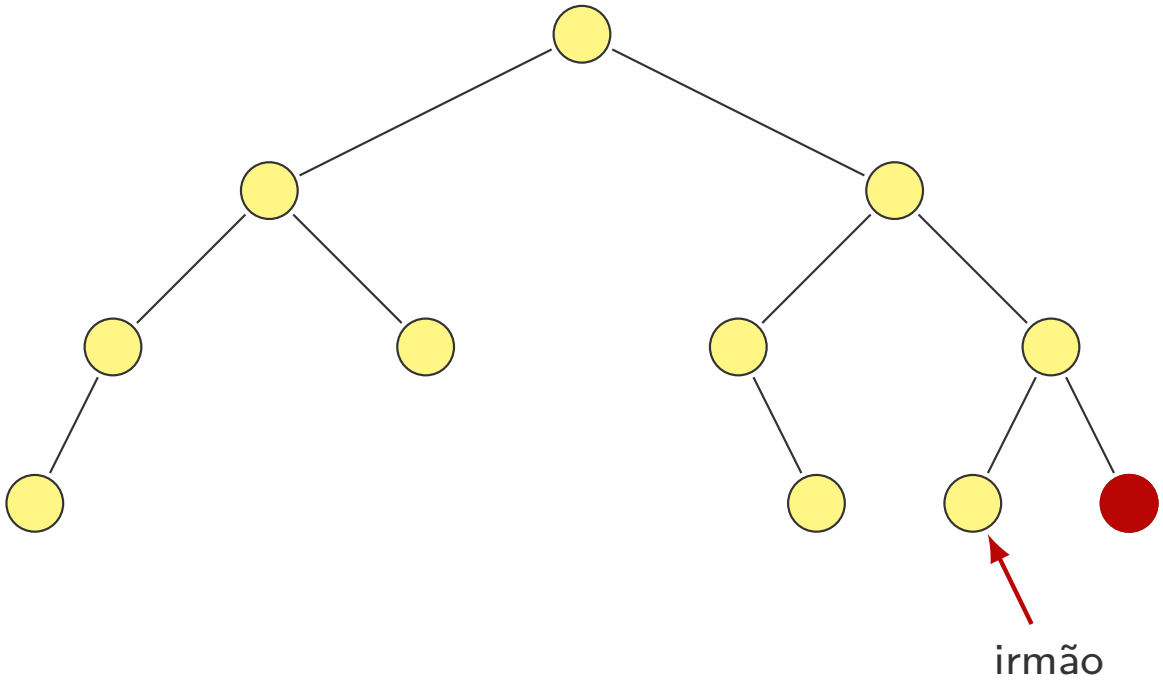
Definições Adicionais



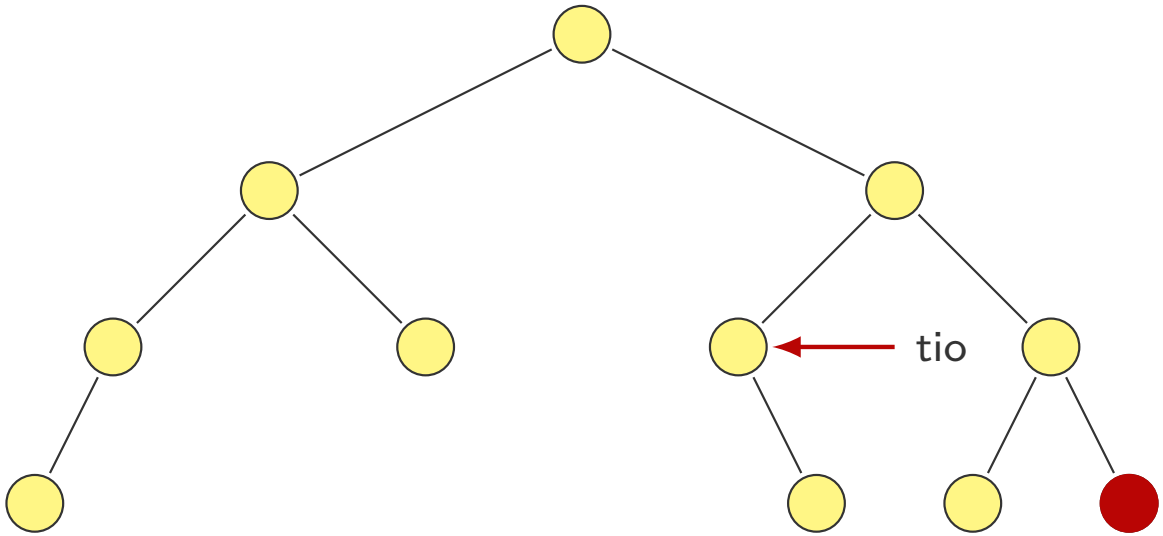
Definições Adicionais



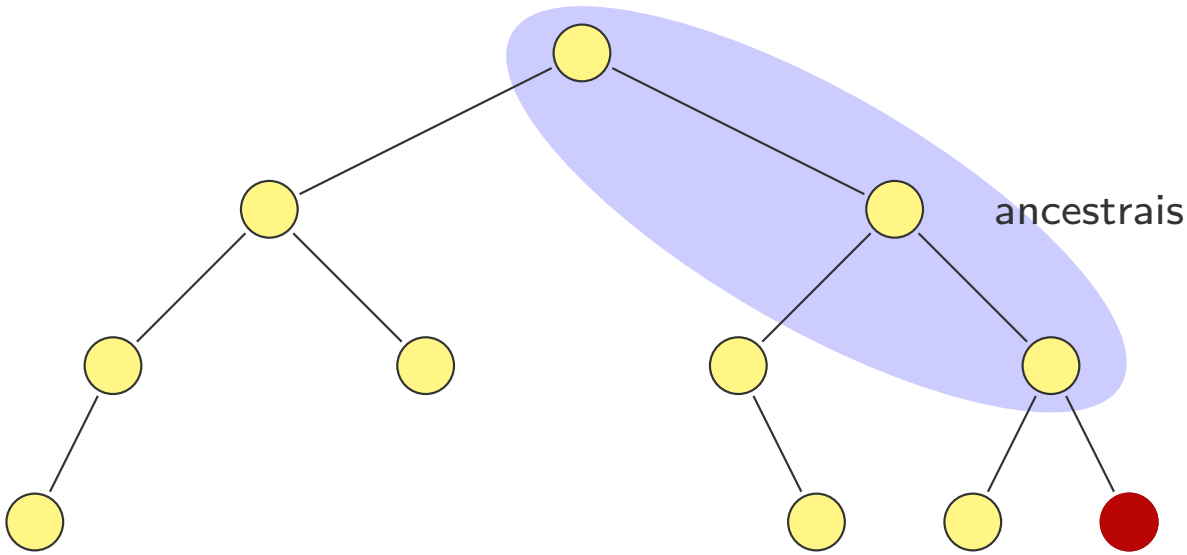
Definições Adicionais



Definições Adicionais

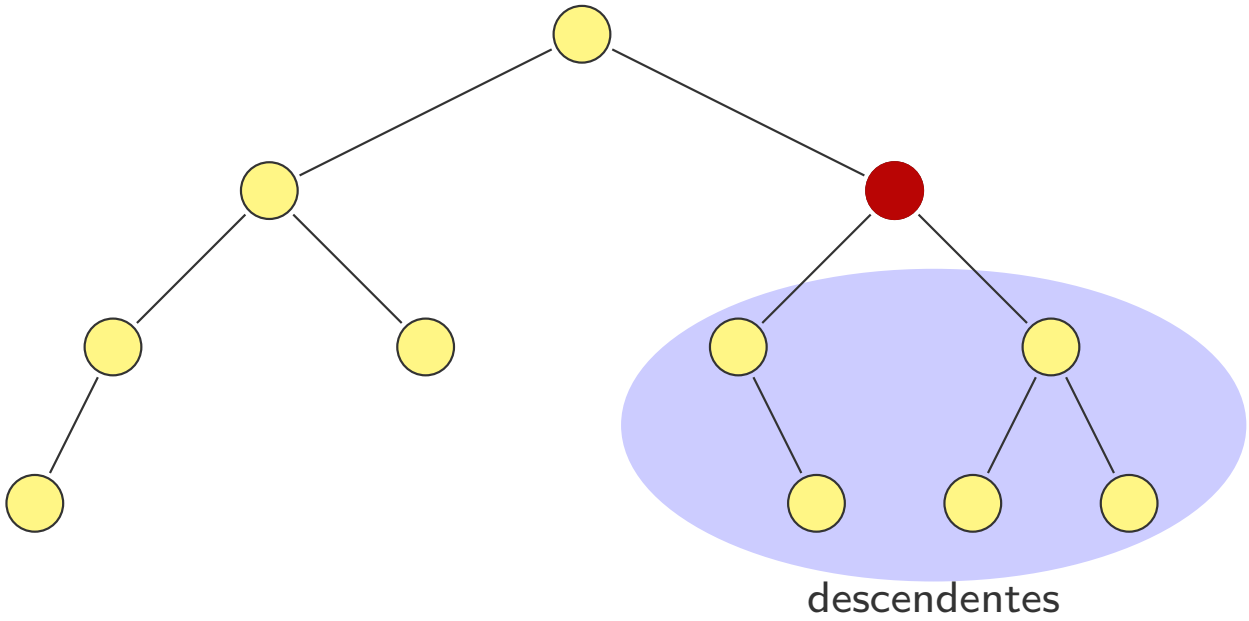


Definições Adicionais

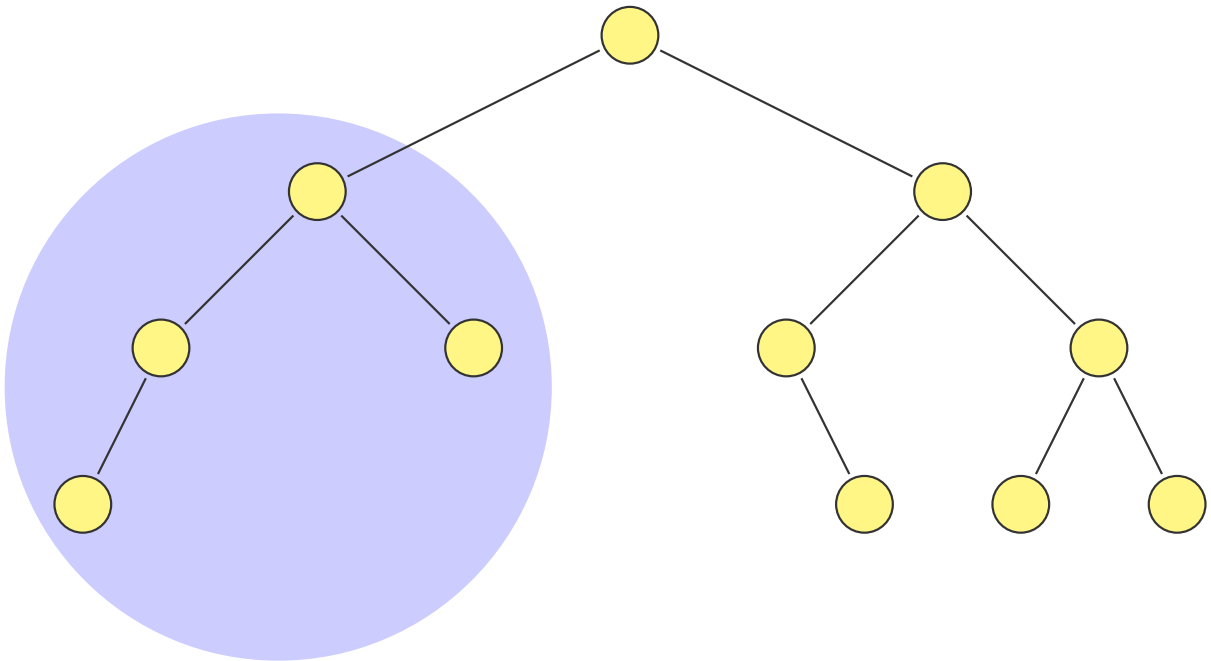


Uma sequência de nós distintos $v_1, v_2, \ldots, v_k$, tal que existe sempre entre nós consecutivos a relação "é filho de" ou "é pai de", é denominada um caminho na árvore.

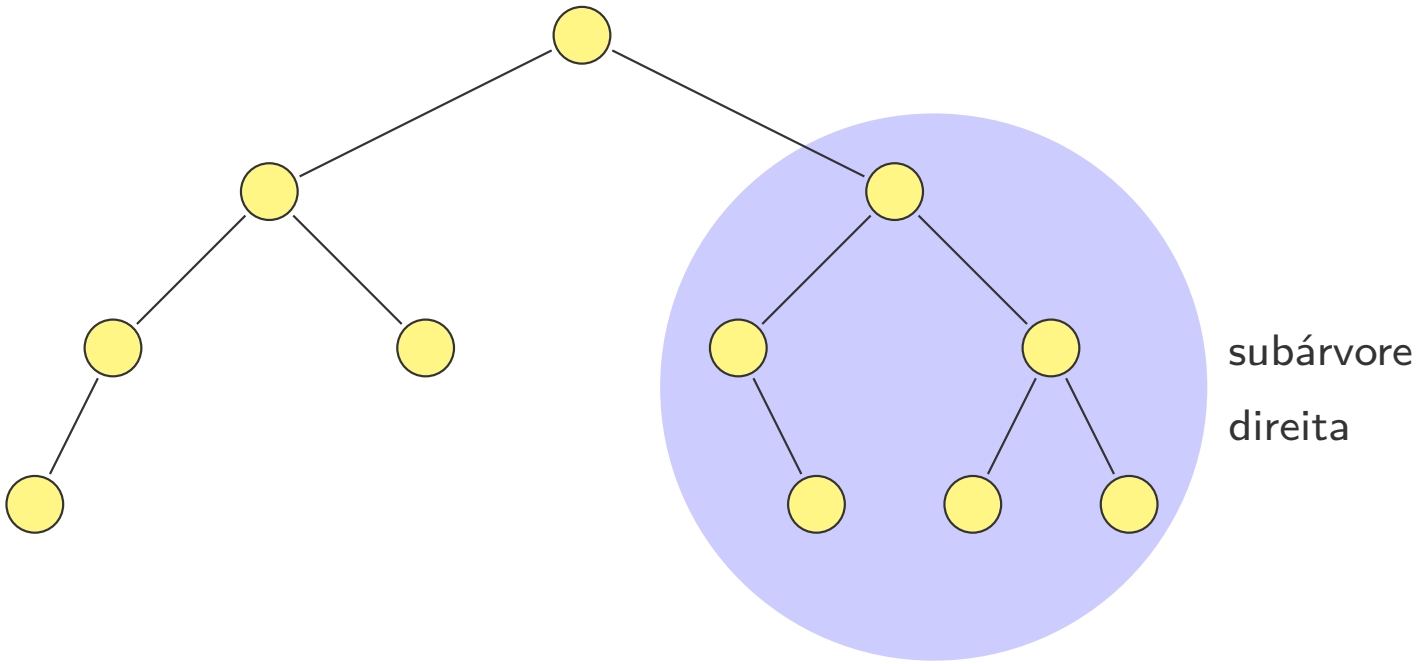
Definições Adicionais



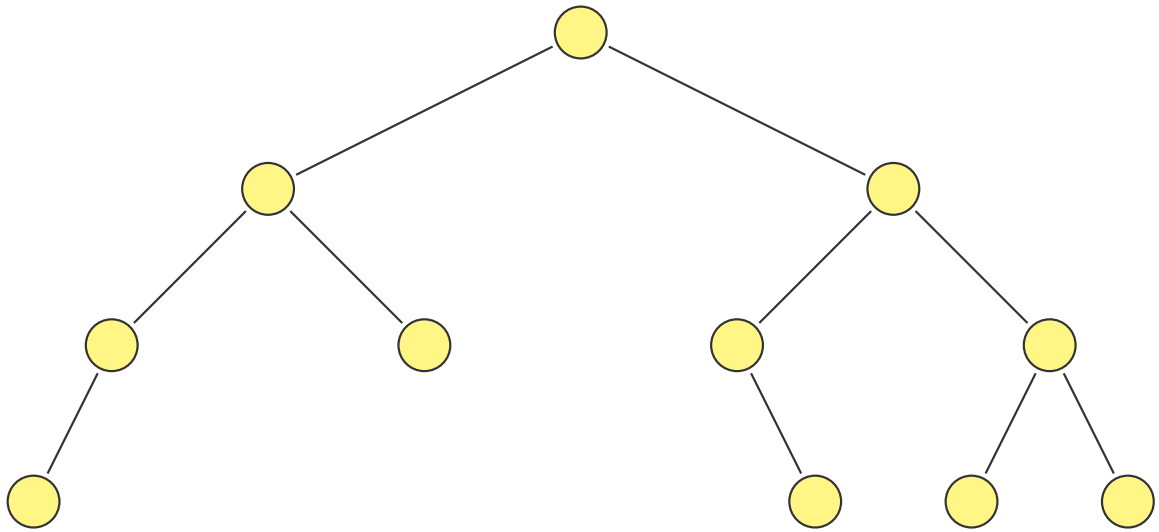
Definições Adicionais



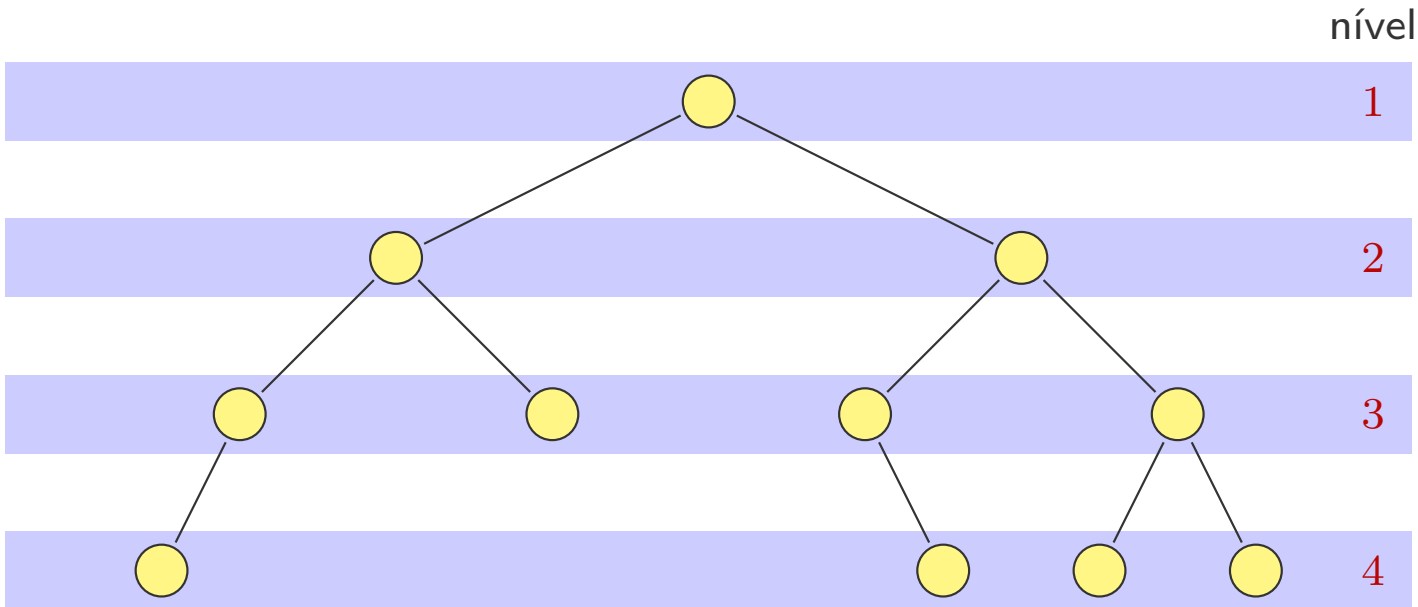
Definições Adicionais



Definições — Profundidade, Nível e Altura

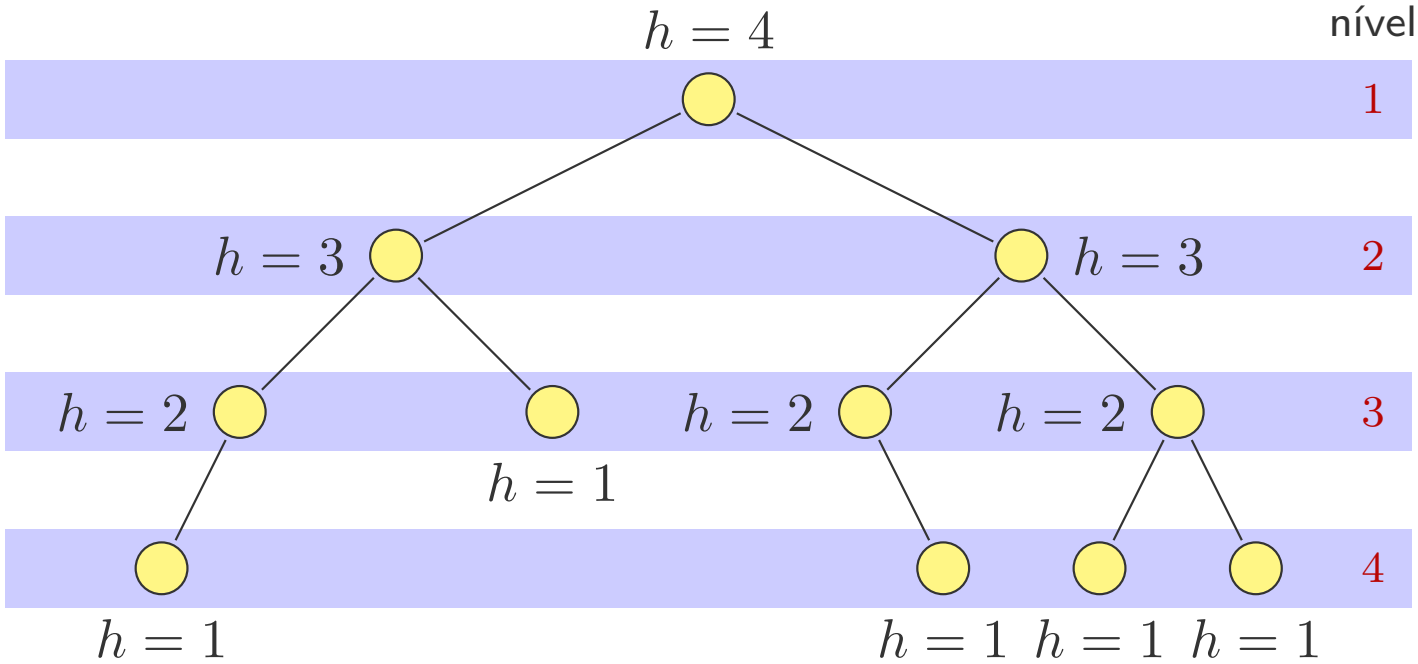


Definições — Profundidade, Nível e Altura



Profundidade de um nó v: Número de nós no caminho de v até a raiz.
Dizemos que todos os nós com profundidade i estão no nível i.

Definições — Profundidade, Nível e Altura

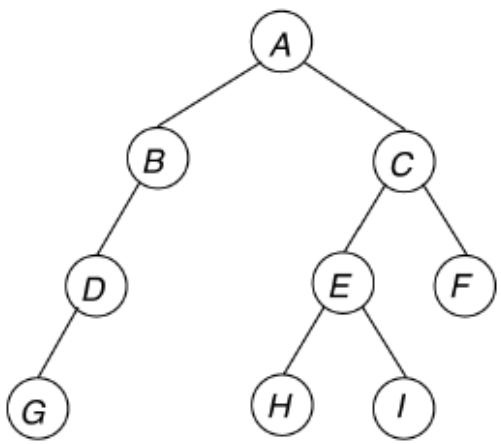


Profundidade de um nó v: Número de nós no caminho de v até a raiz. Dizemos que todos os nós com profundidade i estão no nível i.

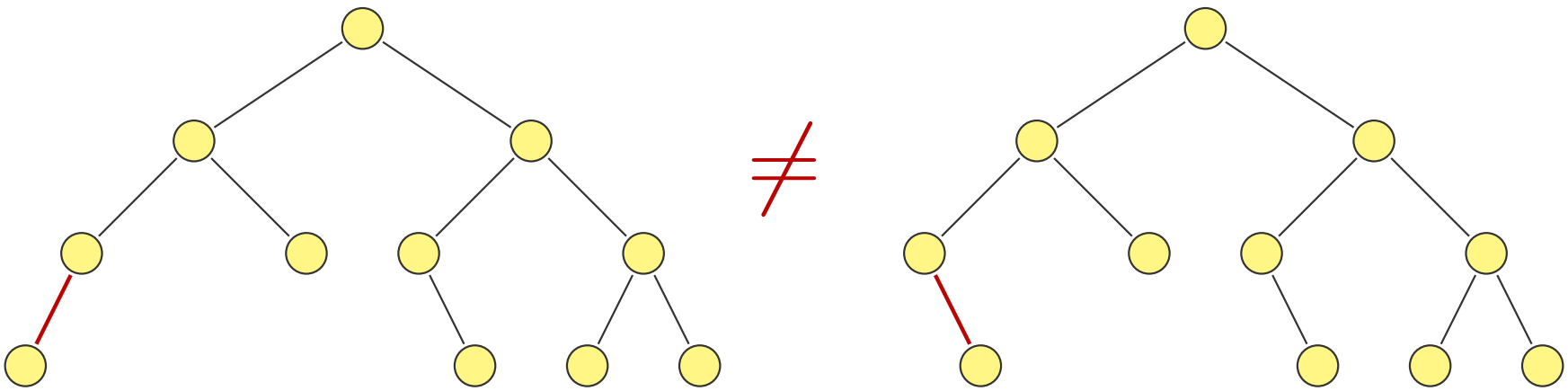
Altura h de um nó v: Número de nós no maior caminho de v até uma folha descendente.

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.



Comparando com atenção



Ordem dos filhos é relevante!

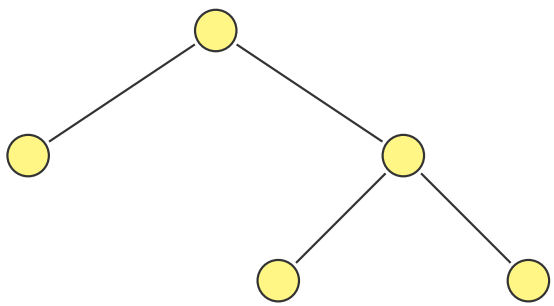
Tipos específicos de árvores binárias

- Árvore estritamente binária: todo nó possui 0 ou 2 filhos.

