Árvores Binárias

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2022

Leitura para esta unidade

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- **Capítulos 3 e 4**

  Jayme Luiz SZWARCFITER e Lilian MARKENZON.
  Estruturas de dados e seus algoritmos.
  2. ed. rev. Rio de Janeiro: Livros Técnicos e Científicos, c1994. 320 p.
  ISBN 8521610149.

- **Capítulos 14 e 15**

  Paulo FEOFILOFF. Algoritmos em linguagem C.
  Rio de Janeiro: Elsevier, 2009. 208p. ISBN 9788535232493.

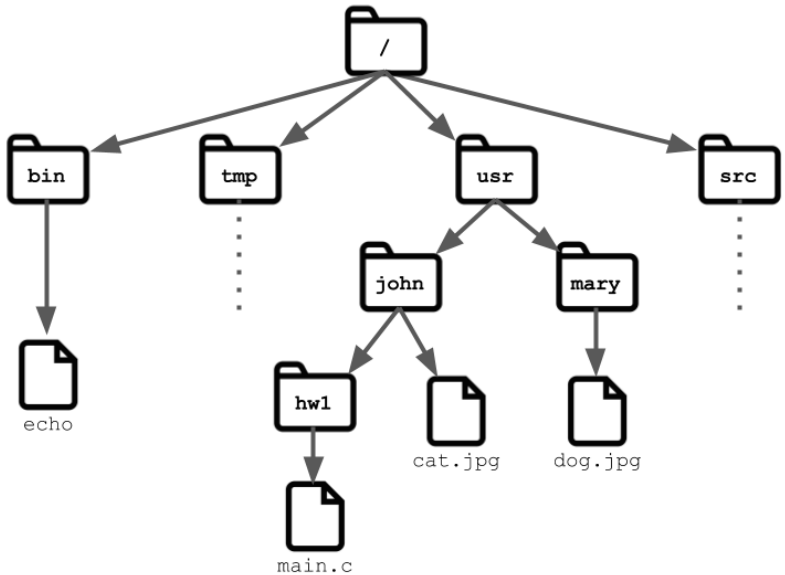
Introdução

Representando uma hierarquia

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

Representando uma hierarquia

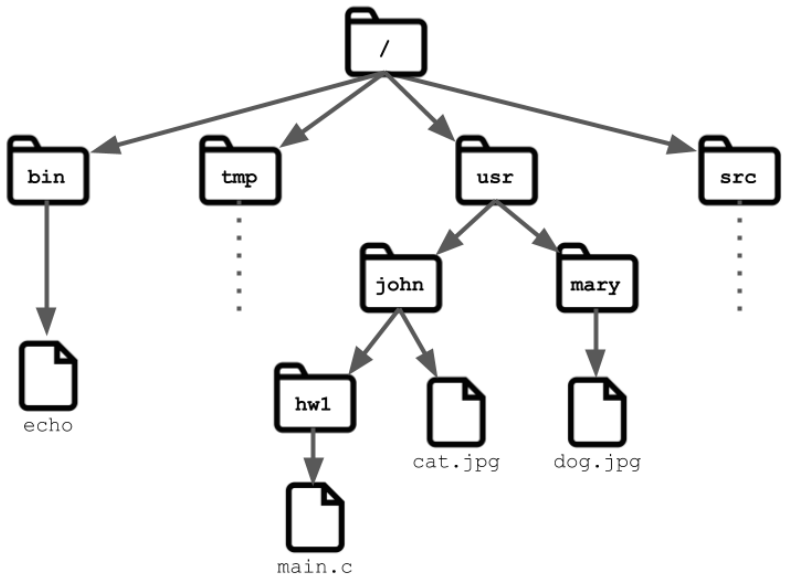
- Vetores e listas são estruturas lineares.
- A importância dessas estruturas é inegável, mas elas não são adequadas para representar dados dispostos de maneira hierárquica.