Tipos específicos de árvores binárias

- Árvore estritamente binária: todo nó possui 0 ou 2 filhos.

- Árvore binária completa: possui a propriedade de que, se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no penúltimo ou no último nível da árvore.

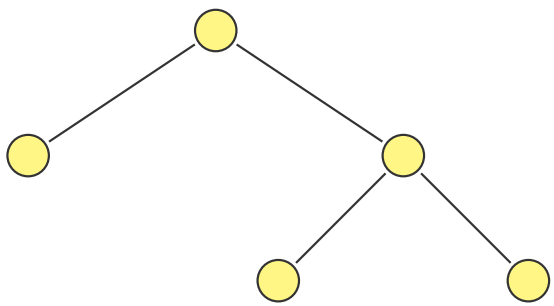


binária completa

Tipos específicos de árvores binárias

- Árvore estritamente binária: todo nó possui 0 ou 2 filhos.

- Árvore binária completa: possui a propriedade de que, se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no penúltimo ou no último nível da árvore.

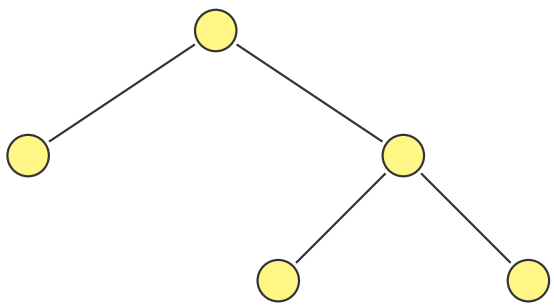


binária completa

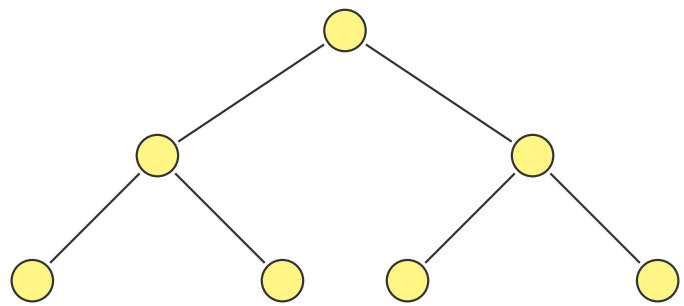
Tipos específicos de árvores binárias

- Árvore estritamente binária: todo nó possui 0 ou 2 filhos.

- Árvore binária completa: possui a propriedade de que, se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no penúltimo ou no último nível da árvore.



binária completa



binária cheia

- Árvore binária cheia: todos os seus nós internos têm dois filhos e todas as folhas estão no último nível da árvore.

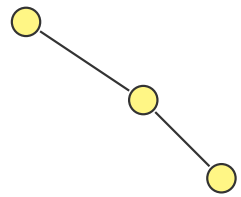
Relação entre altura e número de nós da Árvore Binária

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:

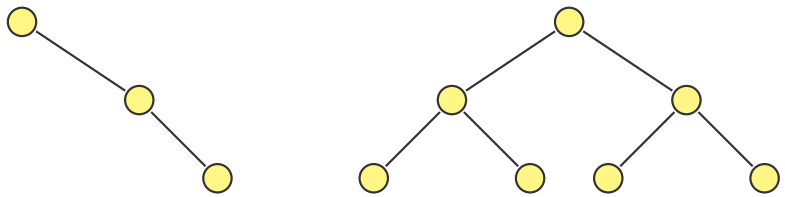
- tem no mínimo h nós



Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

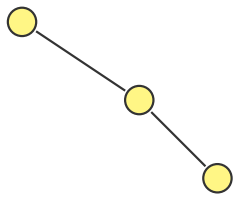
- tem no mínimo h nós
- tem no máximo $2^h - 1$ nós

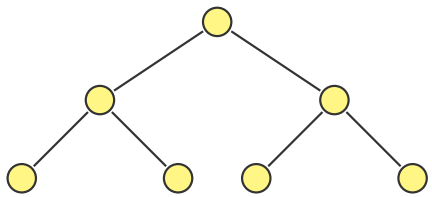


Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



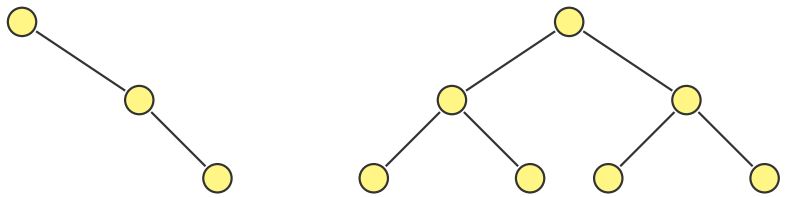


Se a árvore binária tem $n \geq 1$ nós, então:

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



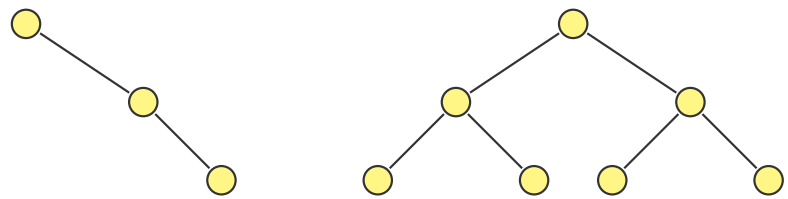
Se a árvore binária tem $n \geq 1$ nós, então:

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

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



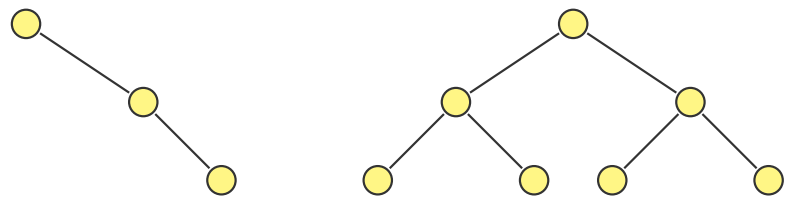
Se a árvore binária tem $n \geq 1$ nós, então:

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

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



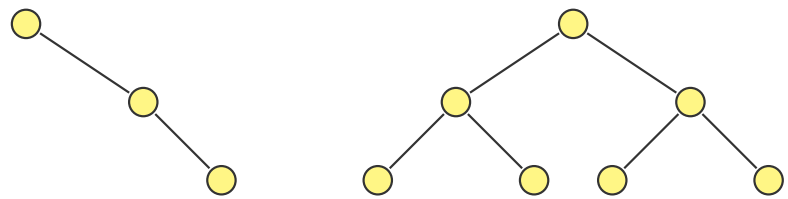
Se a árvore binária tem $n \geq 1$ nós, então:

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



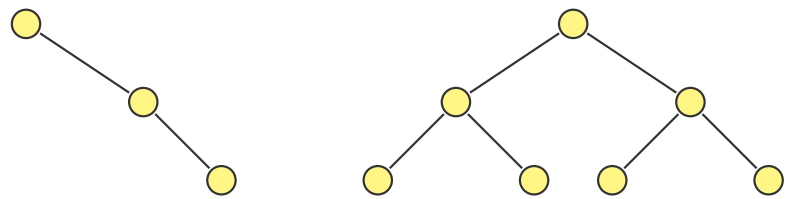
Se a árvore binária tem $n \geq 1$ nós, então:

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

Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



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

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

Árvore Binária - relação entre altura e número de nós

Lema 2: Se T é uma árvore binária completa com $n > 0$ nós, então T possui altura h mínima.

Árvore Binária - relação entre altura e número de nós

Lema 2: Se T é uma árvore binária completa com $n > 0$ nós, então T possui altura h mínima.

Prova: Seja T é uma árvore binária completa com n nós, e seja T' uma árvore binária de altura mínima com n nós.

Árvore Binária - relação entre altura e número de nós

Lema 2: Se T é uma árvore binária completa com $n > 0$ nós, então T possui altura h mínima.

Prova: Seja T é uma árvore binária completa com n nós, e seja T' uma árvore binária de altura mínima com n nós.

Caso 1: Se T' é também completa, então T e T' possuem a mesma altura, isto é, T possui altura mínima.

Árvore Binária - relação entre altura e número de nós

Lema 2: Se T é uma árvore binária completa com $n > 0$ nós, então T possui altura h mínima.

Prova: Seja T é uma árvore binária completa com n nós, e seja T' uma árvore binária de altura mínima com n nós.

Caso 1: Se T' é também completa, então T e T' possuem a mesma altura, isto é, T possui altura mínima.

Caso 2: Se T' não é completa, efetua-se a seguinte operação: retirar uma folha w de seu último nível e tornar w o filho de algum nó v que possui alguma de suas subárvores vazias, localizado em algum nível acima do penúltimo.

Árvore Binária - relação entre altura e número de nós

Lema 2: Se T é uma árvore binária completa com $n > 0$ nós, então T possui altura h mínima.

Prova: Seja T é uma árvore binária completa com n nós, e seja T' uma árvore binária de altura mínima com n nós.

Caso 1: Se T' é também completa, então T e T' possuem a mesma altura, isto é, T possui altura mínima.

Caso 2: Se T' não é completa, efetua-se a seguinte operação: retirar uma folha w de seu último nível e tornar w o filho de algum nó v que possui alguma de suas subárvores vazias, localizado em algum nível acima do penúltimo.

Repete-se a operação até que não seja mais possível realizá-la, isto é, até que a árvore T'', resultante da transformação, seja completa.

Árvore Binária - relação entre altura e número de nós

Continuação da prova

T'' não pode ter altura inferior a T', pois T' é mínima.

Árvore Binária - relação entre altura e número de nós

Continuação da prova

T'' não pode ter altura inferior a T', pois T' é mínima.

T'' não pode ter altura superior a T', pois nenhum nó foi movido para baixo.

Árvore Binária - relação entre altura e número de nós

Continuação da prova

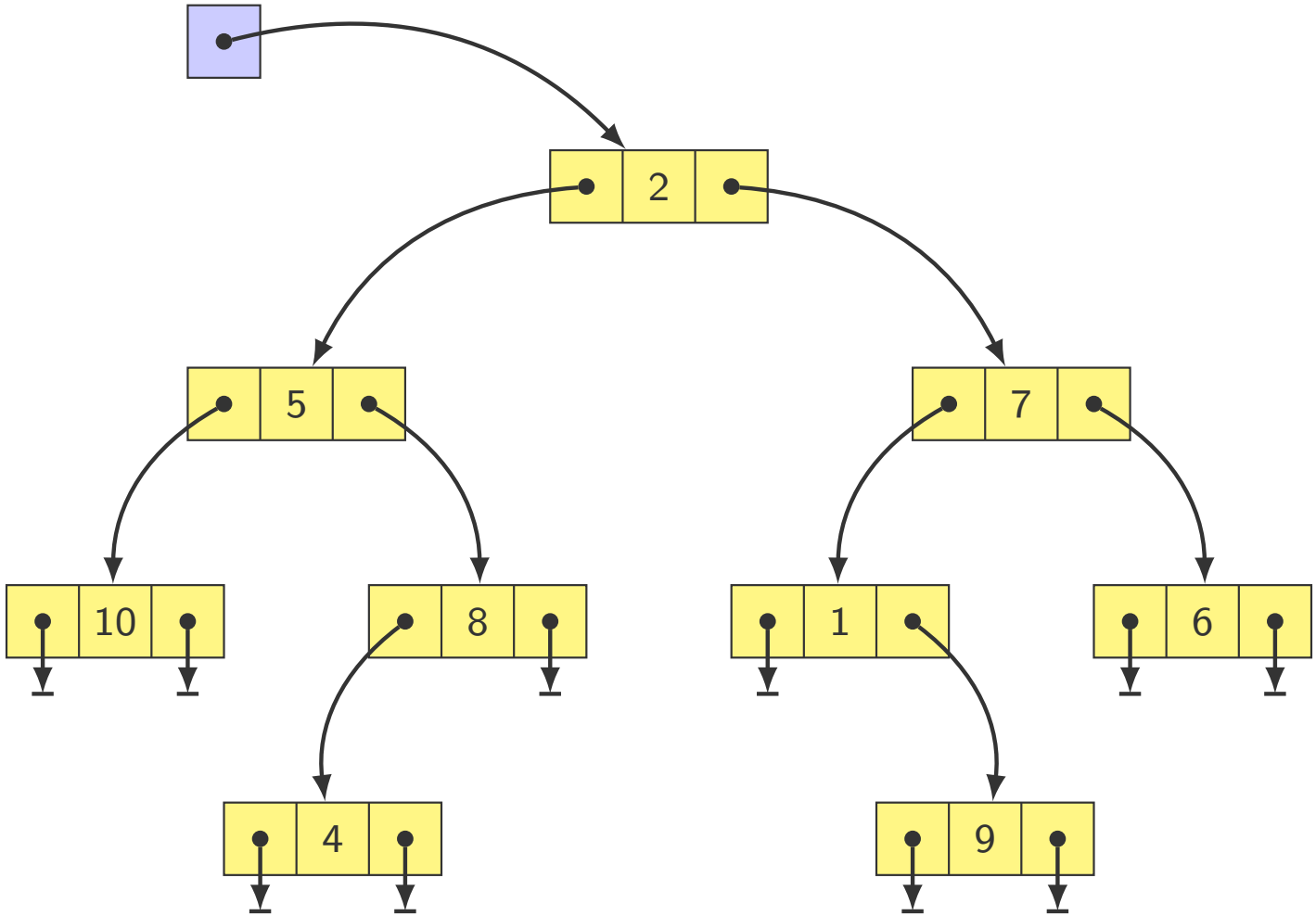
T'' não pode ter altura inferior a T', pois T' é mínima.

T'' não pode ter altura superior a T', pois nenhum nó foi movido para baixo.

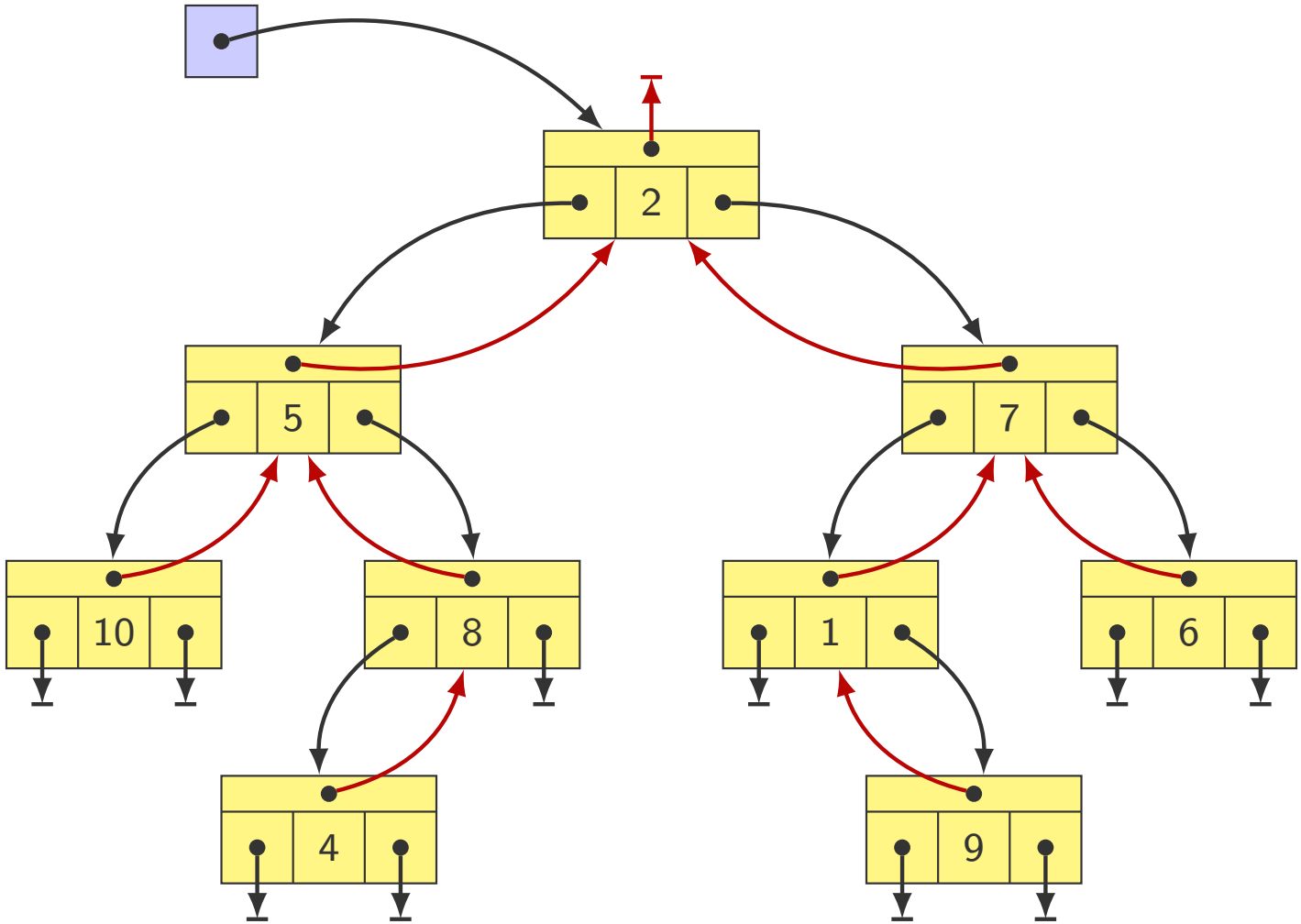
Então as alturas de T' e T'' são iguais. Como T' é completa, conclui-se que as alturas de T e T'' também coincidem. Isto é, T possui altura mínima. $\square$

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++

```
1  #ifndef NODE_H
2  #define NODE_H
3
4  template<typename Type>
5  struct Node
6  {
7      // atributos
8      Type data;
9      Node<Type>* left;
10     Node<Type>* right;
11
12     // Construtor
13     Node(int data, Node<Type>* left, Node<Type>* right)
14     {
15         this->data = data;
16         this->left = left;
17         this->right = right;
18     }
19 };
20
21 #endif /* NODE_H */
```

Implementação da Árvore em C++

```
1  #ifndef BINARY_TREE_H
2  #define BINARY_TREE_H
3  #include "Node.h"
4
5  template<typename Type>
6  class BinaryTree {
7  private:
8      Node<Type> *m_root; // ponteiro para o node raiz
9
10 public:
11     // Cria arvore vazia
12     BinaryTree();
13
14     // Cria arvore a partir de duas outras
15     // As arvores passadas por parametro ficam vazias
16     BinaryTree(const Type& d, BinaryTree<Type>& b1,
17                BinaryTree<Type>& b2);
18
19     // retorna true se arvore contem valor
20     bool contains(const Type& val) const;
```

Implementação da Árvore em C++

```
21      // retorna true se e somente se arvore vazia
22      bool empty() const;
23
24      // imprime os valores no terminal
25      void print() const;
26
27      // deixa arvore vazia
28      void clear();
29
30      // destrutor
31      ~BinaryTree();
32
33      // funcoes deletadas
34      BinaryTree(const BinaryTree& b) = delete;
35      BinaryTree& operator=(const BinaryTree& b) = delete;
36  };
37
38  #endif /* BINARY_TREE_H */
```

Exercício

- Implementar a árvore binária.

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

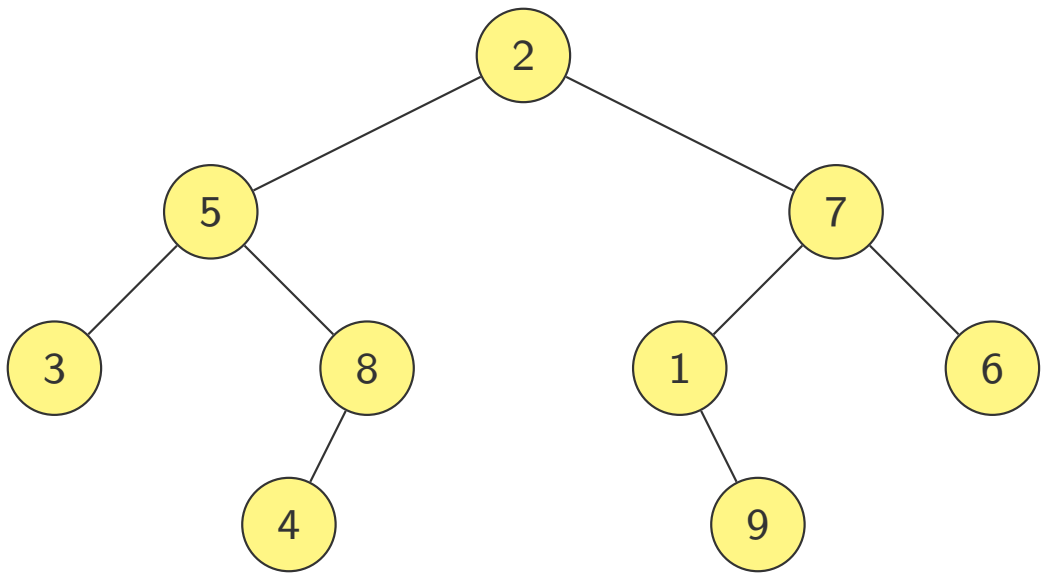
A pré-ordem

- 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

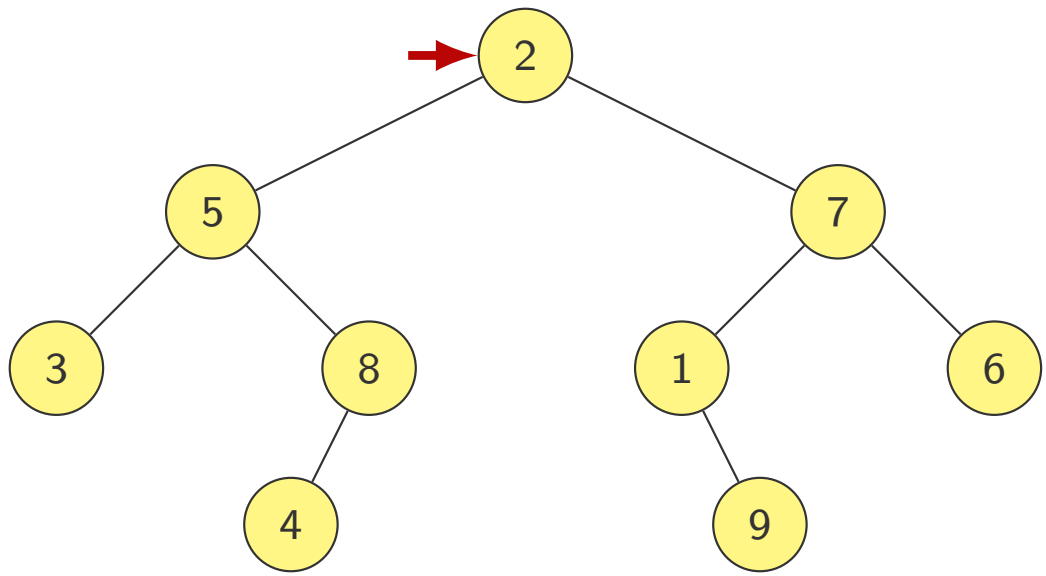


Ex:

Percorrendo os nós — Pré-ordem

A pré-ordem

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

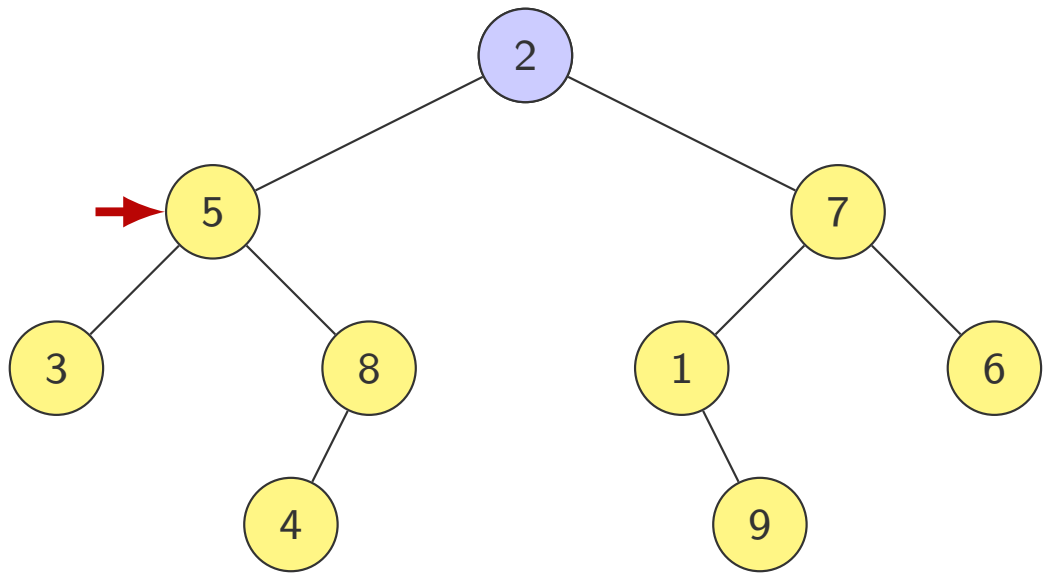


Ex:

Percorrendo os nós — Pré-ordem

A pré-ordem

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

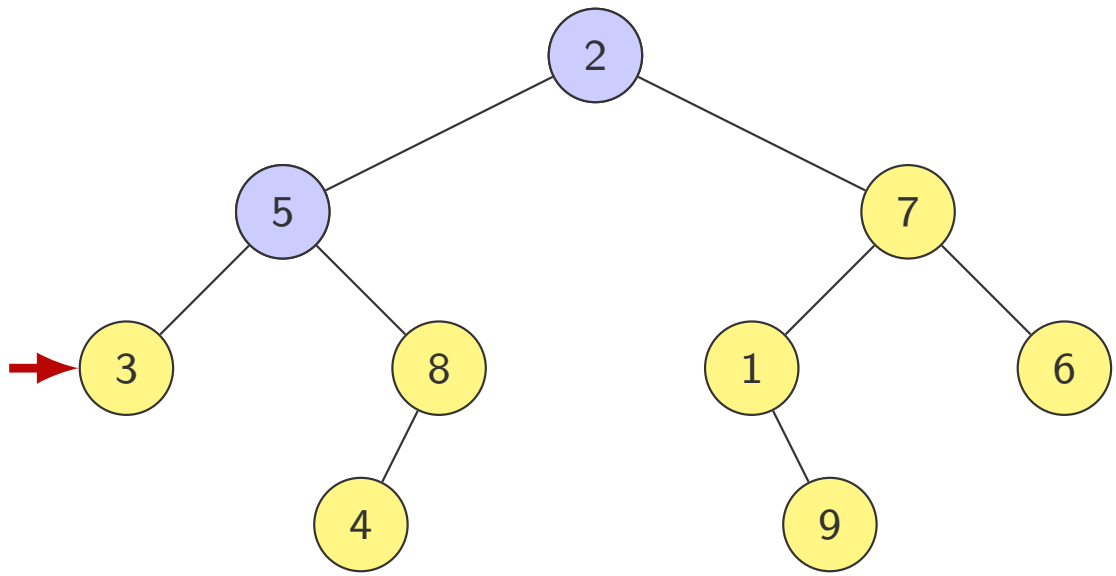


Ex: 2,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

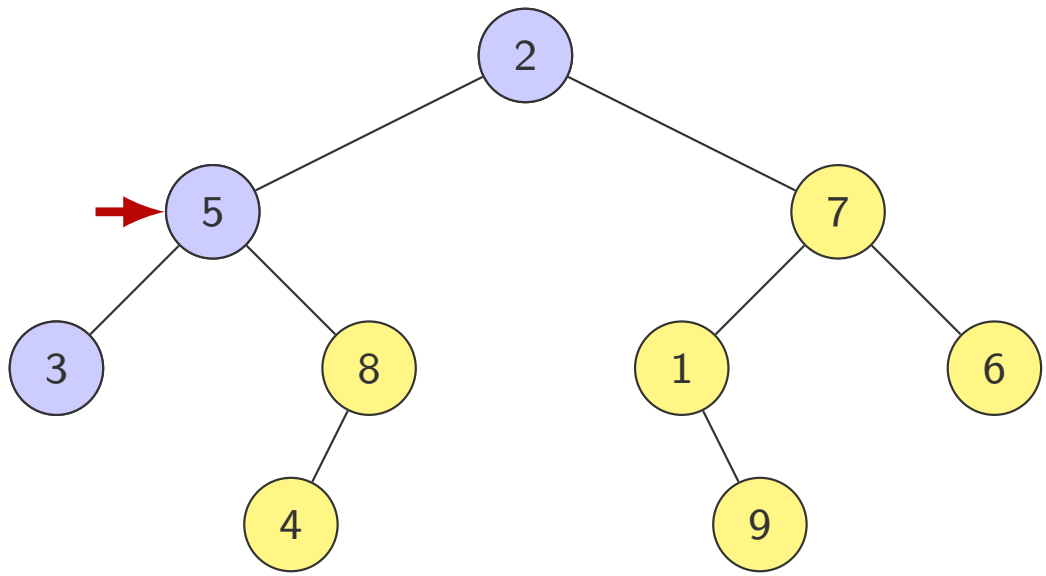


Ex: 2, 5,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

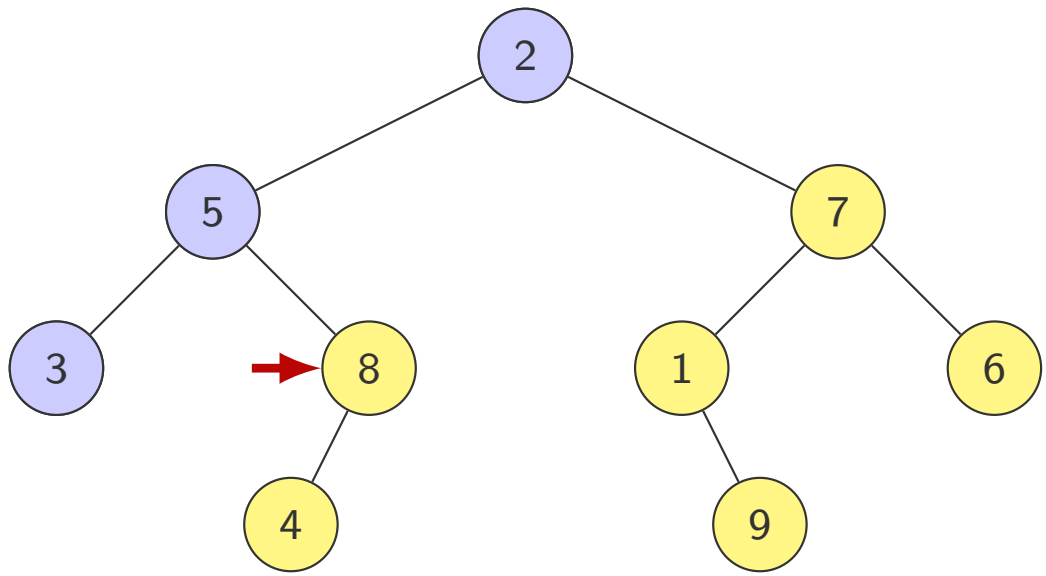


Ex: 2, 5, 3,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

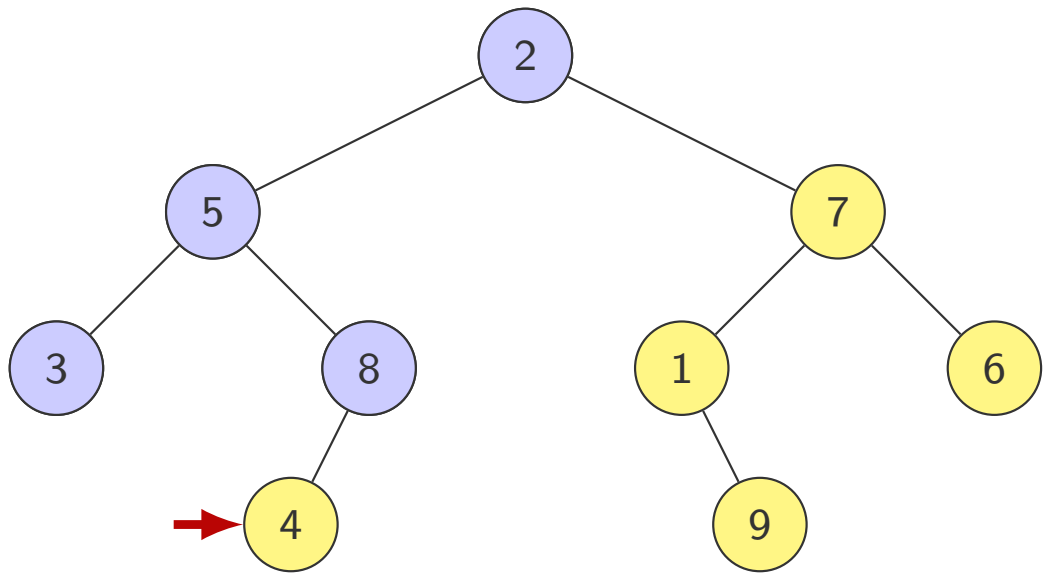


Ex: 2, 5, 3,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

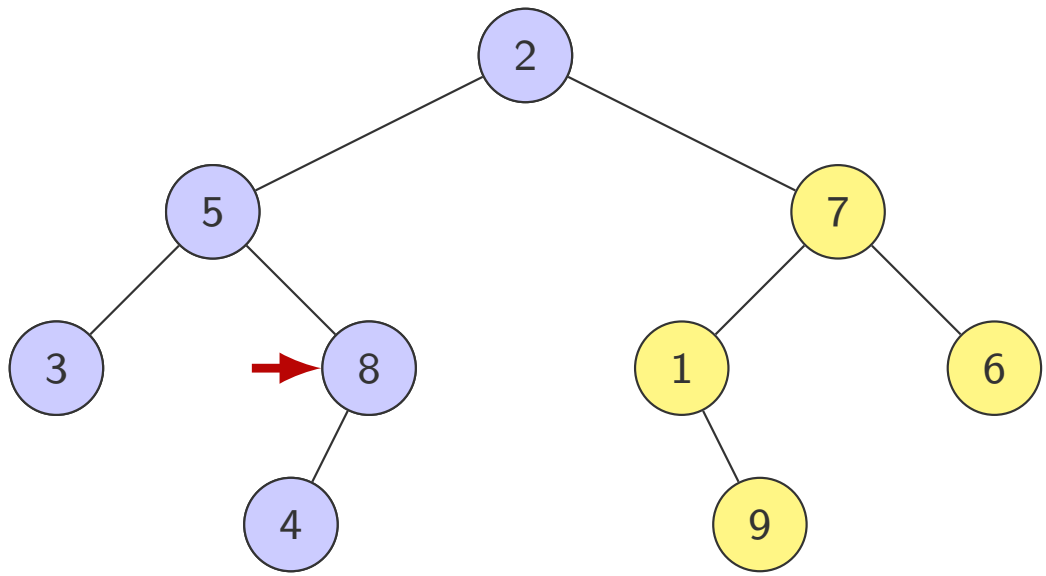


Ex: 2, 5, 3, 8,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

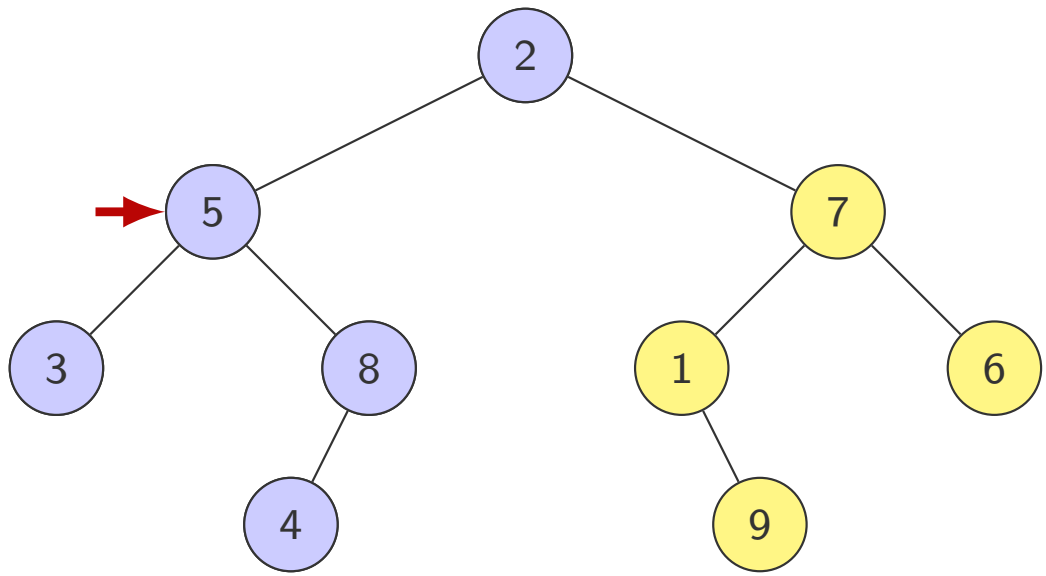


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

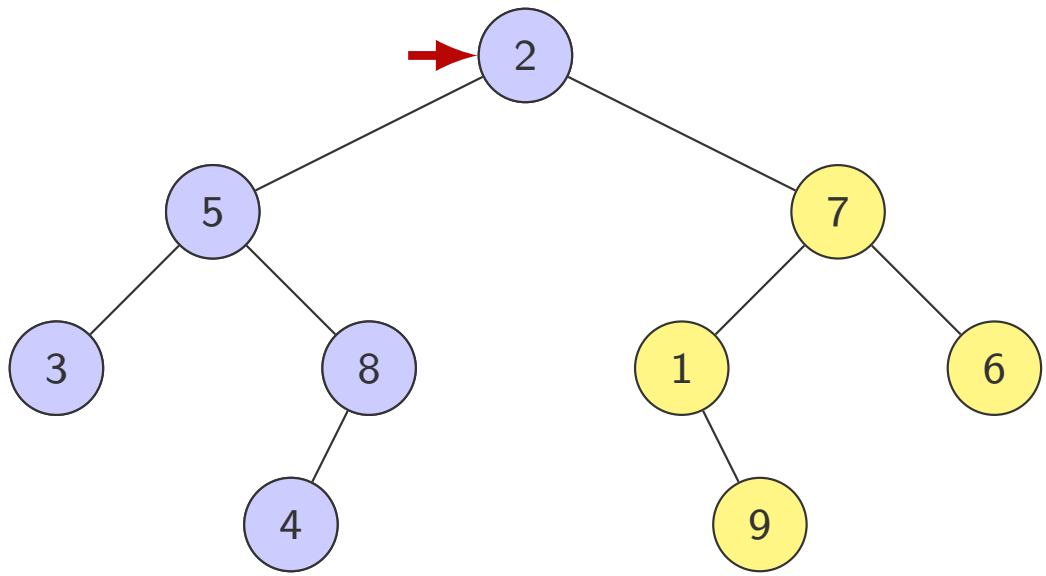


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

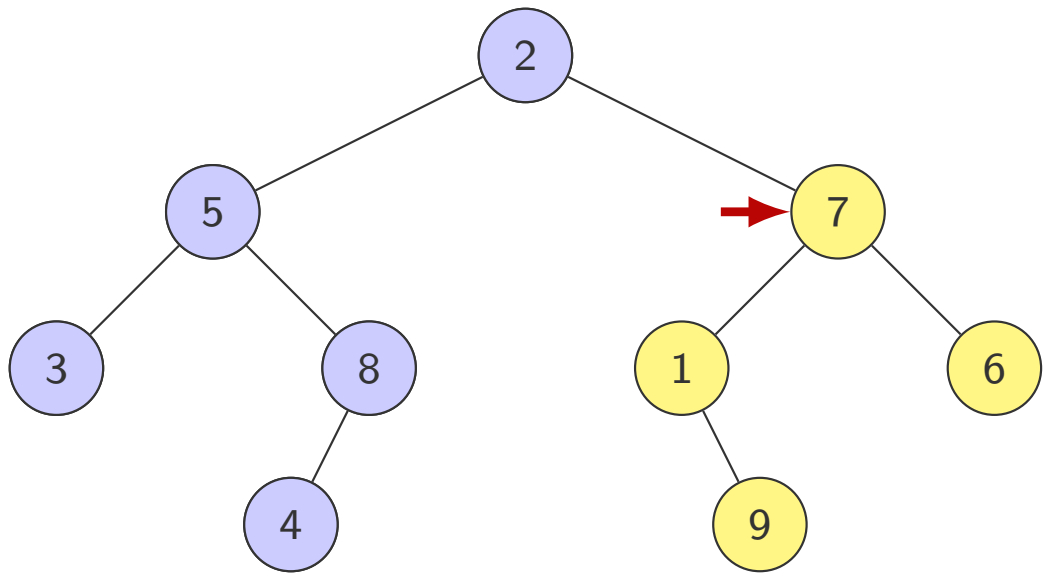


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

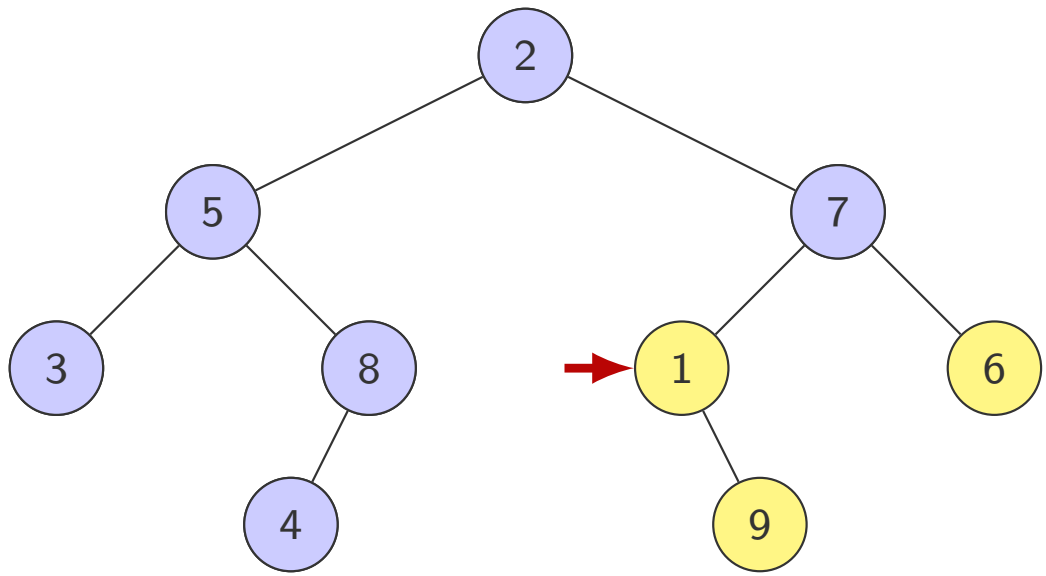


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

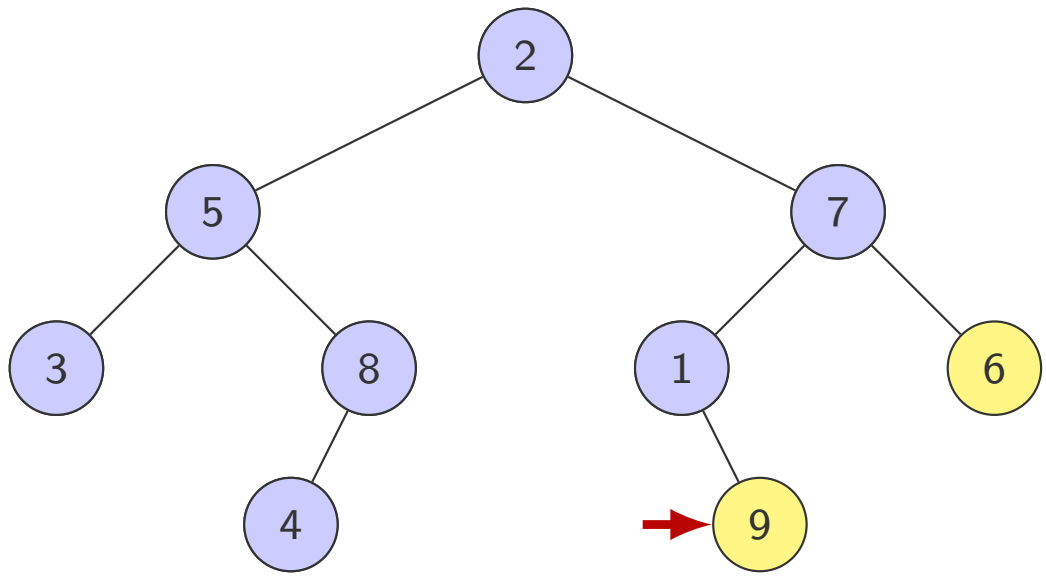


Ex: 2, 5, 3, 8, 4, 7,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

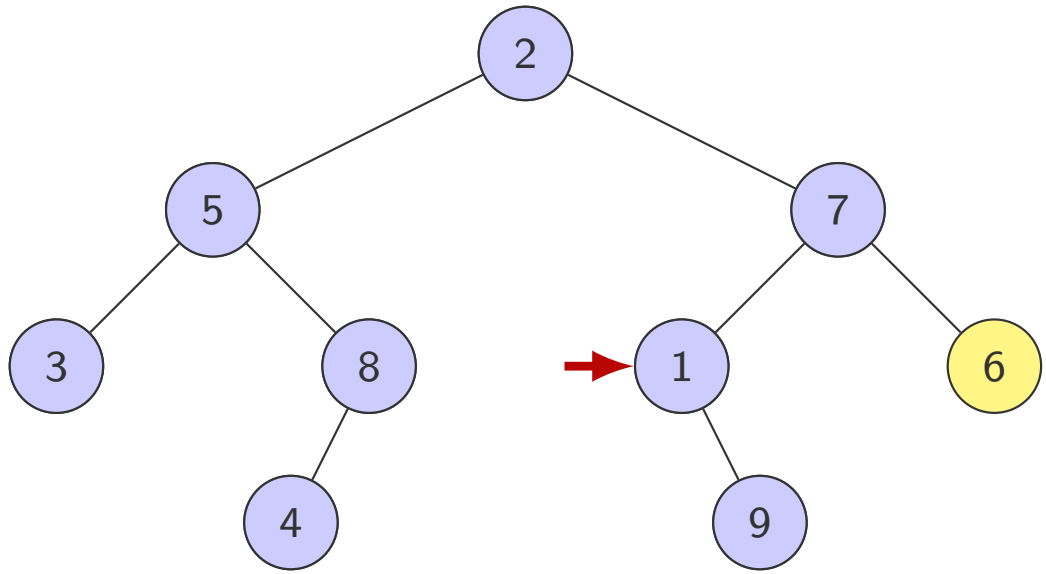


Ex: 2, 5, 3, 8, 4, 7, 1,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

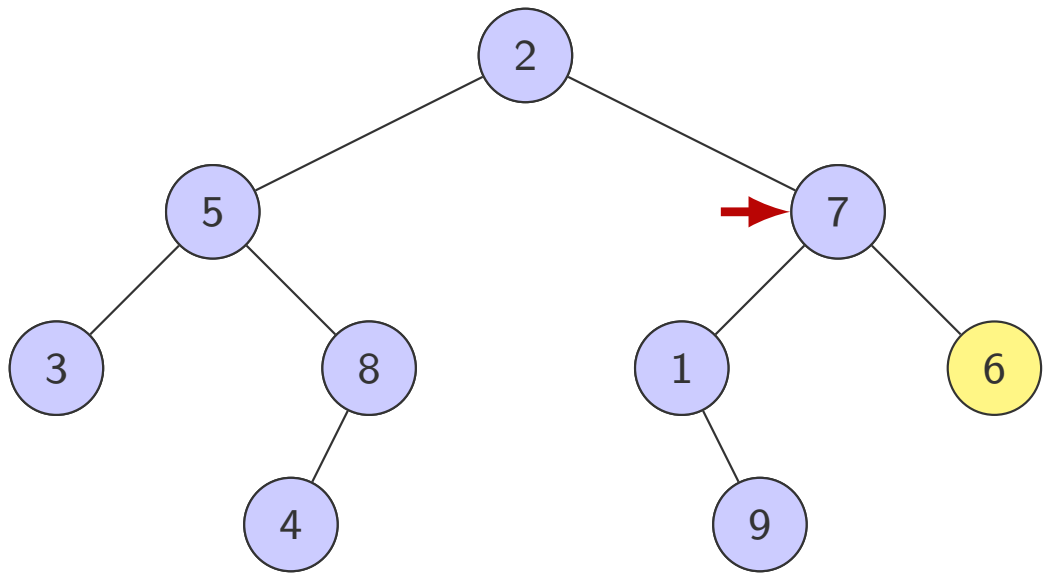


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

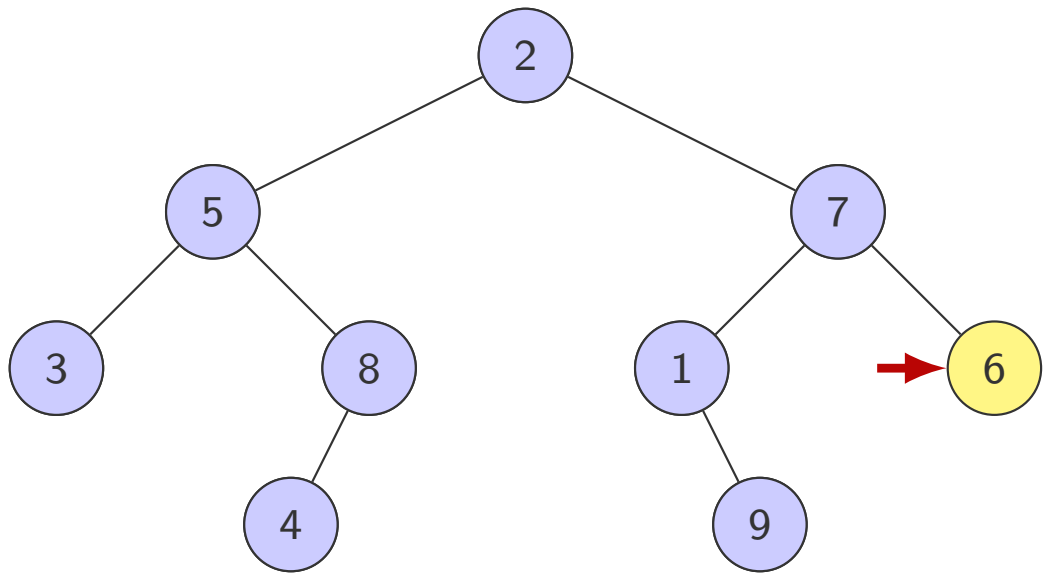


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

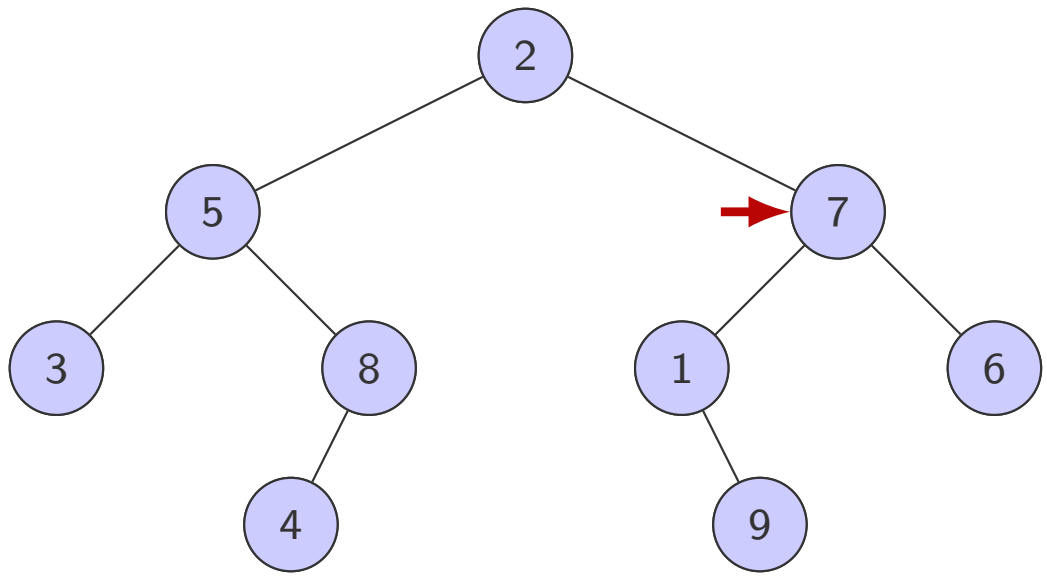


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós — Pré-ordem

A pré-ordem

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

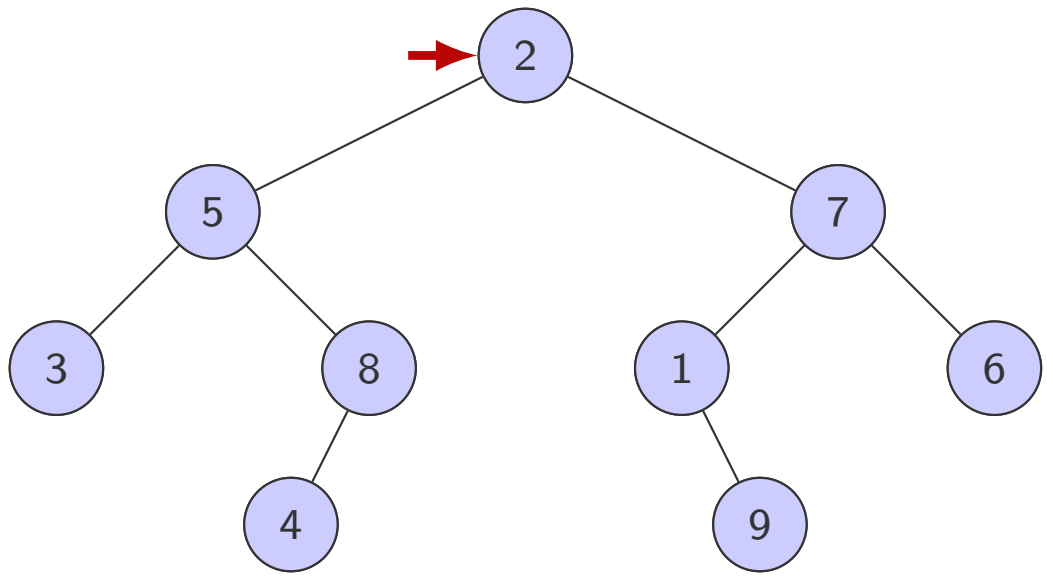


Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

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

Algorithm preorder(ptr)

Require: ptr (pointer to node)

```
1: if ptr ≠ NULL then
2:     visit(ptr)
3:     preorder(ptr→left)
4:     preorder(ptr→right)
5: end if
```

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

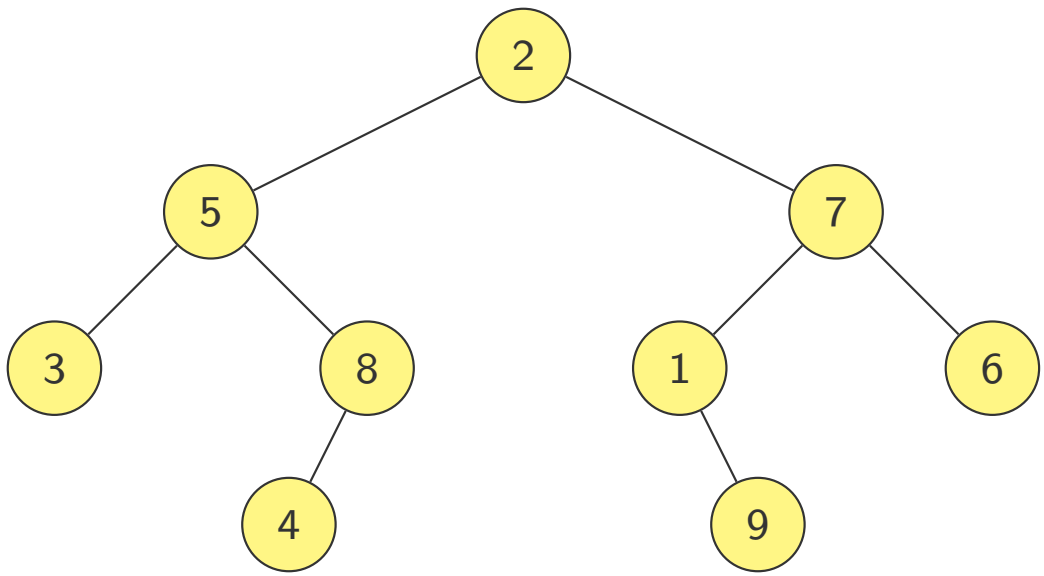
A pós-ordem

- 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

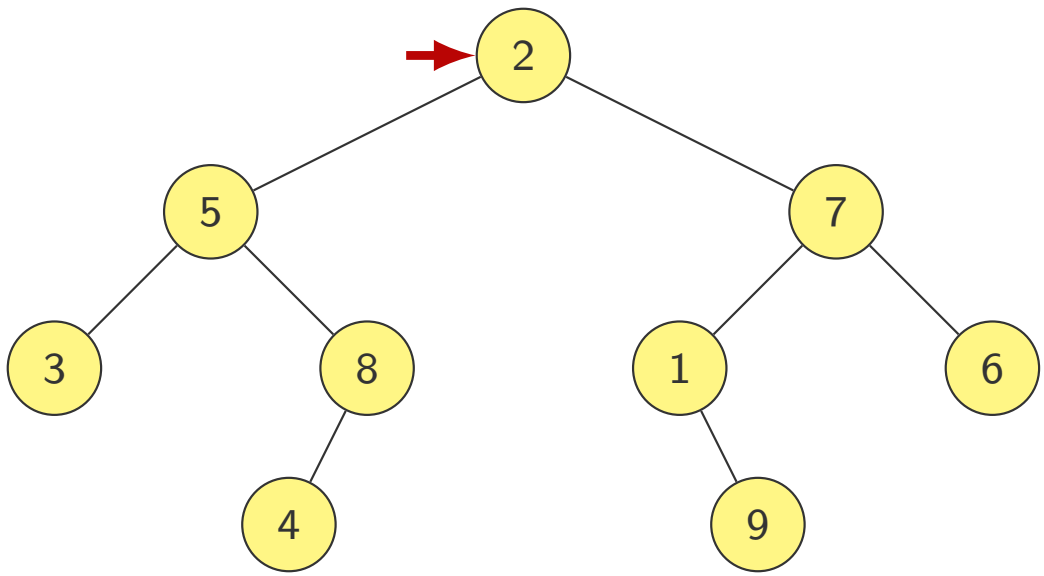


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

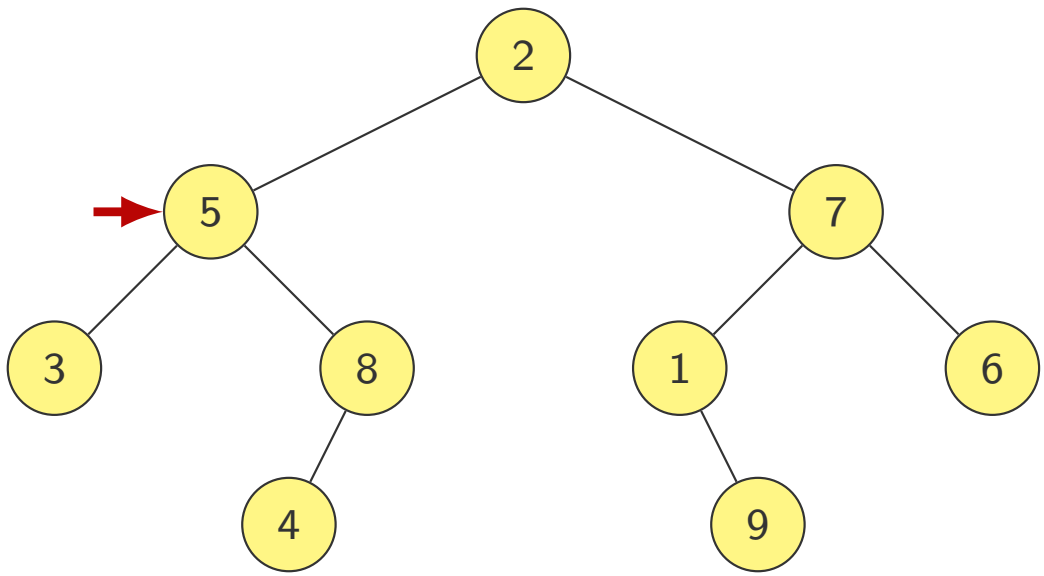


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

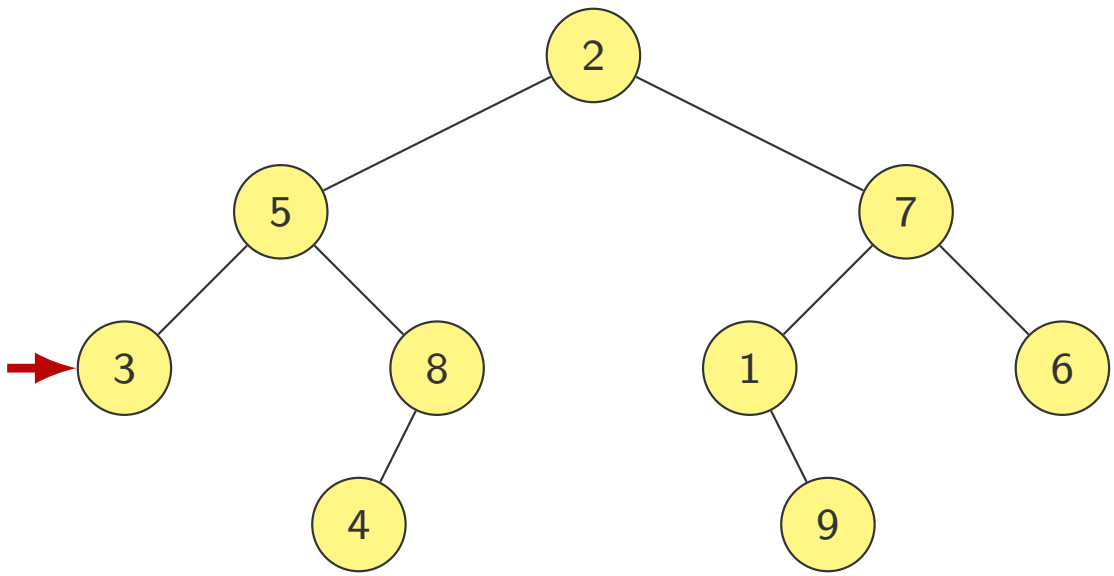


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

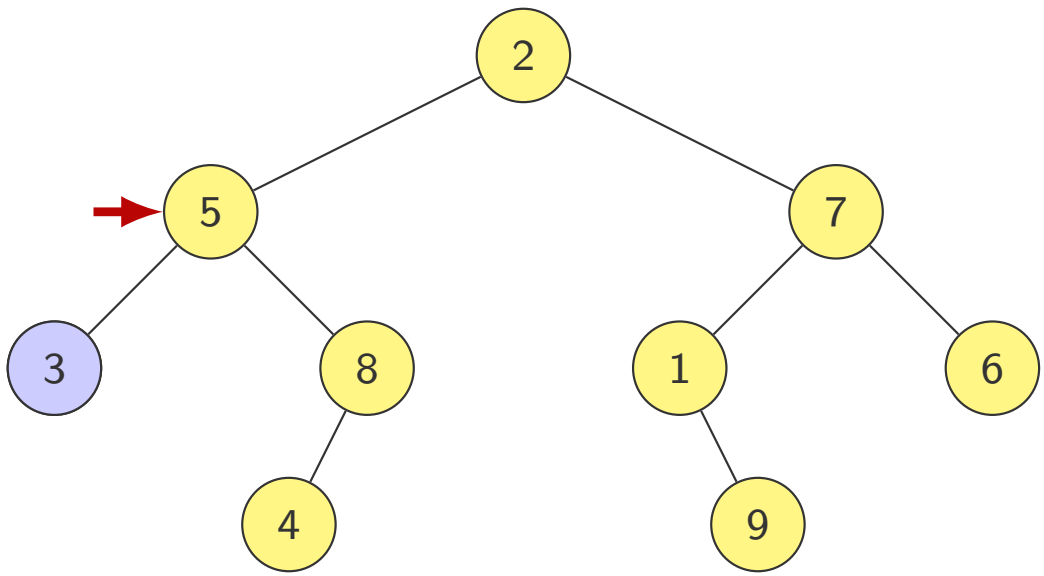


Ex:

Percorrendo os nós — Pós-ordem

A pós-ordem

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

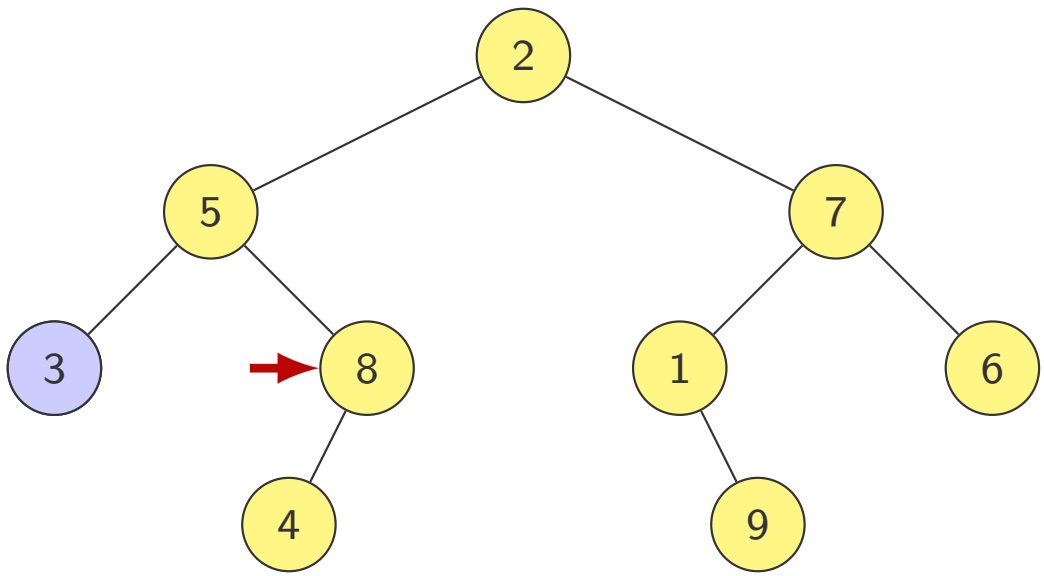


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

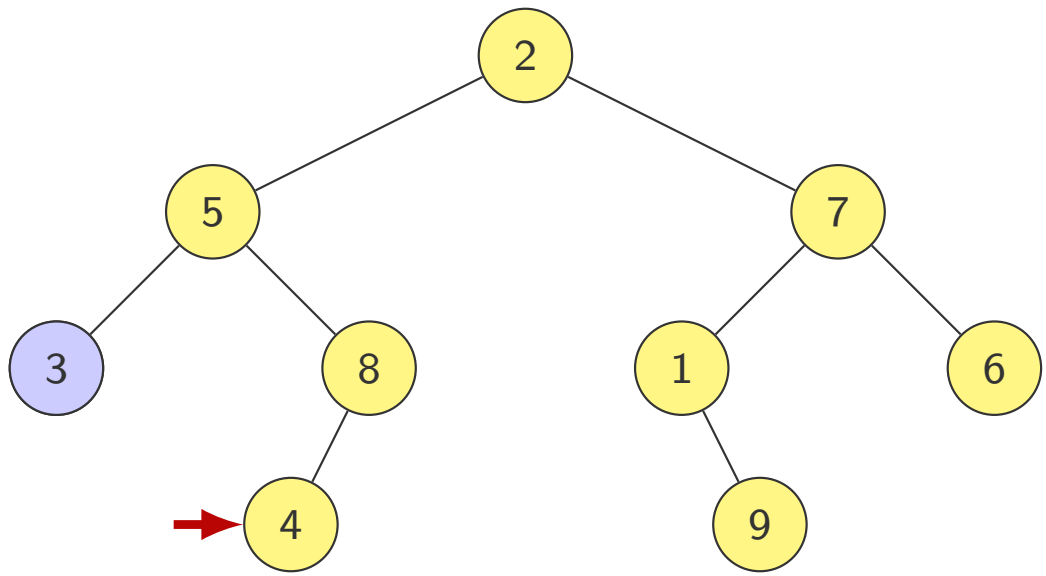


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

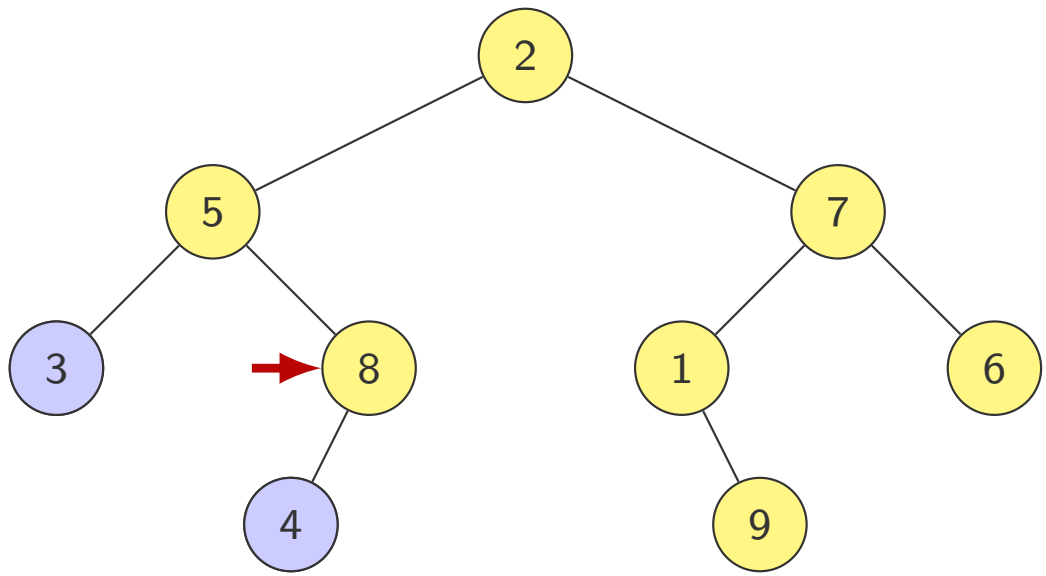


Ex: 3,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

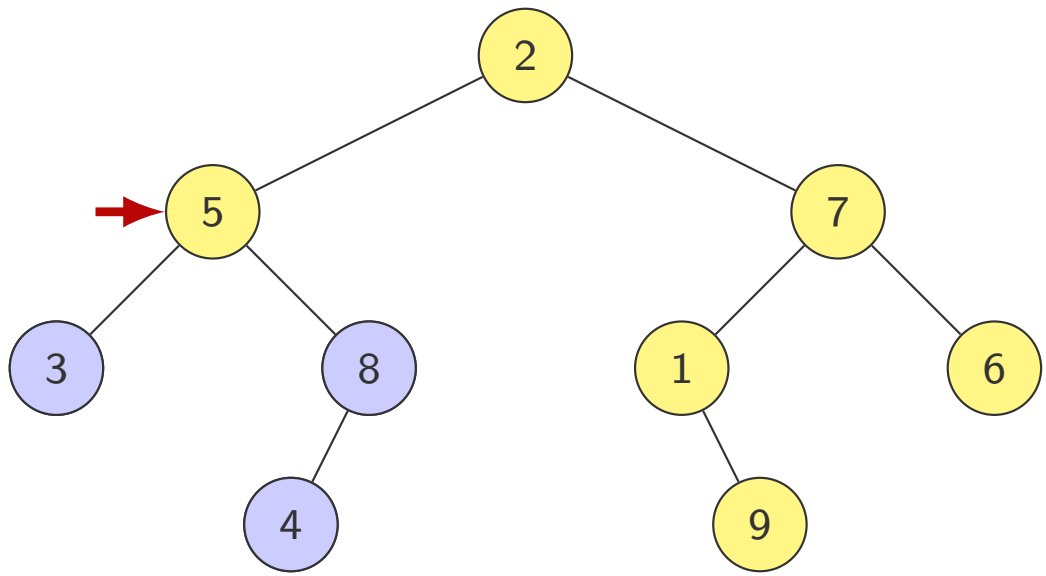


Ex: 3, 4,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

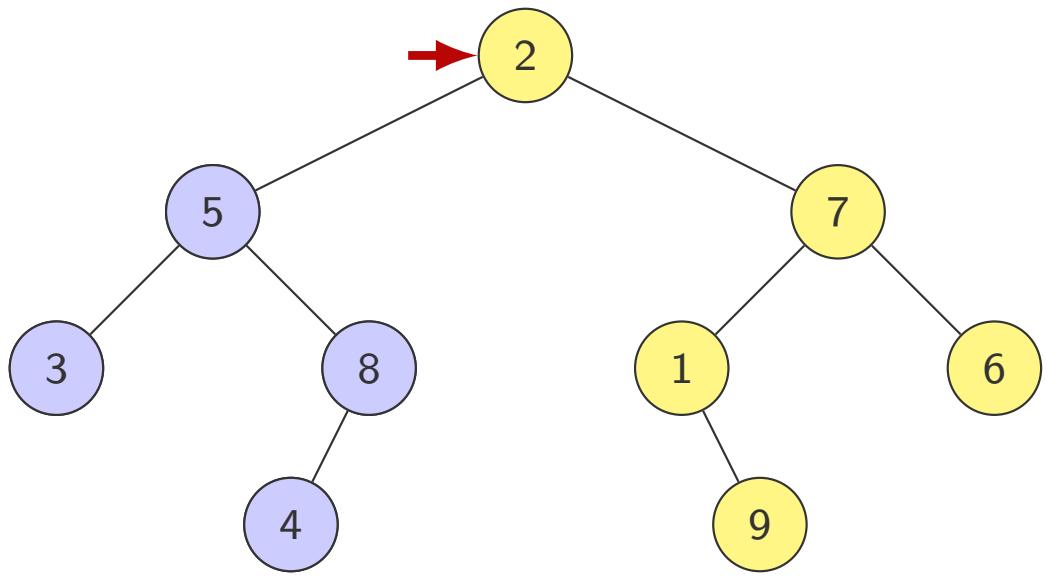


Ex: 3, 4, 8,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

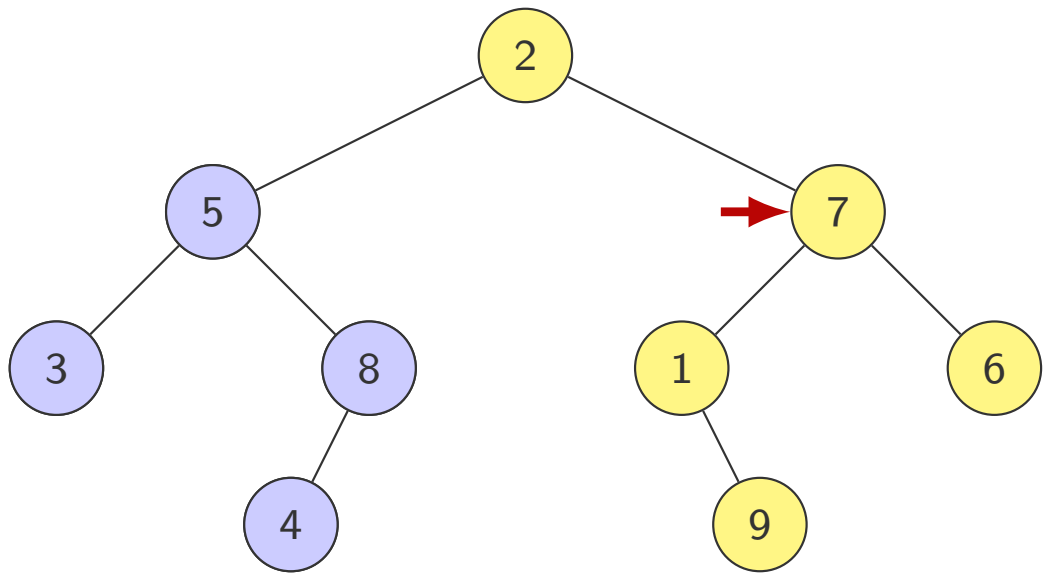


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

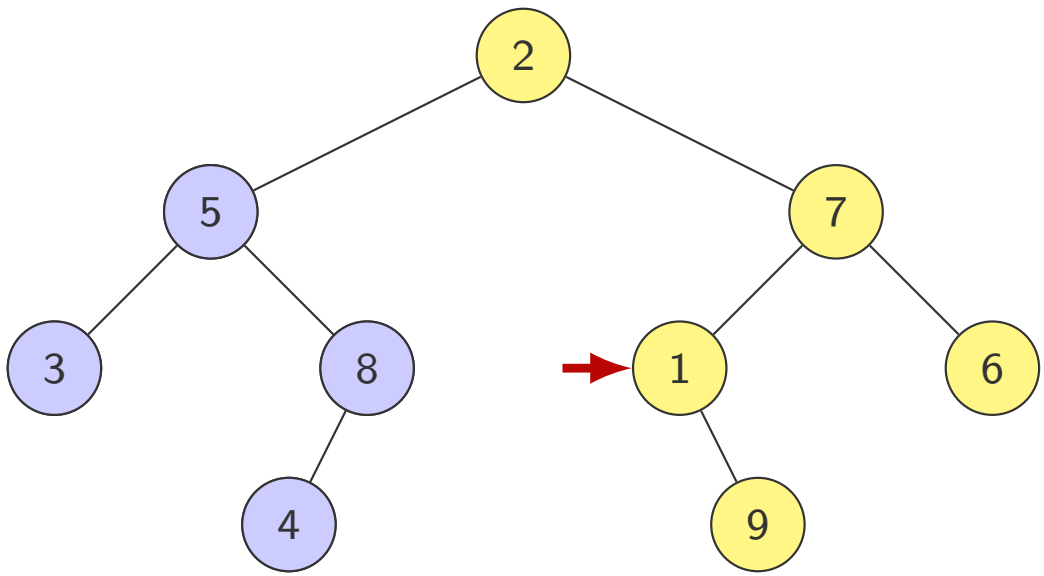


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

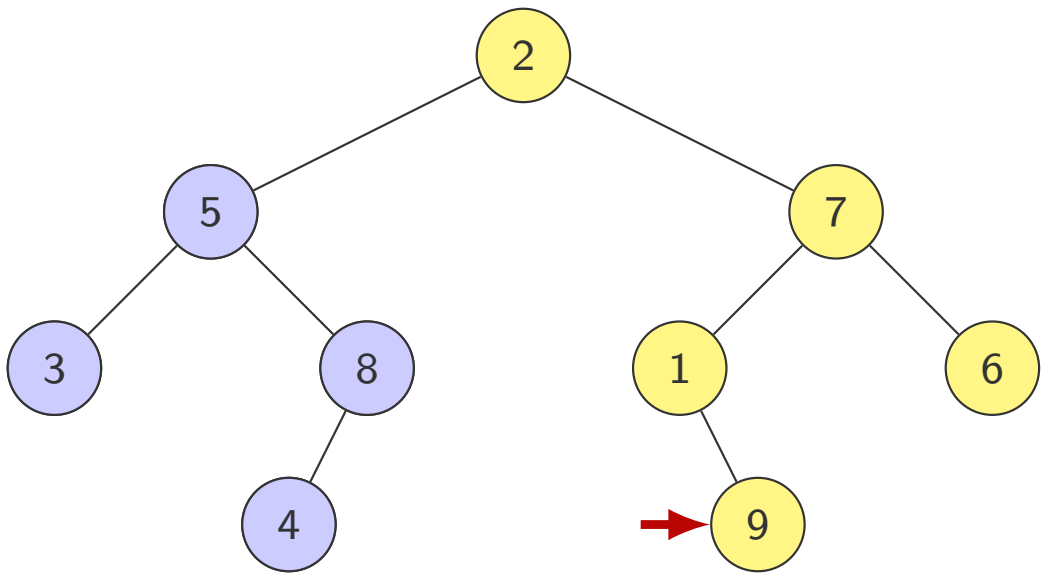


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

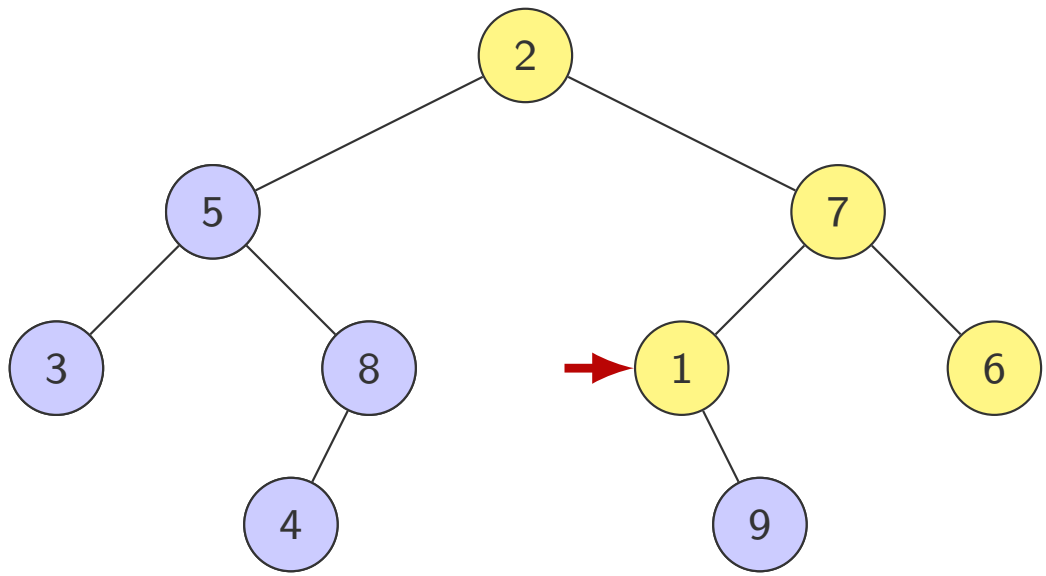


Ex: 3, 4, 8, 5,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

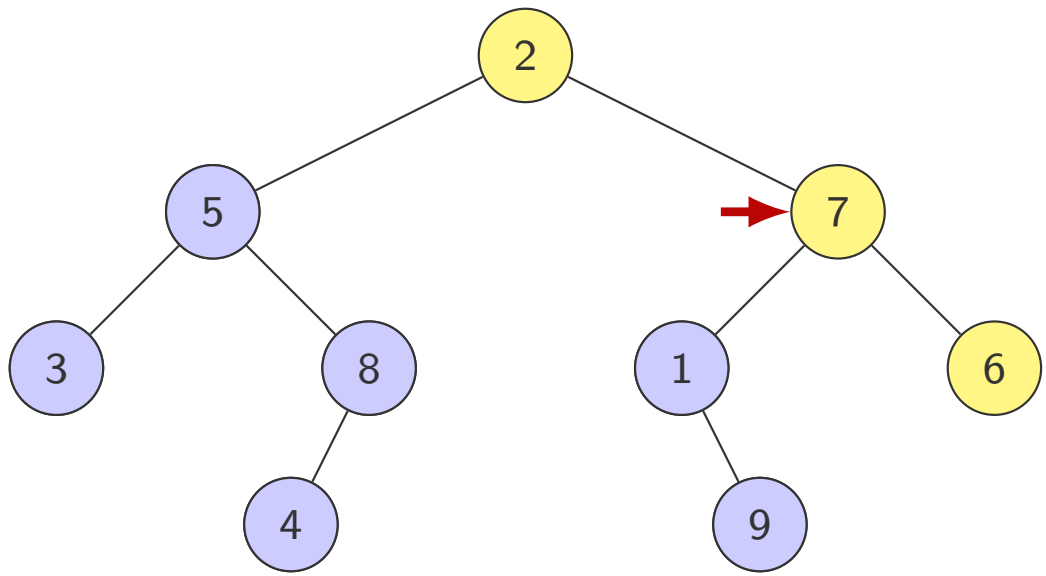


Ex: 3, 4, 8, 5, 9,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

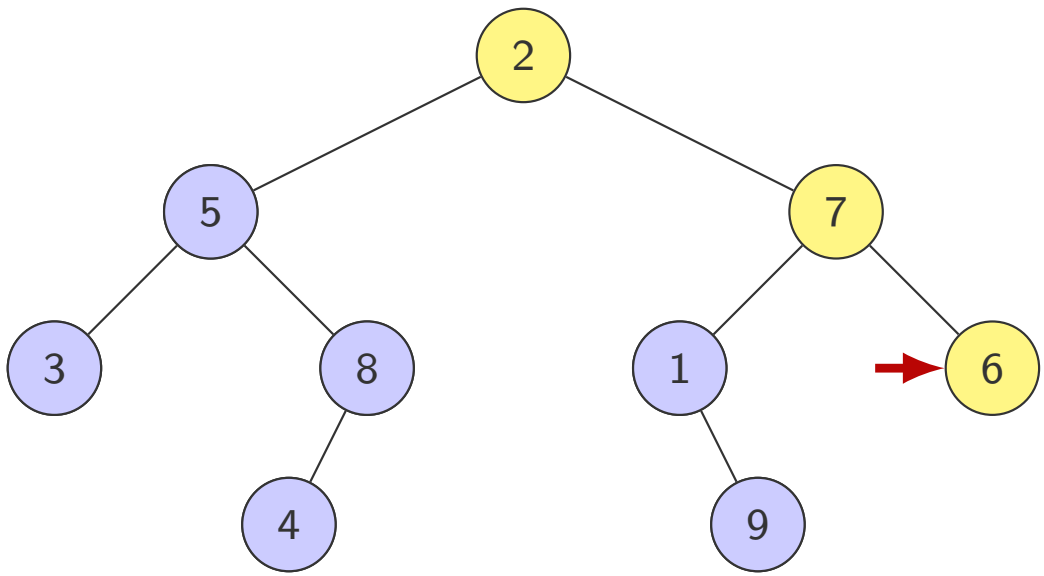


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

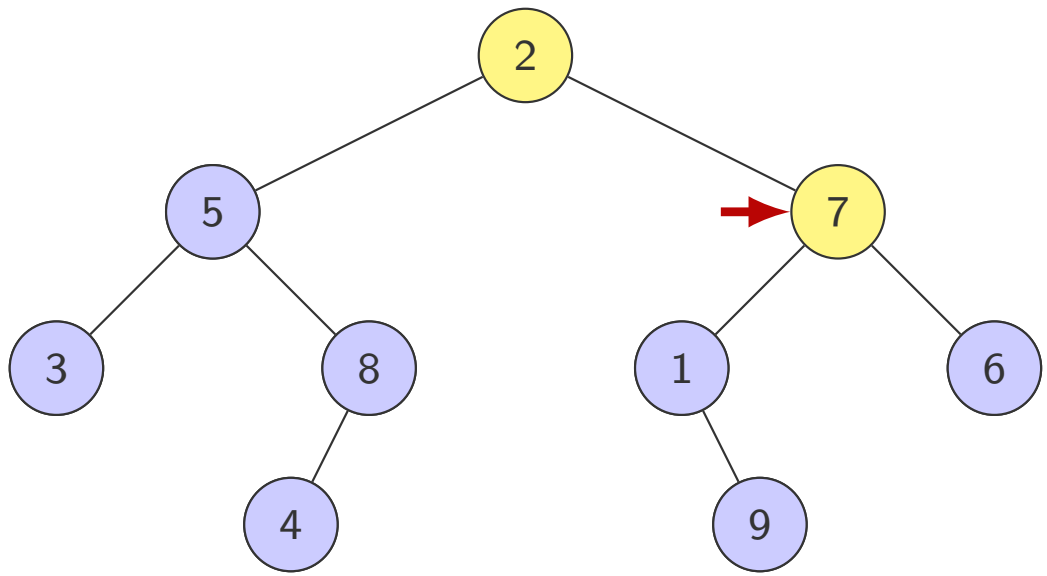


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

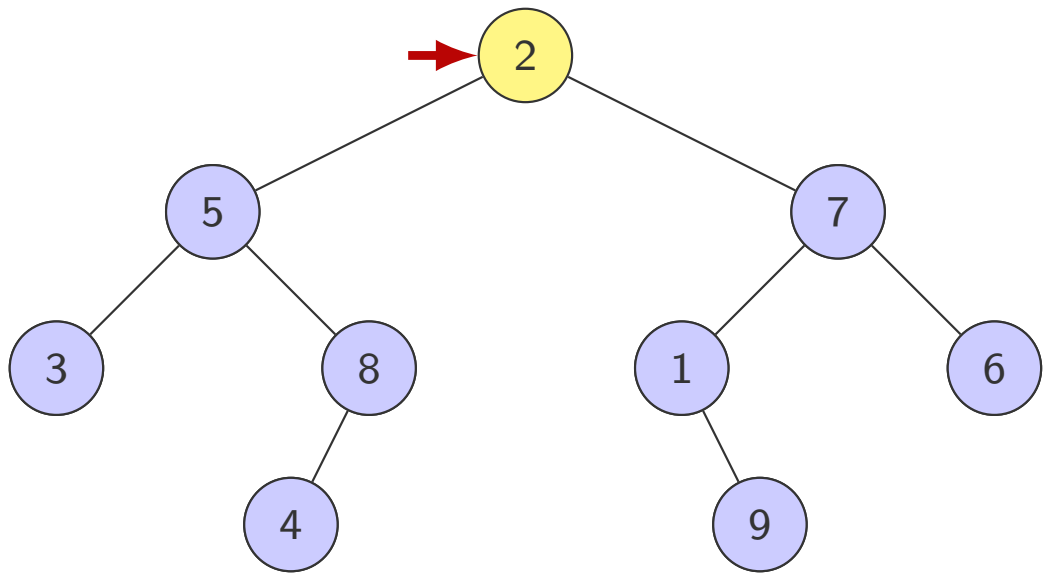


Ex: 3, 4, 8, 5, 9, 1, 6,

Percorrendo os nós — Pós-ordem

A pós-ordem

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

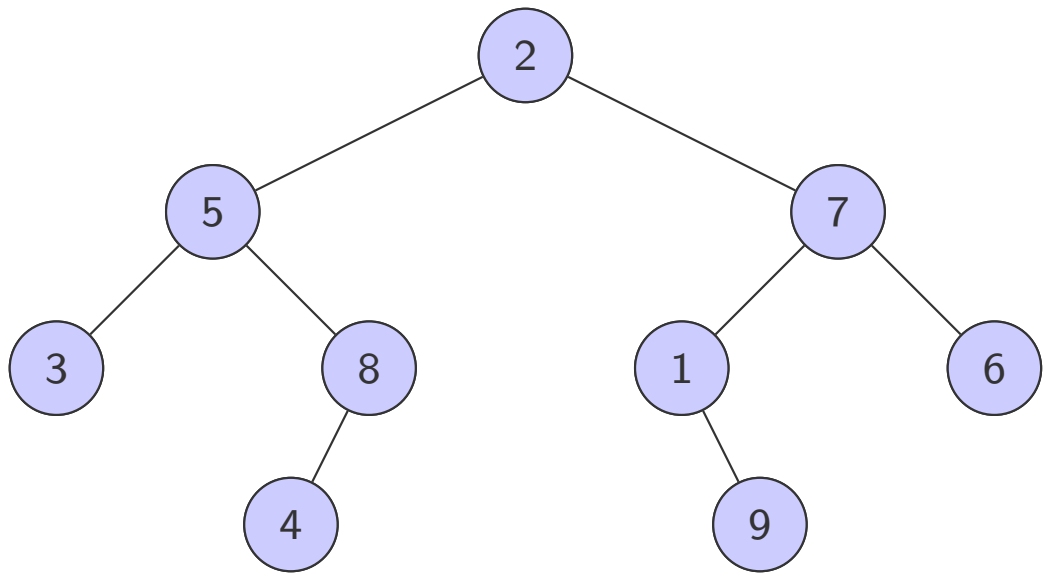


Ex: 3, 4, 8, 5, 9, 1, 6, 7,

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

Algorithm posorder(ptr)

Require: ptr (pointer to node)

```
1: if ptr ≠ NULL then
2:     posorder(ptr→left)
3:     posorder(ptr→right)
4:     visit(ptr)
5: end if
```

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

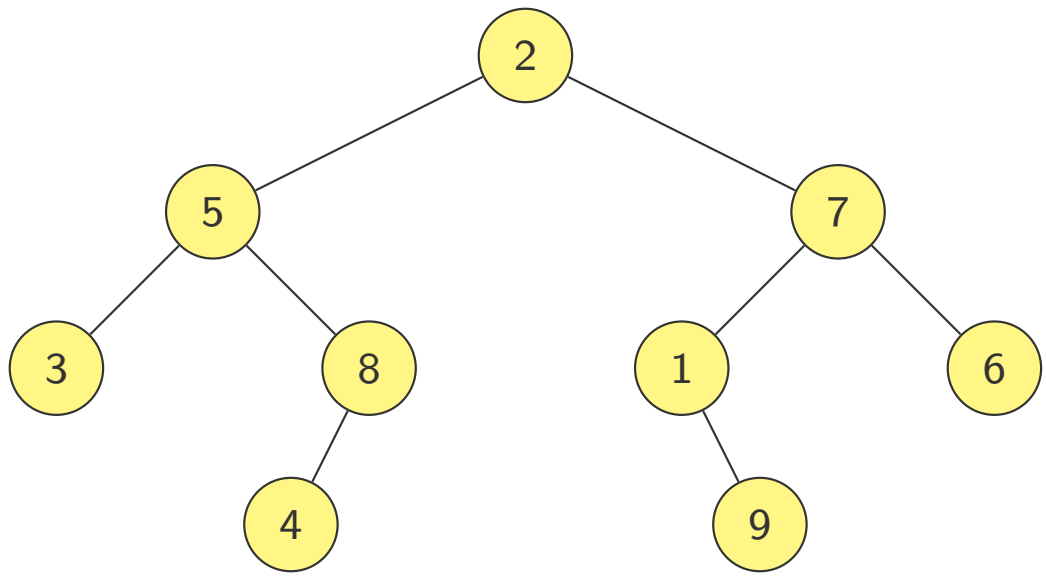
A ordem simétrica

- 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

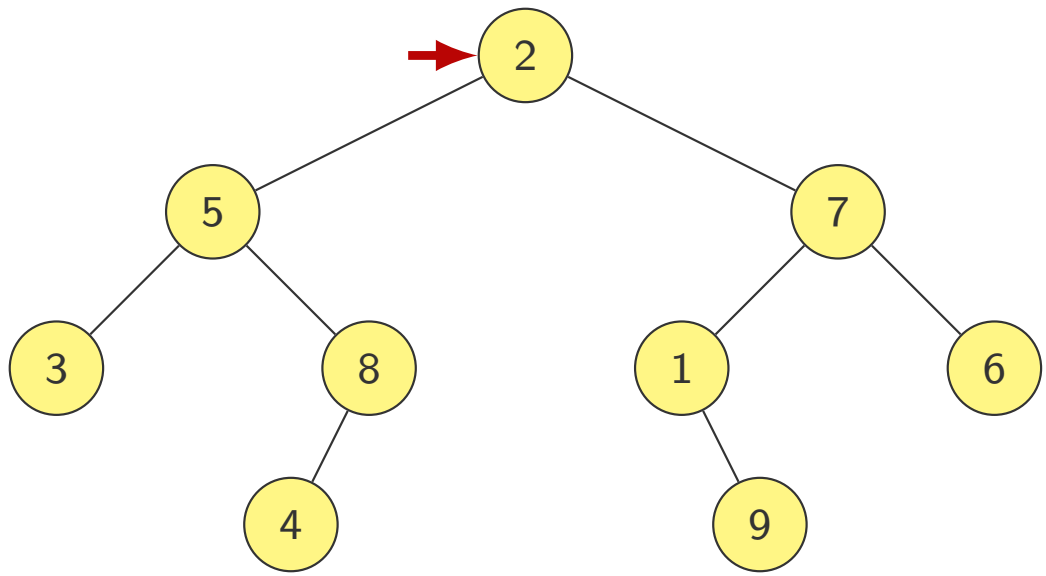


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

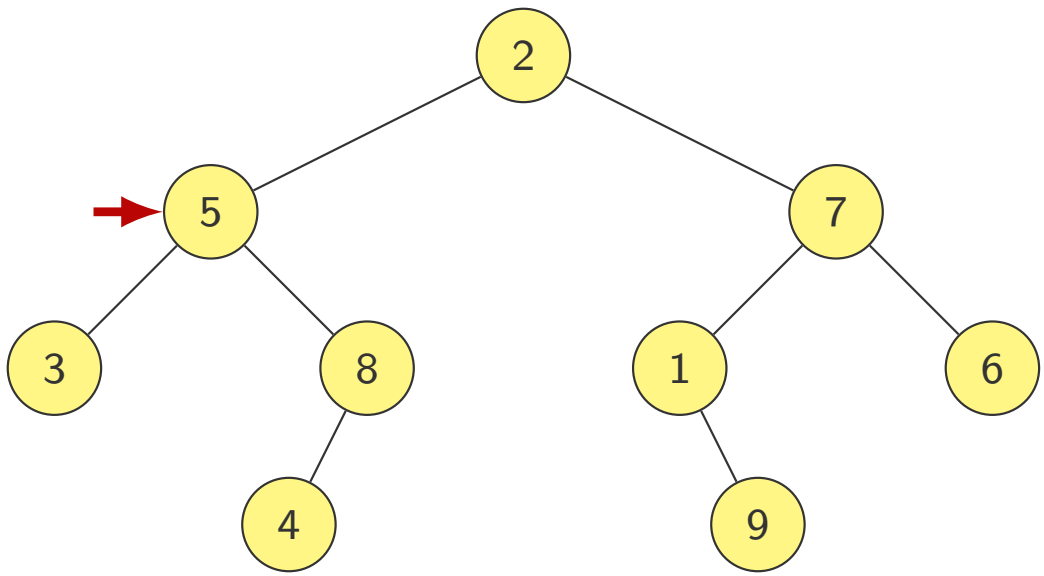


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

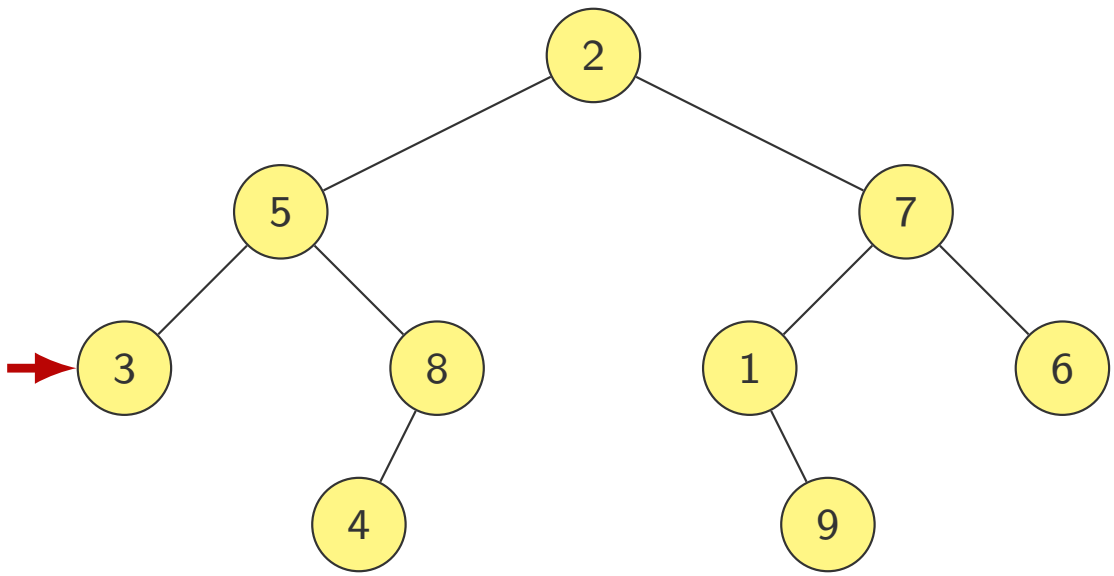


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

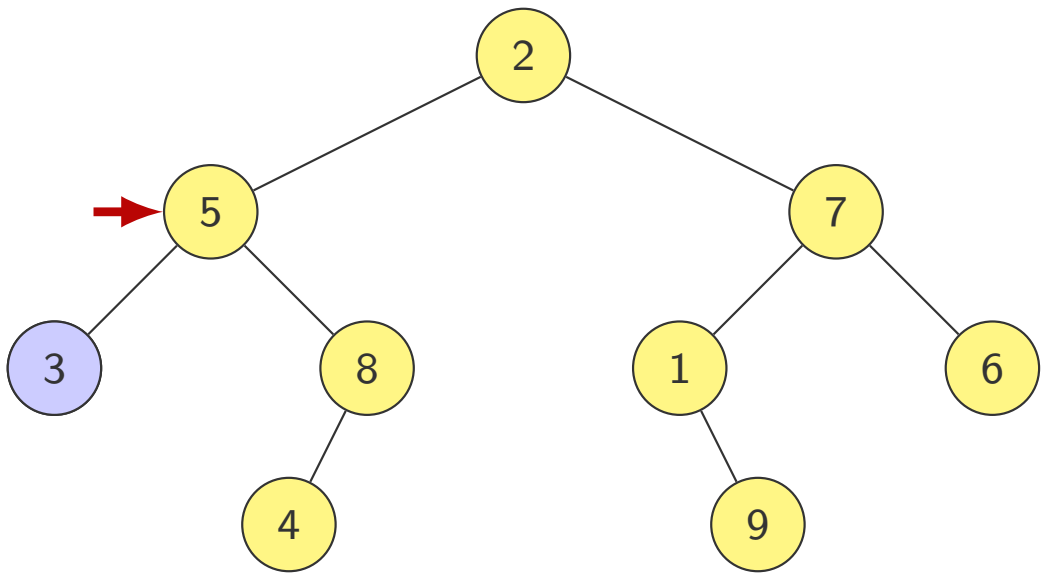


Ex:

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

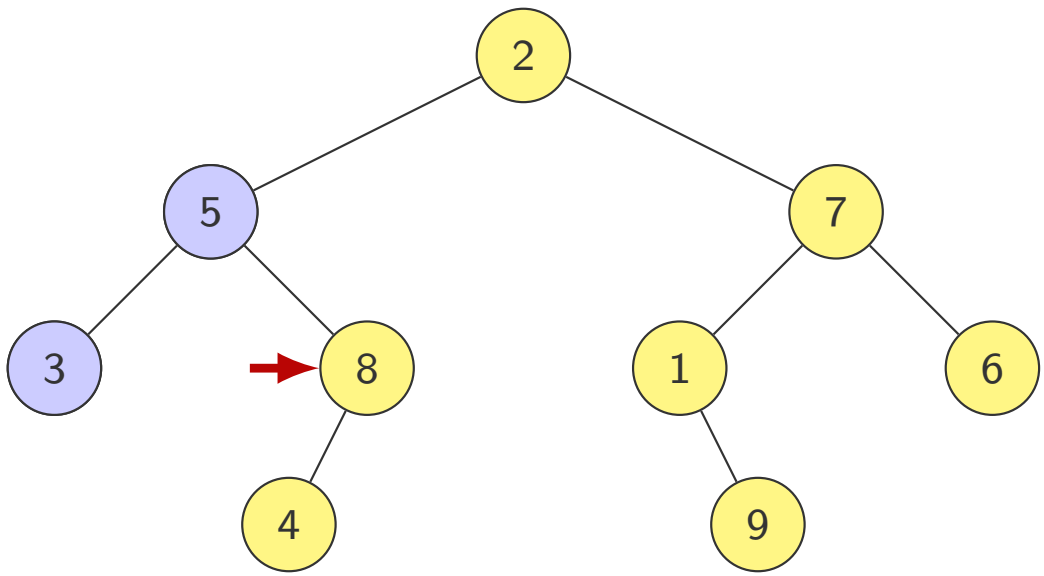


Ex: 3,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

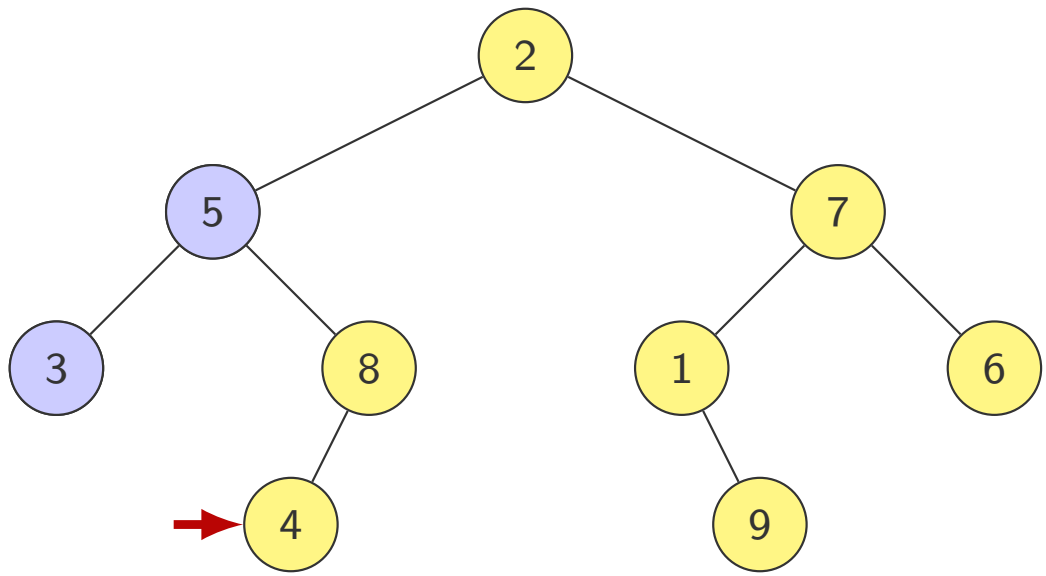


Ex: 3, 5,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

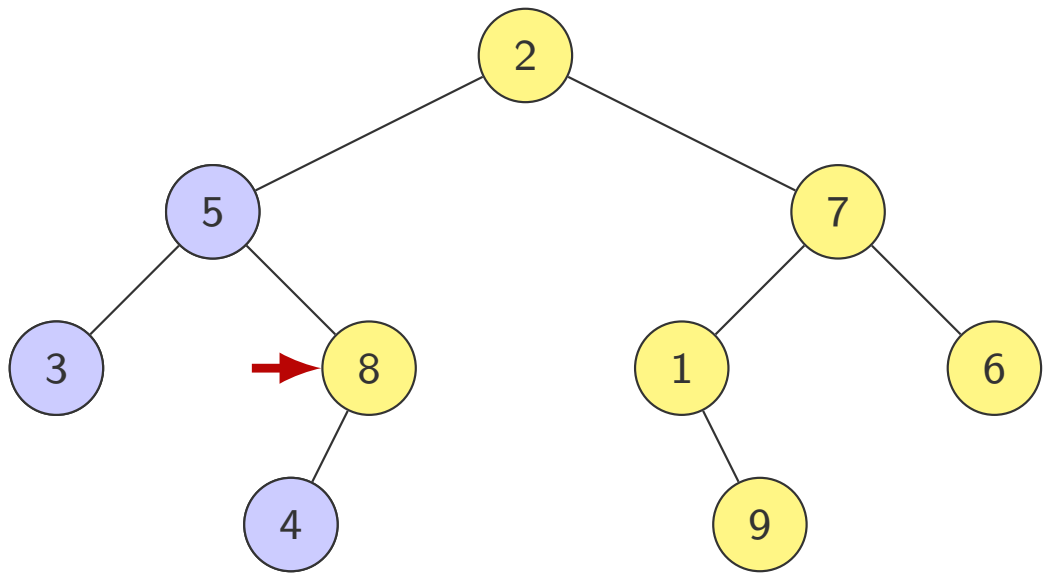


Ex: 3, 5,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

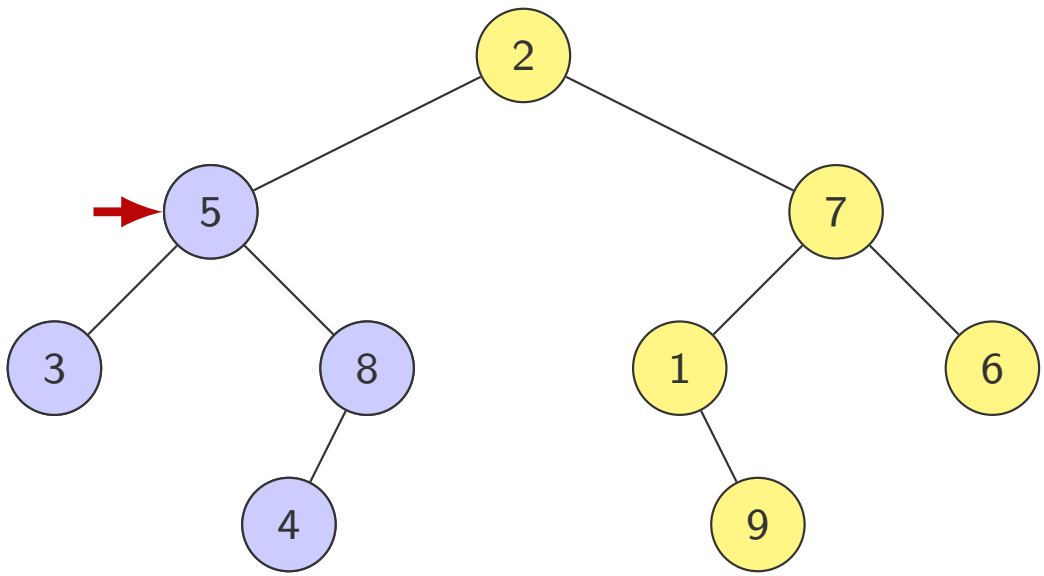


Ex: 3, 5, 4,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

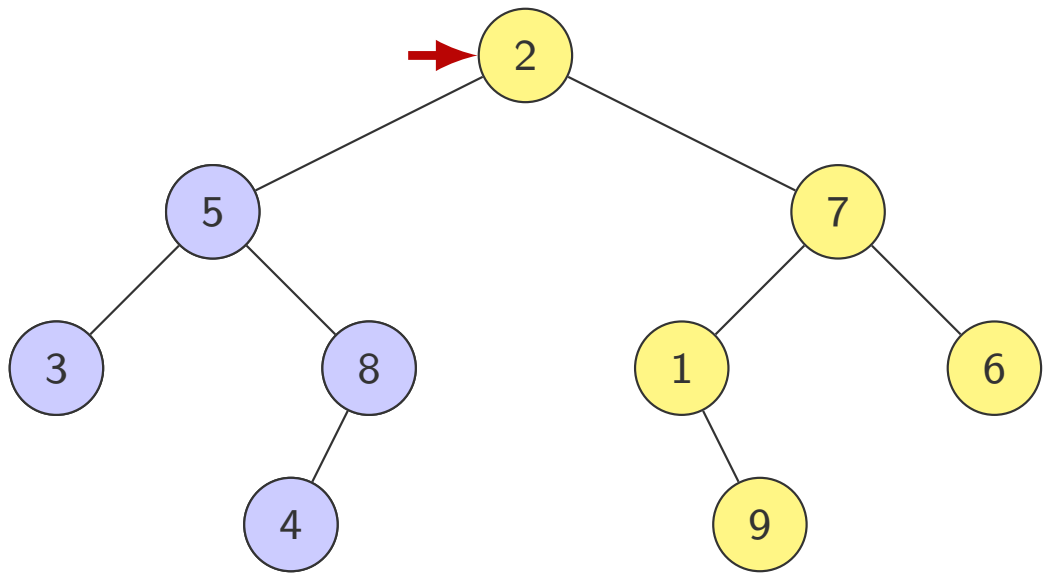


Ex: 3, 5, 4, 8,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

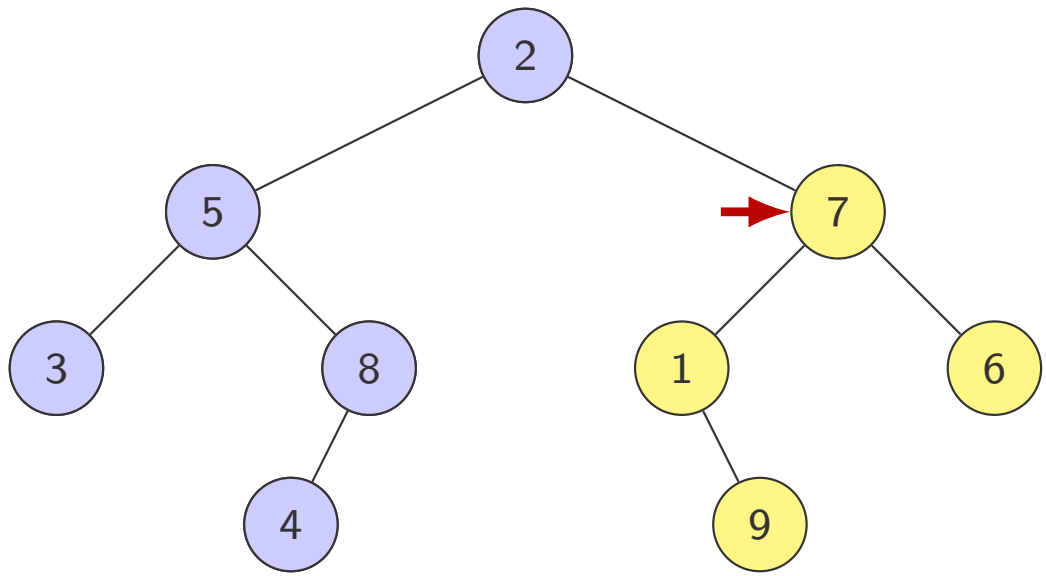


Ex: 3, 5, 4, 8,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

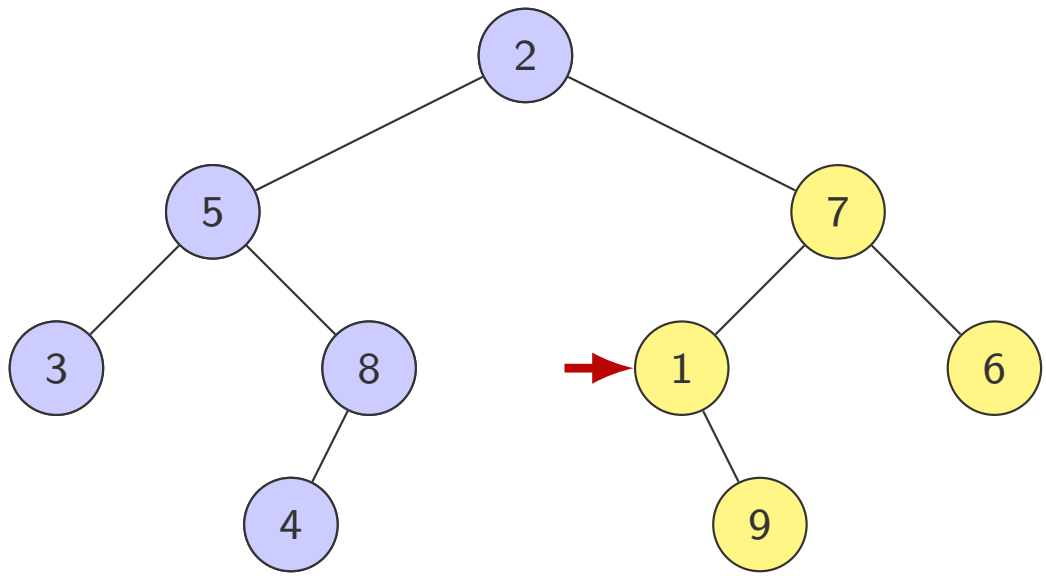


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

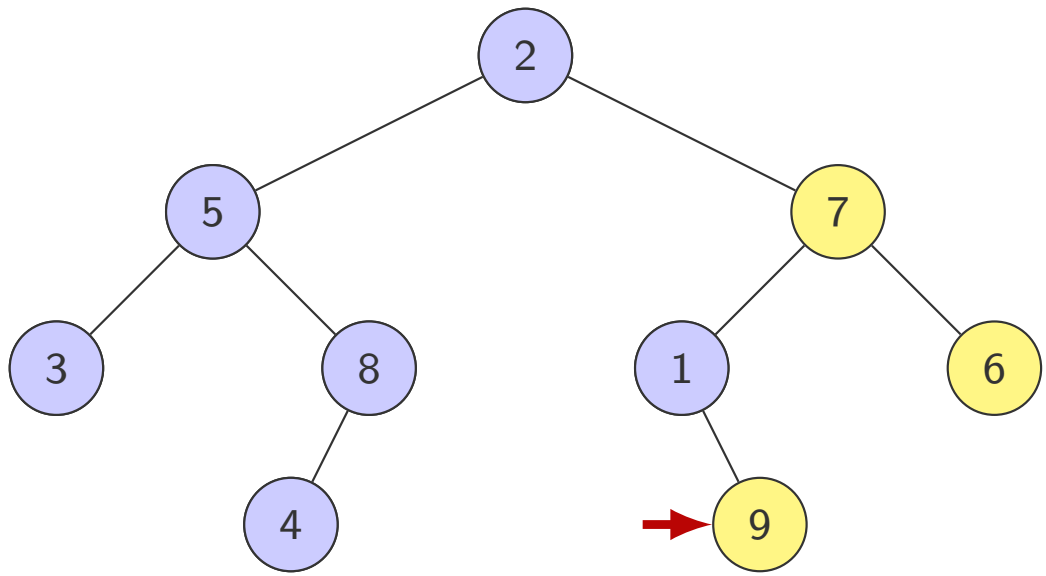


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

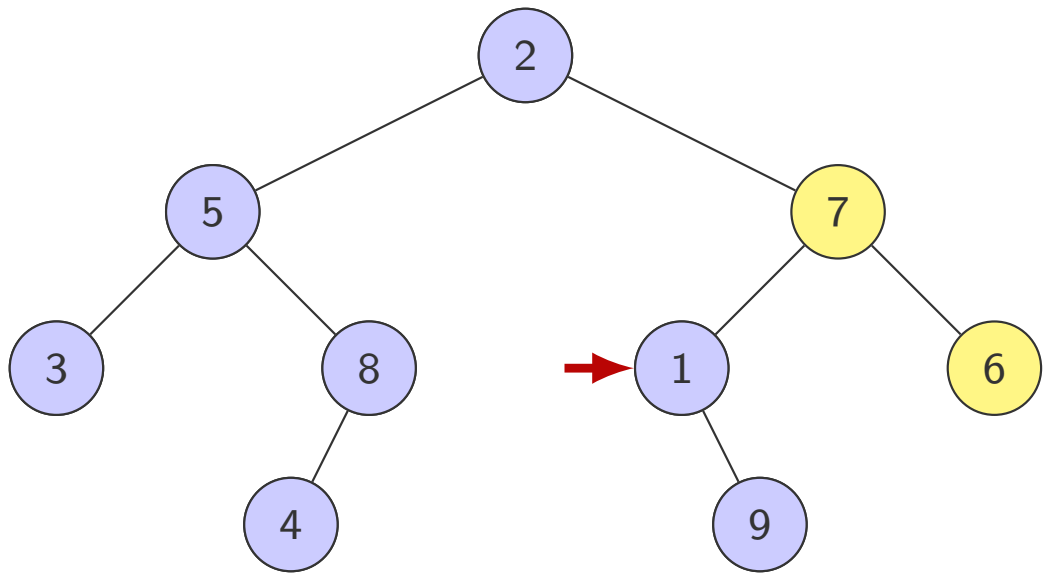


Ex: 3, 5, 4, 8, 2, 1,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

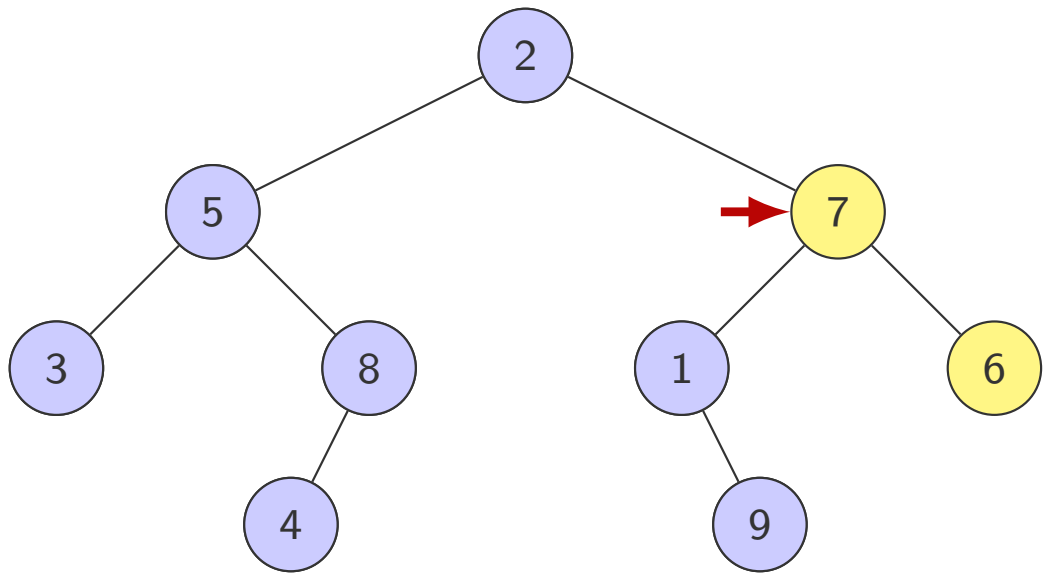


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

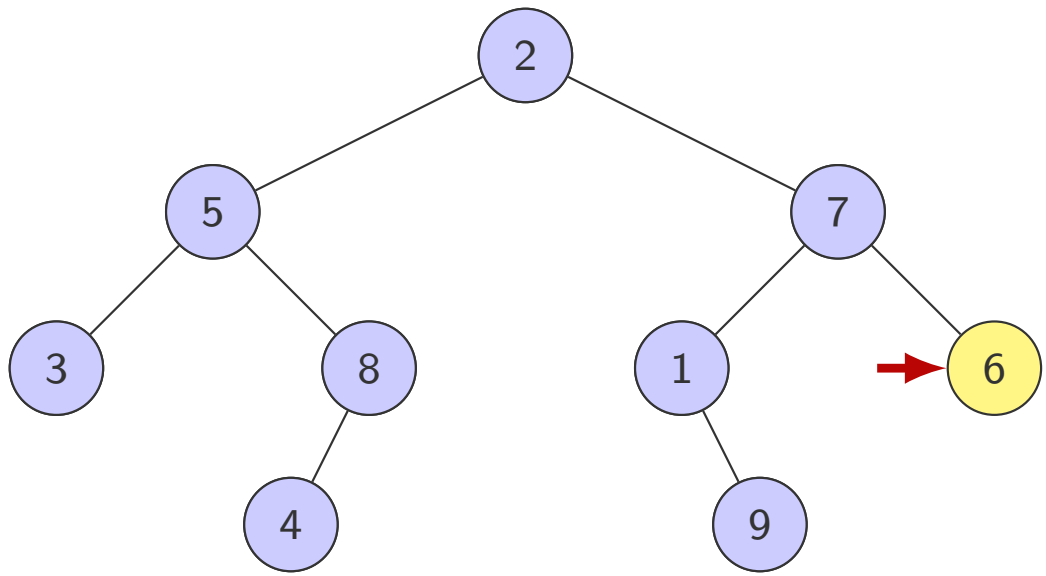


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

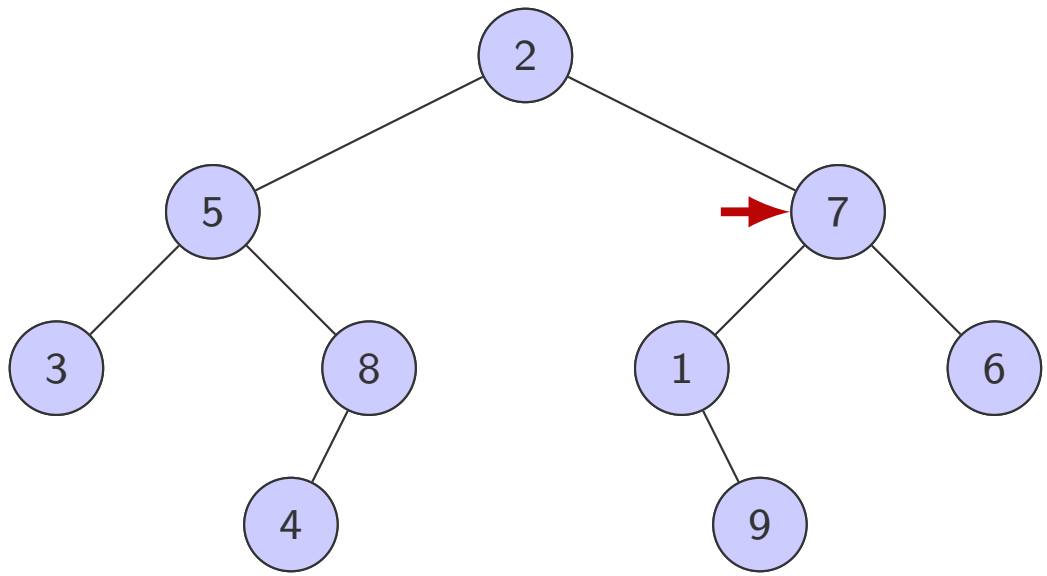


Ex: 3, 5, 4, 8, 2, 1, 9, 7,

Percorrendo os nós — Ordem Simétrica

A ordem simétrica

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

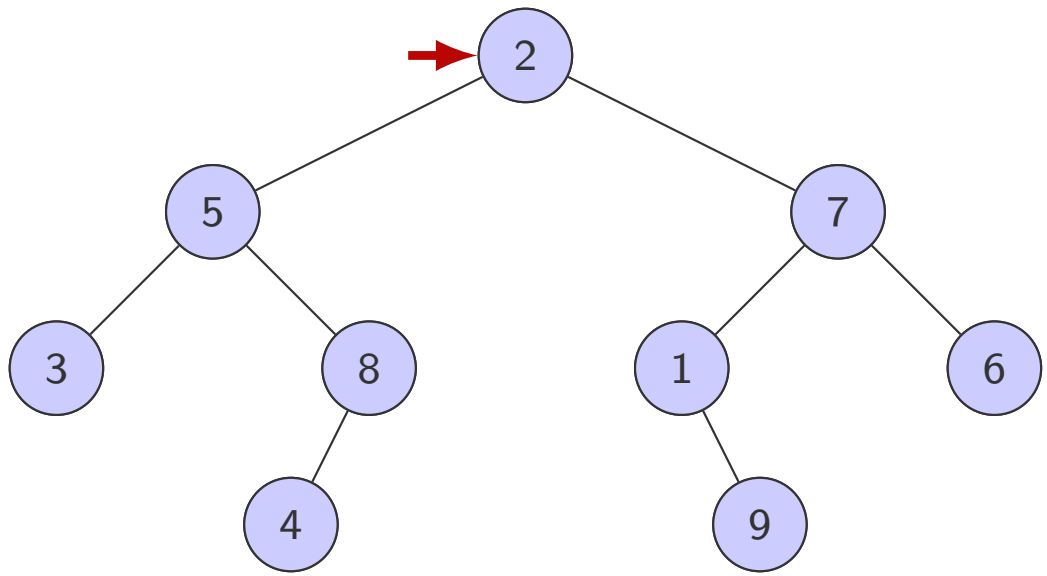


Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

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)

Algorithm inorder(ptr)

Require: ptr (pointer to node)

```
1: if ptr ≠ NULL then
2:     inorder(ptr→left)
3:     visit(ptr)
4:     inorder(ptr→right)
5: end if
```

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves
- Saber se a árvore contém certa chave

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves
- Saber se a árvore contém certa chave
- Remover todos os nós e deixar a árvore vazia

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves
- Saber se a árvore contém certa chave
- Remover todos os nós e deixar a árvore vazia
- Calcular o número de nós da árvore

Exercício

Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves
- Saber se a árvore contém certa chave
- Remover todos os nós e deixar a árvore vazia
- Calcular o número de nós da árvore
- Liberar toda memória alocada para a árvore antes dela ser destruída

Exercício

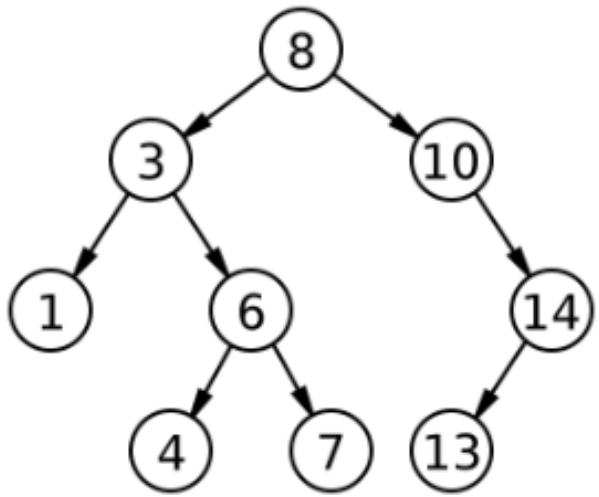
Implementar uma árvore binária em C++ como uma classe com funções-membros que implementem as seguintes funcionalidades:

- Criar árvore vazia
- Criar árvore a partir de duas outras já existentes (vazias ou não).
- Saber se árvore é vazia
- Imprimir as chaves
- Saber se a árvore contém certa chave
- Remover todos os nós e deixar a árvore vazia
- Calcular o número de nós da árvore
- Liberar toda memória alocada para a árvore antes dela ser destruída
- Lembrete: Deletar o construtor de cópia e o operador de atribuição

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 # # #
```

Percursos Iterativos em Árvore

Percurso em pré-ordem — Recursivo

Vimos que o percurso em pré-ordem recursivo é implementado pela seguinte função:

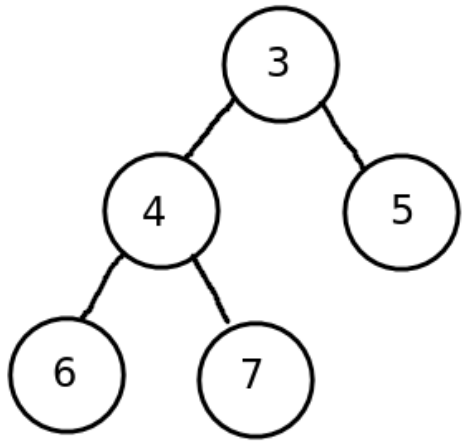
Algorithm preorder(ptr)

Require: ptr (pointer to node)

```
1: if ptr ≠ NULL then
2:     visit(ptr)
3:     preorder(ptr→left)
4:     preorder(ptr→right)
5: end if
```

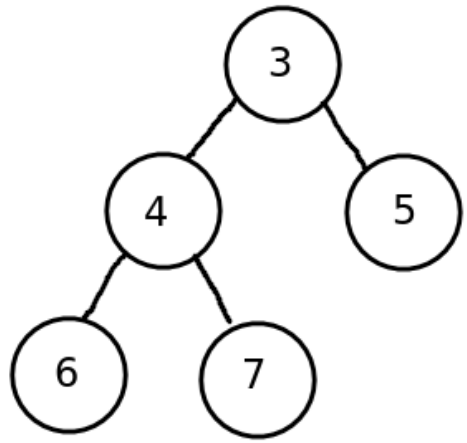
Percurso em pré-ordem — Iterativo

Como implementar a pré-ordem sem usar recursão?



Percurso em pré-ordem — Iterativo

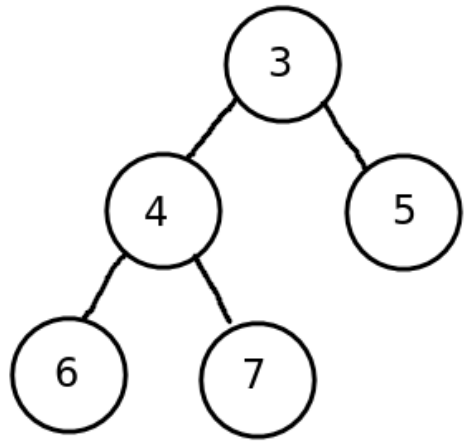
Como implementar a pré-ordem sem usar recursão?



- Na pré-ordem, ao chegarmos ao nó pela primeira vez, nós o visitamos, depois percorremos a esquerda e, só depois a direita.

Percurso em pré-ordem — Iterativo

Como implementar a pré-ordem sem usar recursão?



- Na pré-ordem, ao chegarmos ao nó pela primeira vez, nós o visitamos, depois percorremos a esquerda e, só depois a direita.
- Ao terminar de percorrer a subárvore esquerda do nó, é preciso lembrarmo-nos de que precisamos percorrer a subárvore direita dele.

Percurso em pré-ordem — Iterativo

Algorithm preorderIterativo(Node *root)

Require: root (ponteiro para a raiz)

```
 1: Cria uma pilha vazia P de ponteiros para nós
 2: if node ≠ NULL then
 3:     P.push(root)
 4: end if
 5: while P ≠ ∅ do
 6:     node = P.top()
 7:     P.pop()
 8:     visit(node)
 9:     if node→right ≠ NULL then
10:         P.push(node→right)
11:     end if
12:     if node→left ≠ NULL then
13:         P.push(node→left)
14:     end if
15: end while
```

Percurso em pré-ordem — Iterativo

Algorithm preorderIterativo(Node *root)

Require: root (ponteiro para a raiz)

```
 1: Cria uma pilha vazia P de ponteiros para nós
 2: if node ≠ NULL then
 3:     P.push(root)
 4: end if
 5: while P ≠ ∅ do
 6:     node = P.top()
 7:     P.pop()
 8:     visit(node)
 9:     if node→right ≠ NULL then
10:         P.push(node→right)
11:     end if
12:     if node→left ≠ NULL then
13:         P.push(node→left)
14:     end if
15: end while
```

Por que empilhamos `node->right` primeiro? E se fosse o contrário?

Percurso em ordem simétrica — Recursivo

O percurso em ordem simétrica (inordem) recursivo é implementado pela seguinte função:

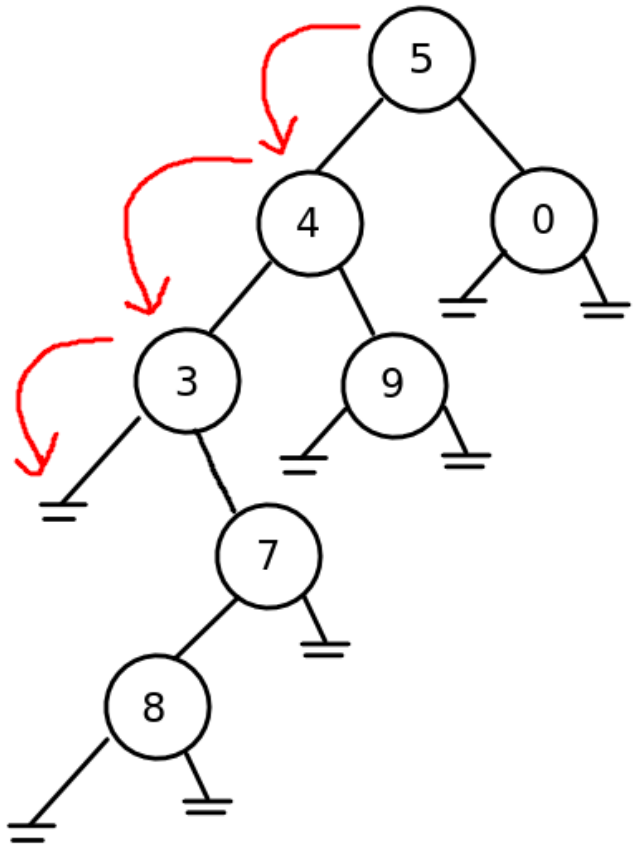
Algorithm inorder(ptr)

Require: ptr (pointer to node)

```
1: if ptr ≠ NULL then
2:     inorder(ptr→left)
3:     visit(ptr)
4:     inorder(ptr→right)
5: end if
```

Percurso em ordem simétrica — Dificuldade

Como percorrer em ordem simétrica sem usar recursão?



Percurso em ordem simétrica — Iterativo

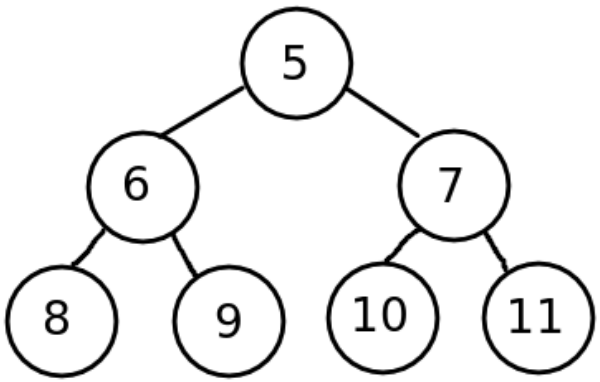
Percurso em ordem simétrica — Iterativo

Algorithm inorderIterativo()

Require: root (ponteiro para a raiz)

```
1:  Cria uma pilha vazia P de ponteiros para nós
2:  node = root
3:  while P ≠ ∅ or node ≠ NULL do
4:      if node ≠ NULL then
5:          P.push(node)
6:          node = node→left
7:      else
8:          node = P.top()
9:          P.pop()
10:         visit(node)
11:         node = node→right
12:     end if
13: end while
```

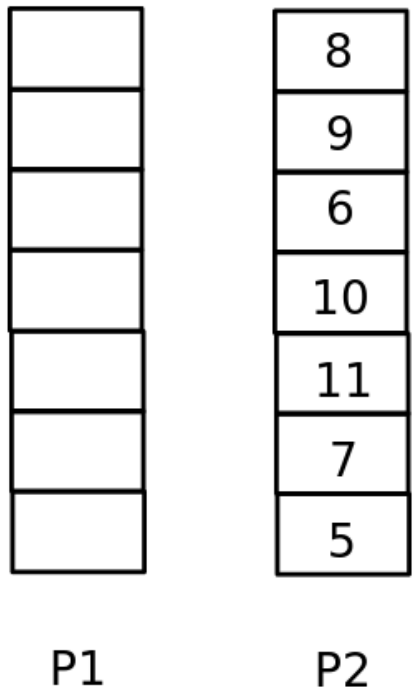
Percurso em pós-ordem iterativo



pos-ordem

8 9 6 10 11 7 5

pré-ordem reversa



Percurso em pós-ordem — Iterativo

Algorithm Pos-Ordem-Iterativo(Node *root)

```
1:  Cria pilhas vazia P1 e P2 de ponteiros para nós
2:  if root != NULL then P1.push(root)
3:  while P1 ≠ ∅ do
4:      node = P1.top()
5:      P1.pop()
6:      P2.push(node)
7:      if node→left != NULL then P1.push(node→left)
8:      if node→right != NULL then P1.push(node→right)
9:  end while
10: while P2 ≠ ∅ do
11:     node = P2.top()
12:     P2.pop()
13:     visit(node)
14: end while
```

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)

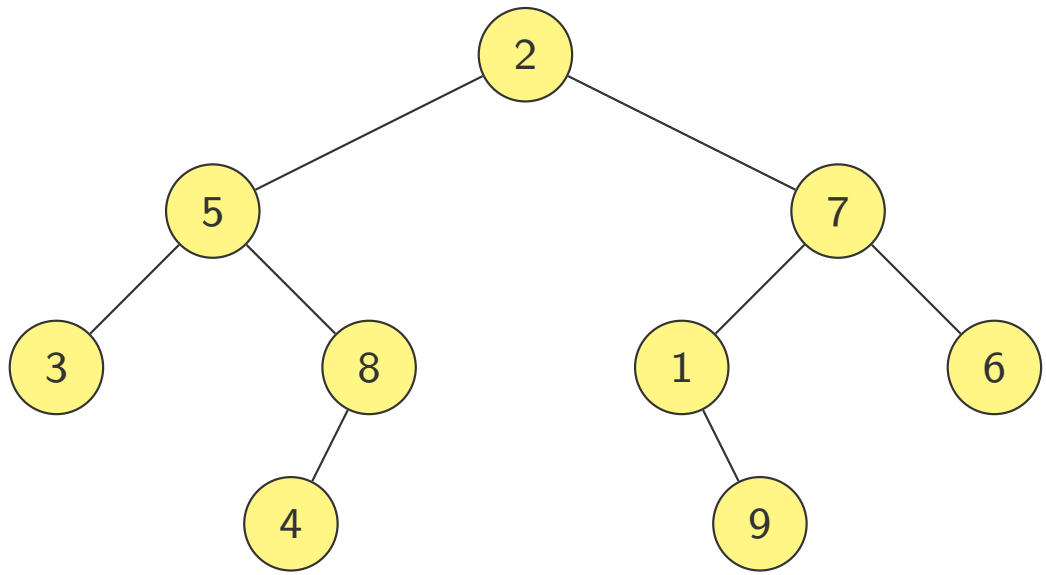
O percurso em largura

- 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

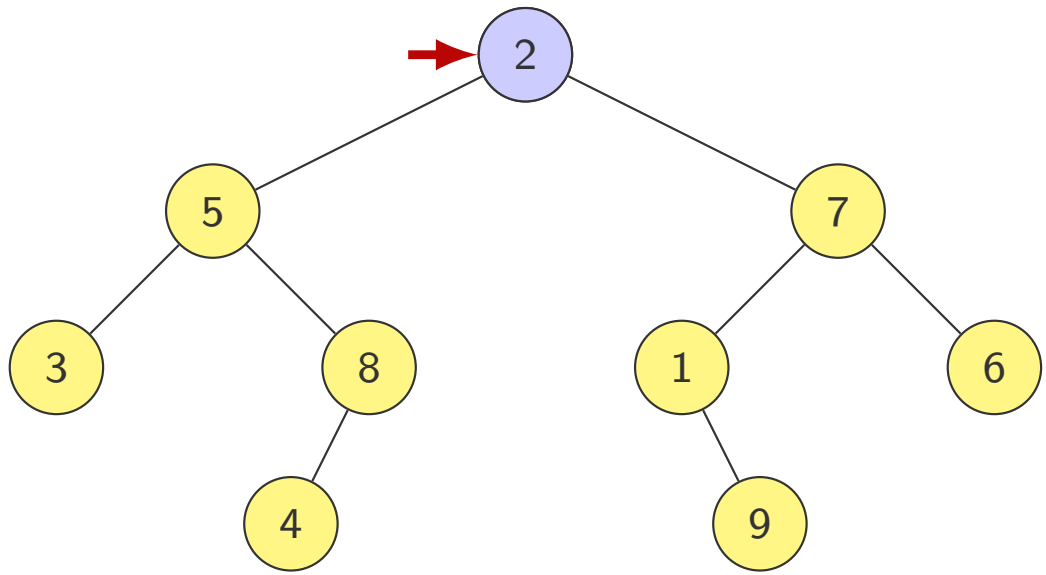


Ex:

Percorrendo os nós (em largura)

O percurso em largura

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

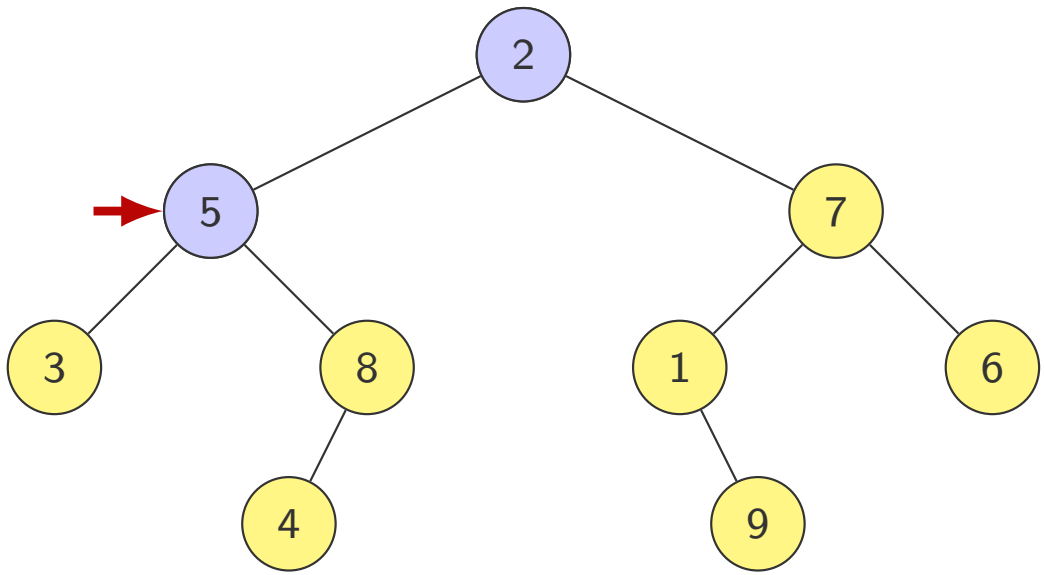


Ex: 2,

Percorrendo os nós (em largura)

O percurso em largura

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

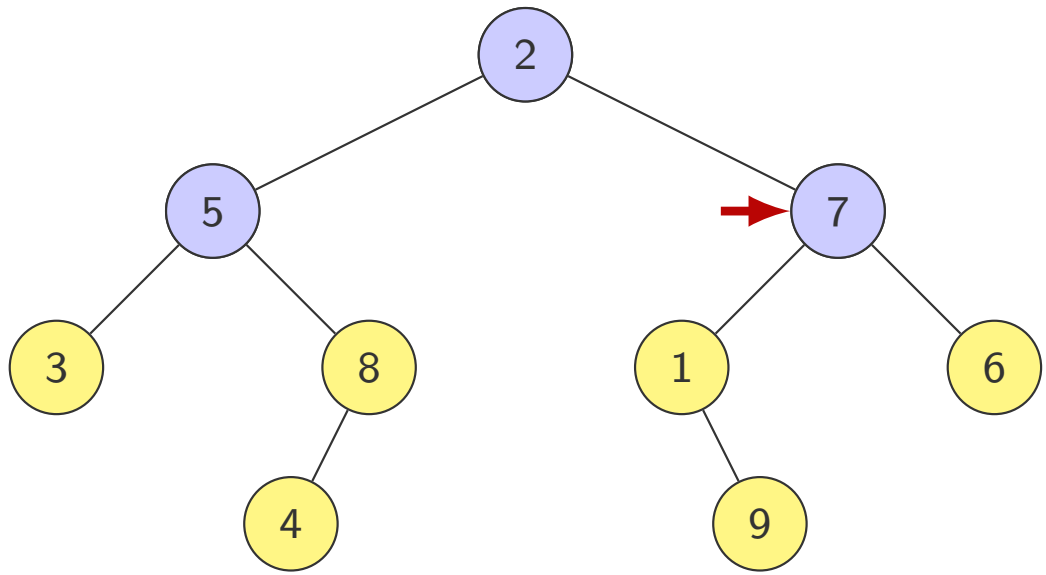


Ex: 2, 5,

Percorrendo os nós (em largura)

O percurso em largura

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

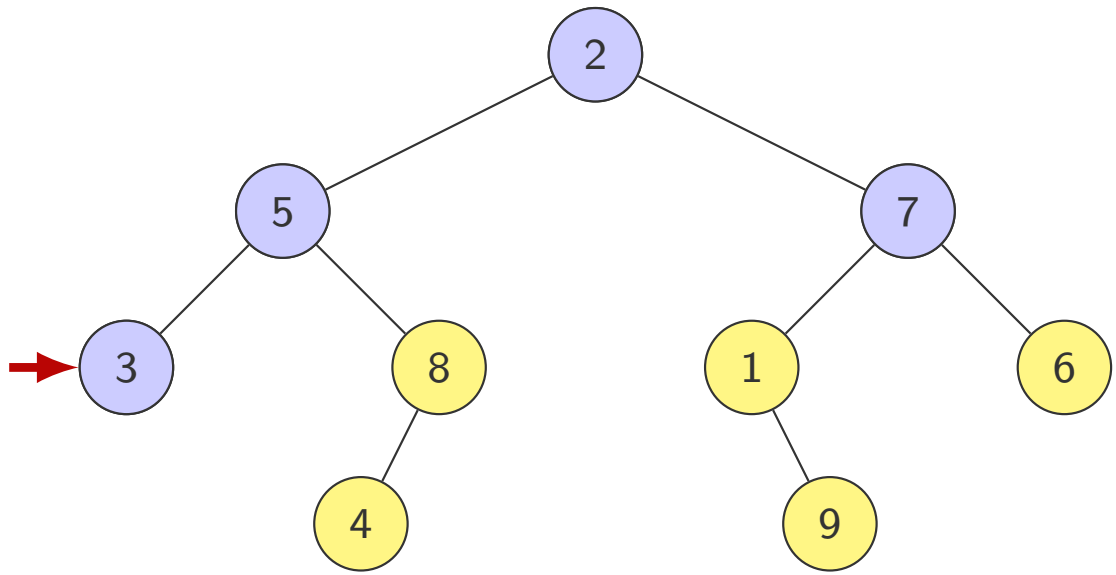


Ex: 2, 5, 7,

Percorrendo os nós (em largura)

O percurso em largura

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

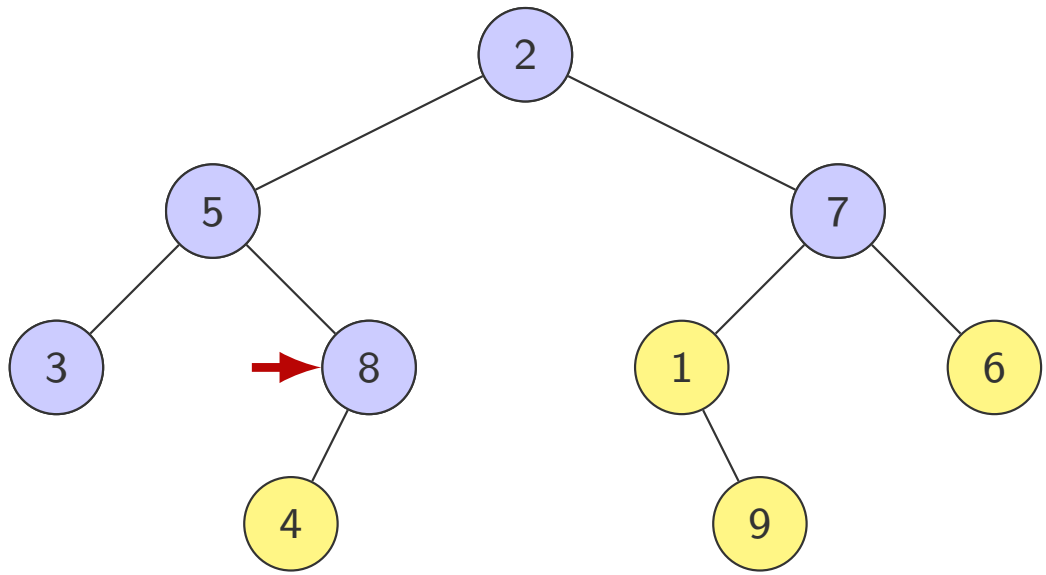


Ex: 2, 5, 7, 3,

Percorrendo os nós (em largura)

O percurso em largura

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

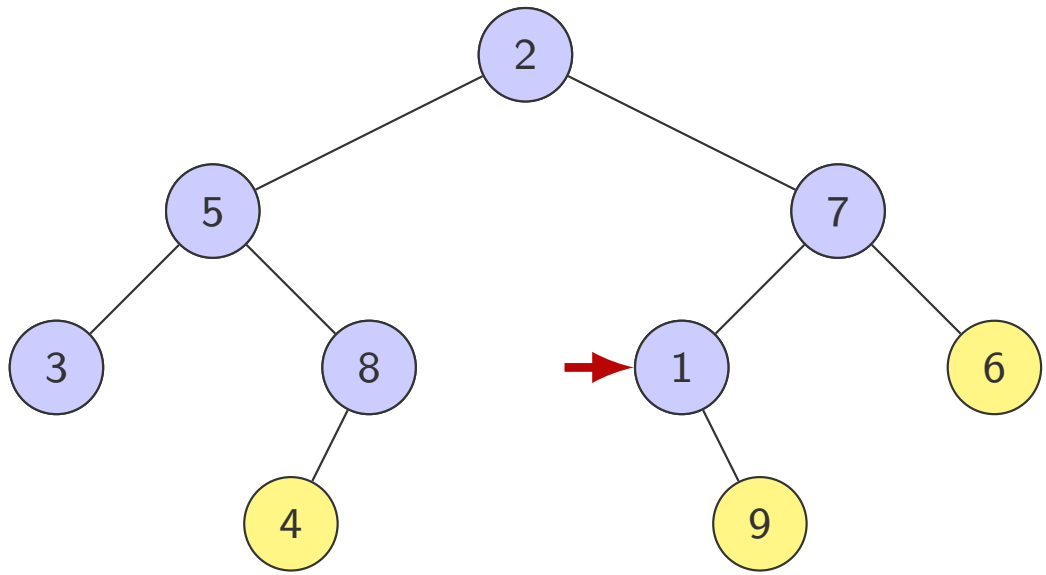


Ex: 2, 5, 7, 3, 8,

Percorrendo os nós (em largura)

O percurso em largura

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

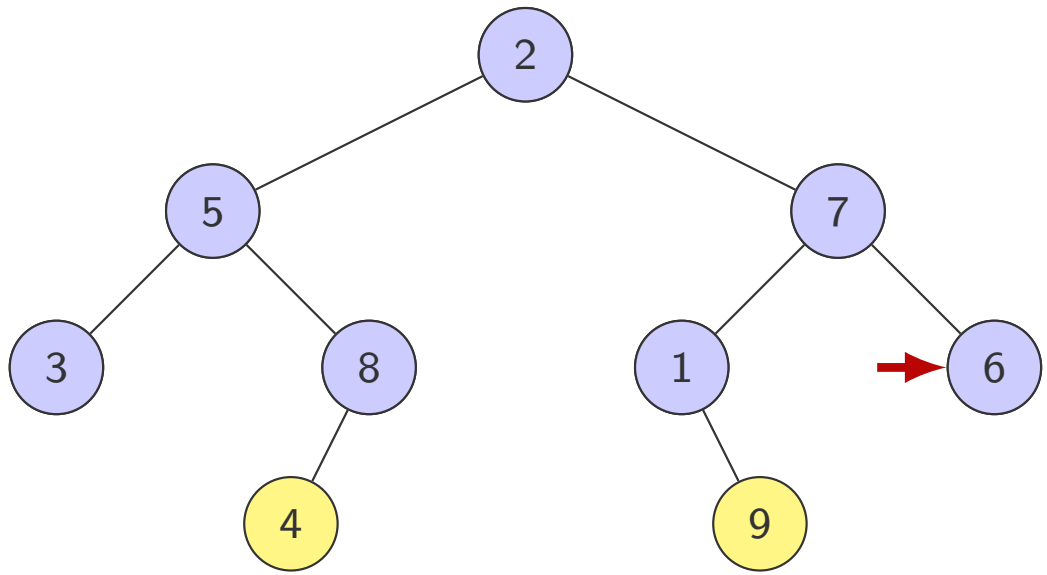


Ex: 2, 5, 7, 3, 8, 1,

Percorrendo os nós (em largura)

O percurso em largura

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

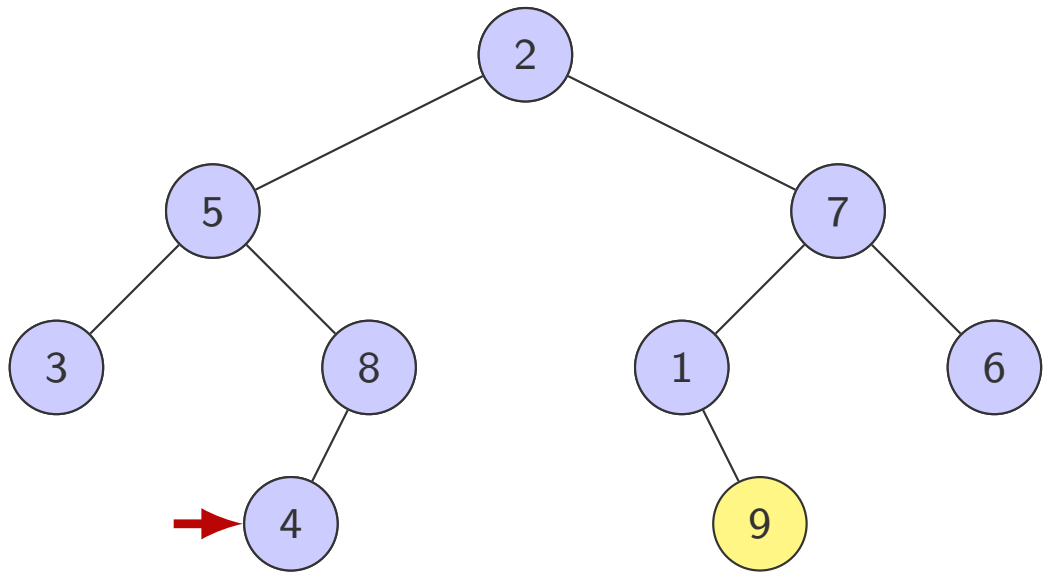


Ex: 2, 5, 7, 3, 8, 1, 6,

Percorrendo os nós (em largura)

O percurso em largura

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

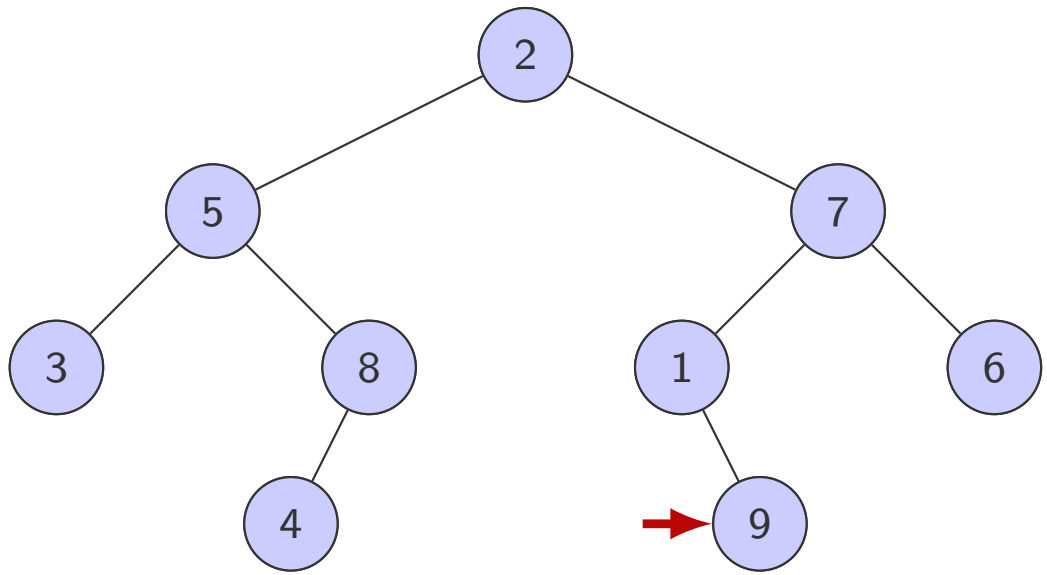


Ex: 2, 5, 7, 3, 8, 1, 6, 4,

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

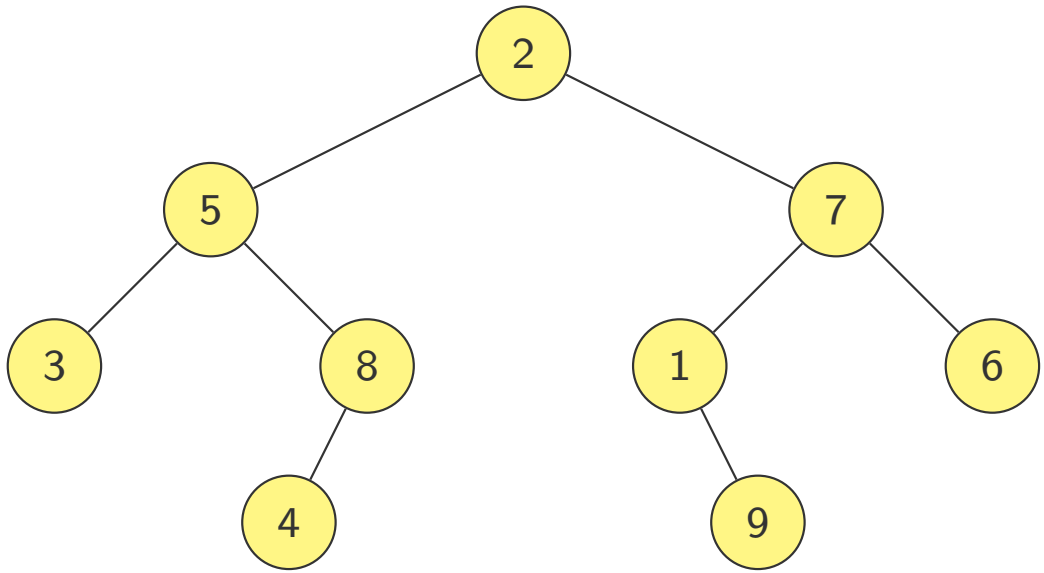
Como implementar a busca em largura?

- 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

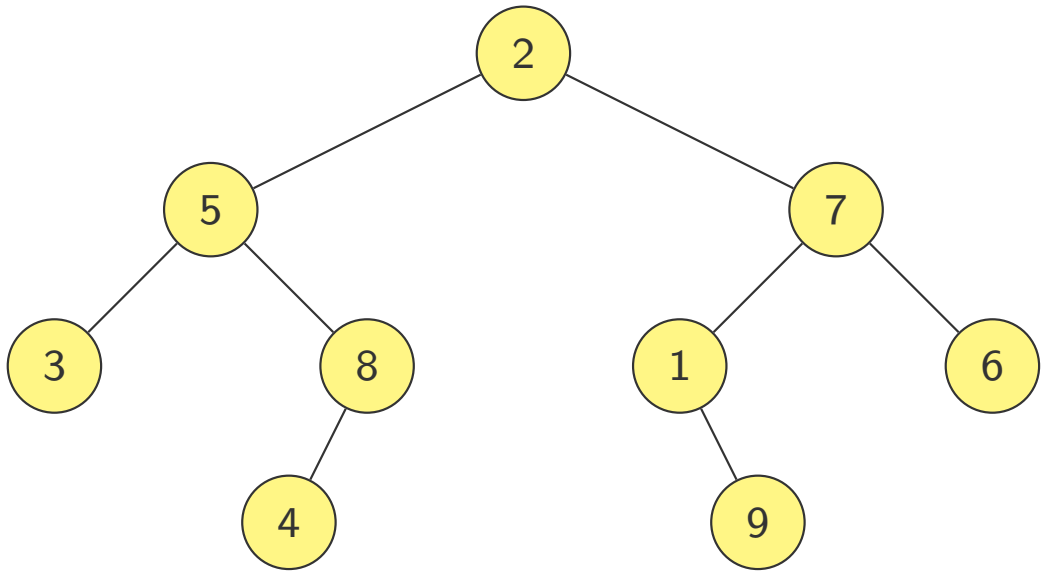


Fila

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

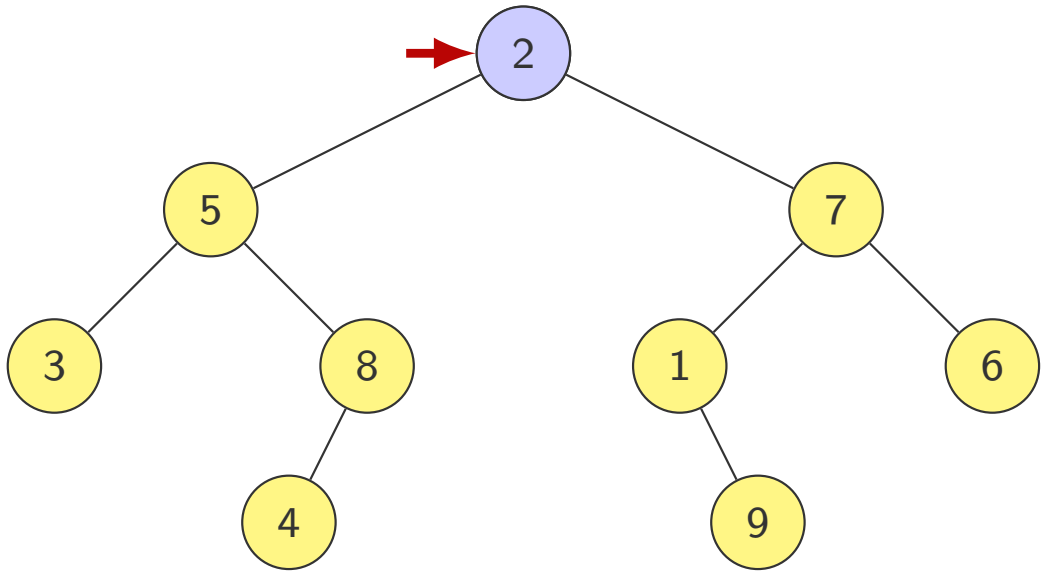


Fila 

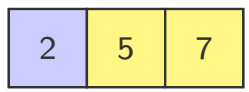
Implementação do percurso em largura

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



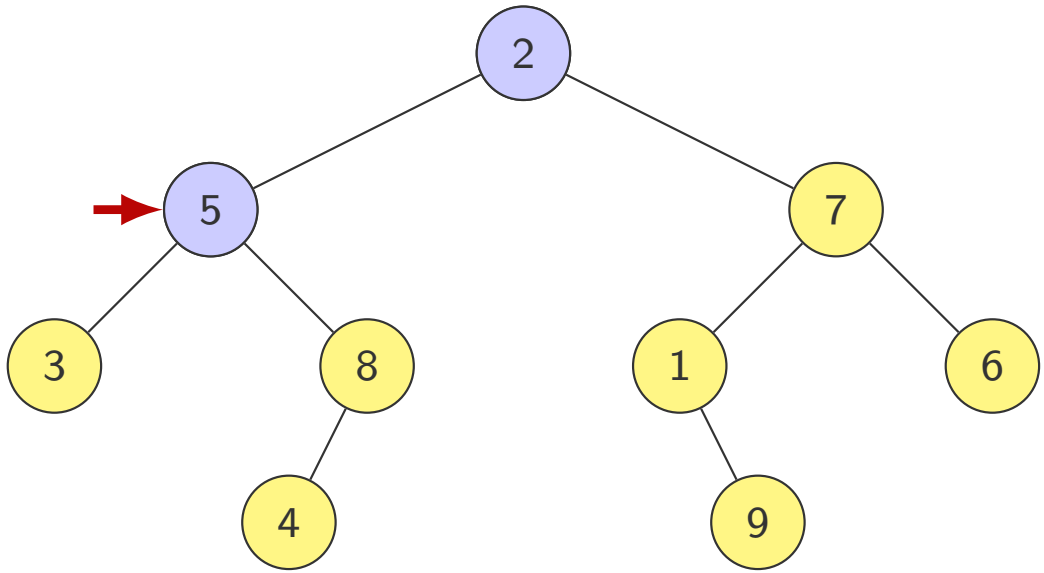
Fila



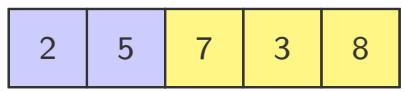
Implementação do percurso em largura

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



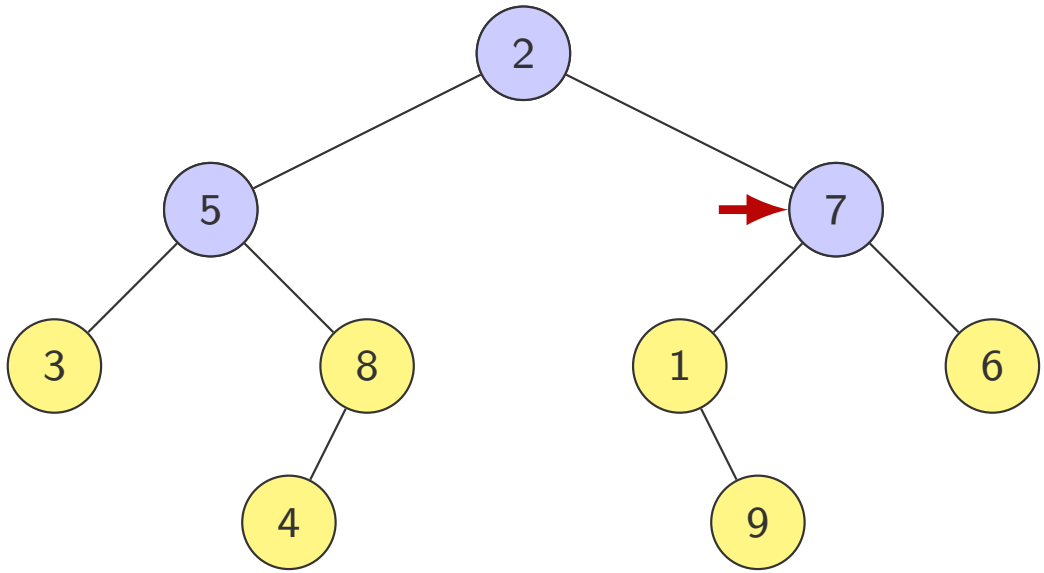
Fila



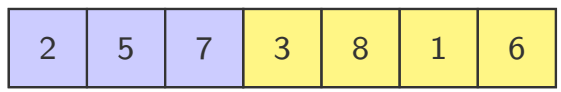
Implementação do percurso em largura

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



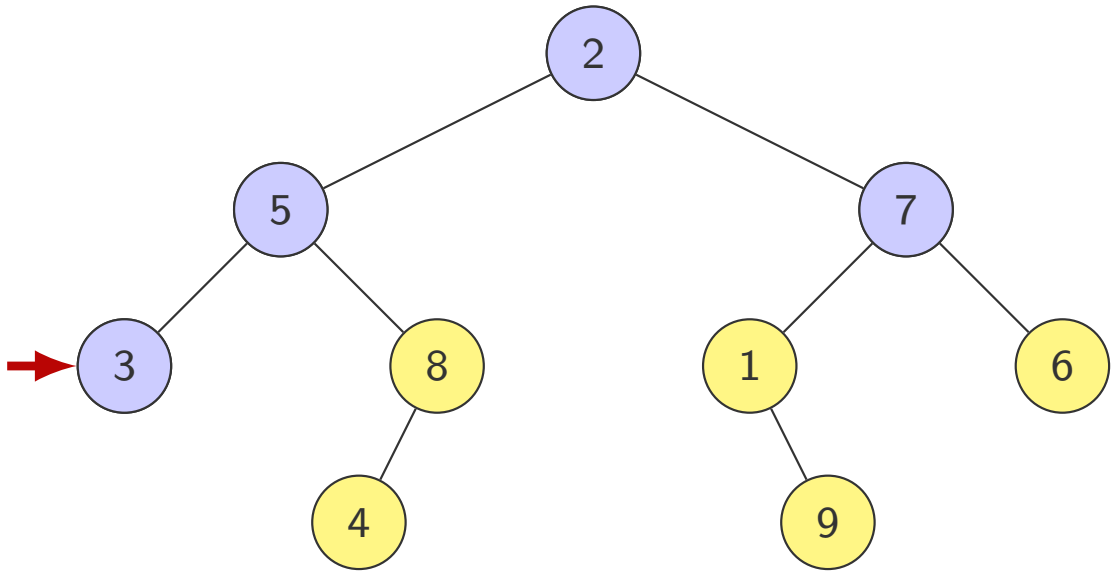
Fila



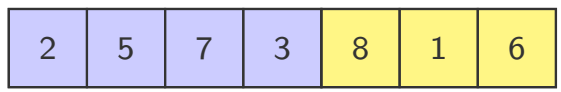
Implementação do percurso em largura

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



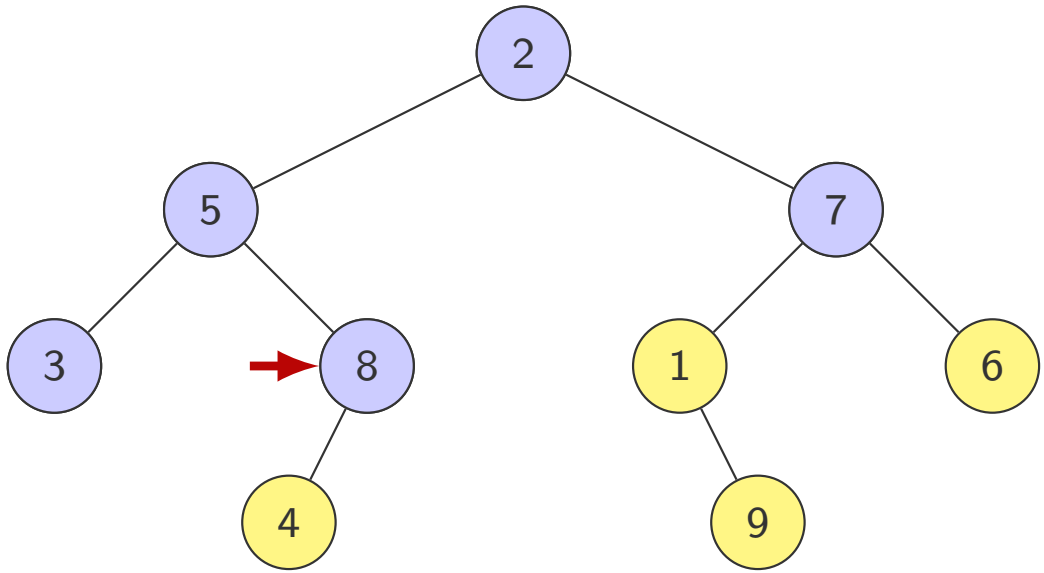
Fila



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

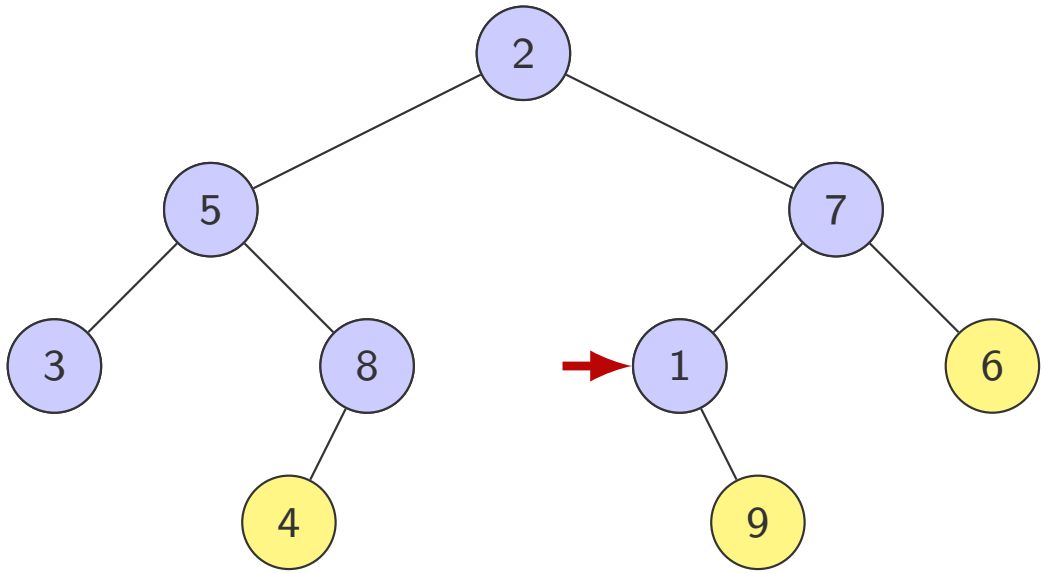


Fila

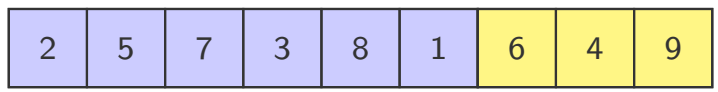
Implementação do percurso em largura

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



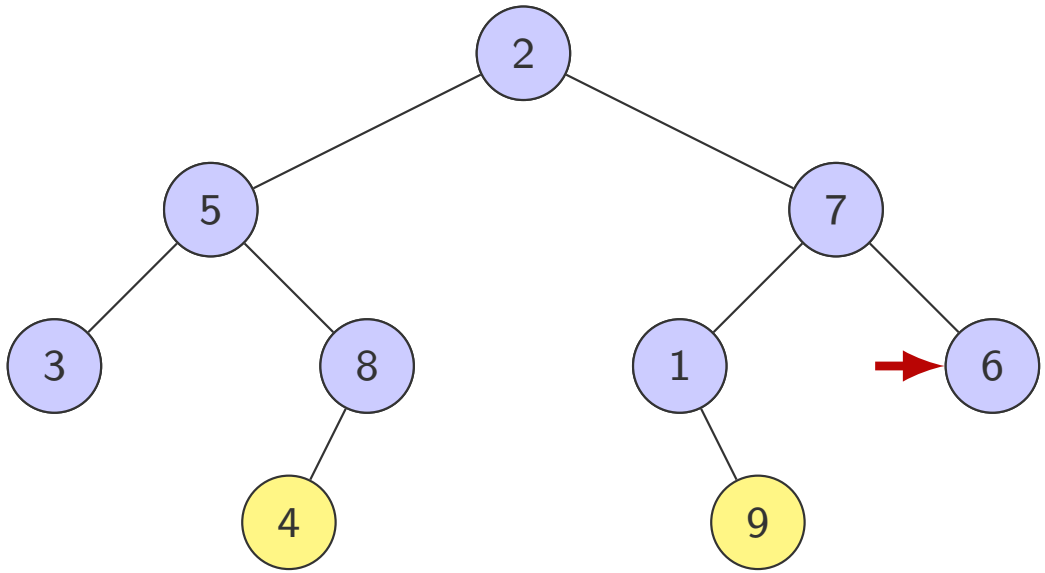
Fila



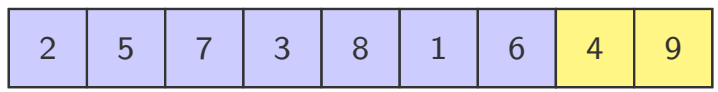
Implementação do percurso em largura

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



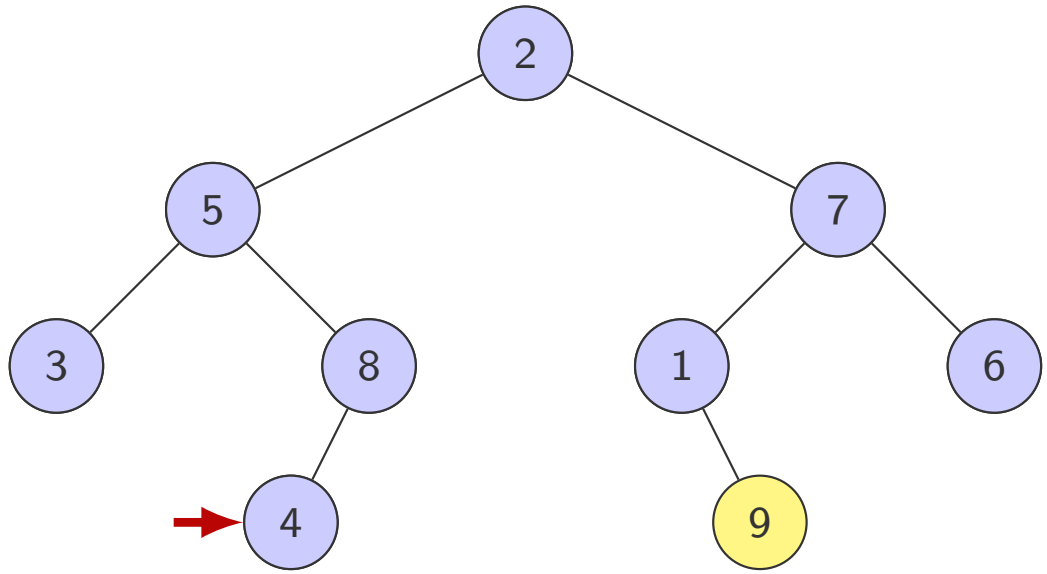
Fila



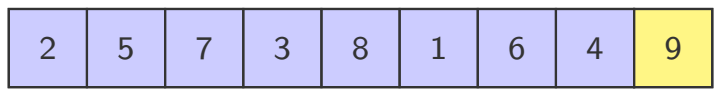
Implementação do percurso em largura

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



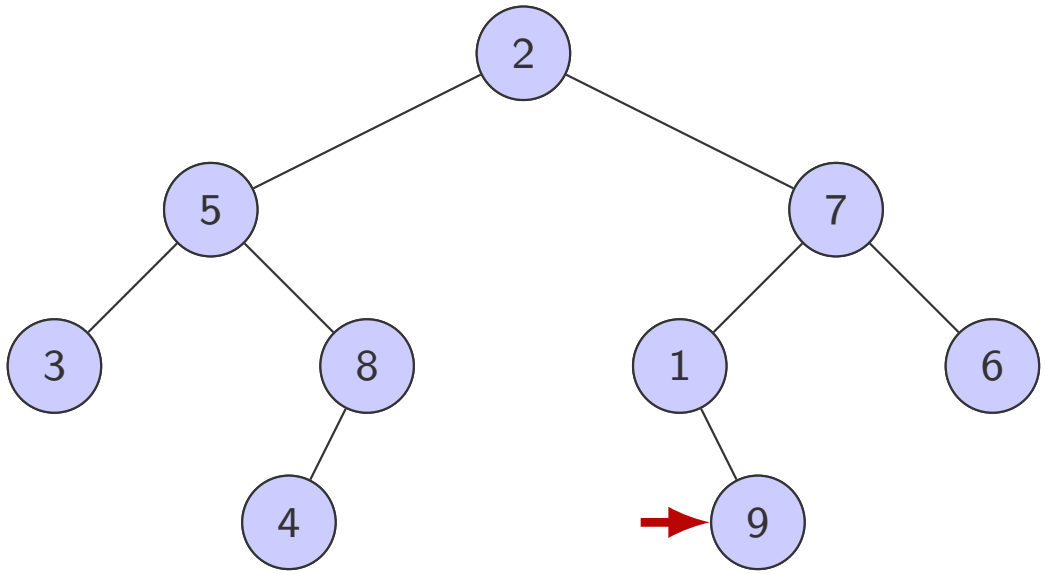
Fila



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

Algorithm levelTraversal()

Require: root (ponteiro para a raiz)

```
1:  Cria uma fila vazia Q de ponteiros para nós
2:  Q.push(root)
3:  while Q ≠ ∅ do
4:      node = Q.front()
5:      Q.pop()
6:      if node ≠ NULL then
7:          visit(node)
8:          Q.push(node→left)
9:          Q.push(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 `std::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