Hierarquia do sistema de arquivos de um PC Linux

Representando uma hierarquia

- Vetores e listas 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 PC Linux

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

Árvore — Definição Recursiva

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

Árvore — Definição Recursiva

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

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

Árvore — Definição Recursiva

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

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

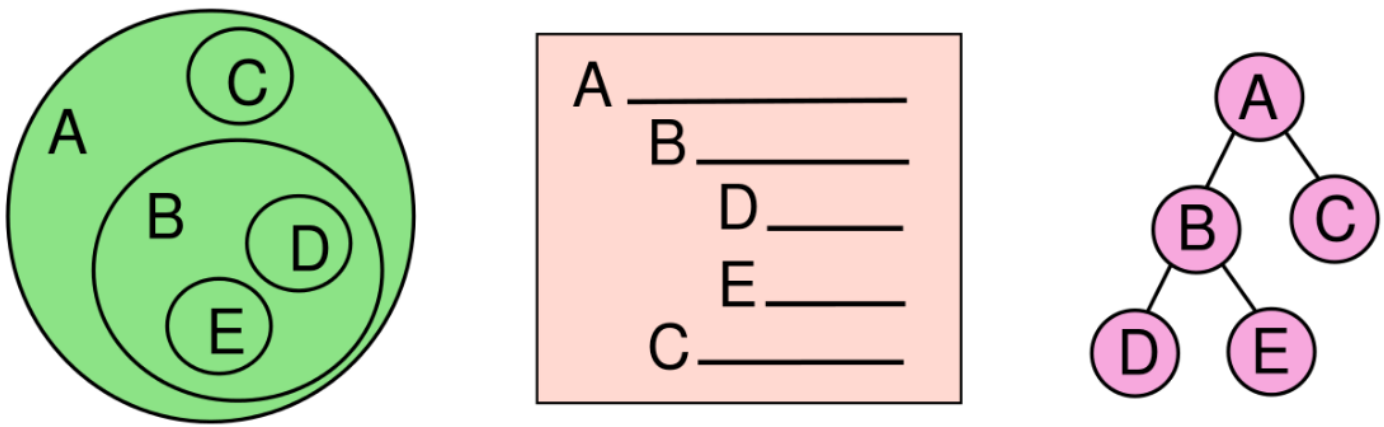
(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz; os nós restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.

Árvore — Definição Recursiva

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

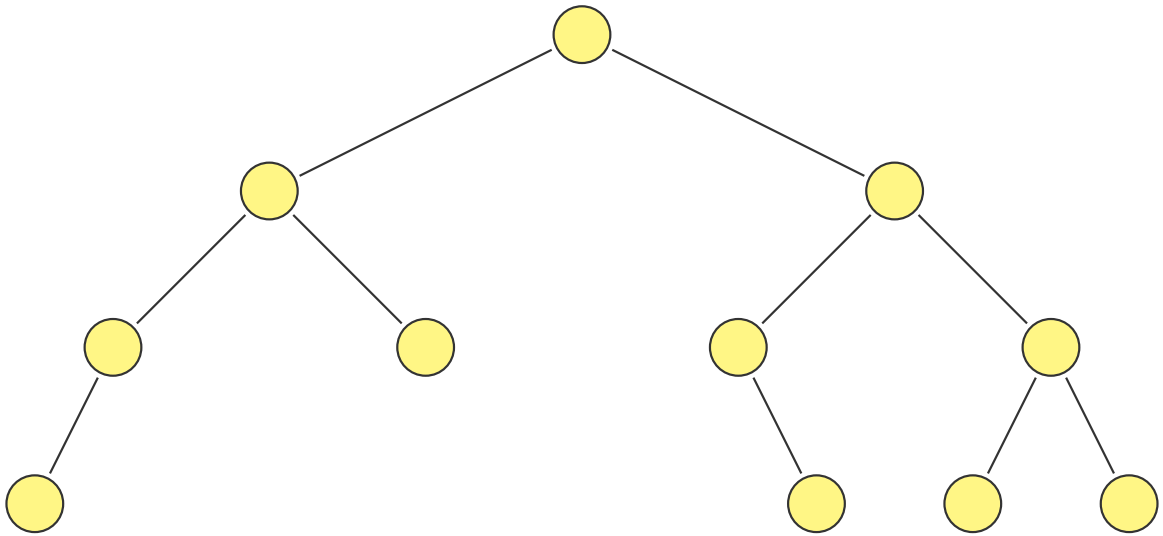
(a) $T = \emptyset$, e a árvore é dita vazia; ou

(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz; 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.



Diferentes representações de uma árvore

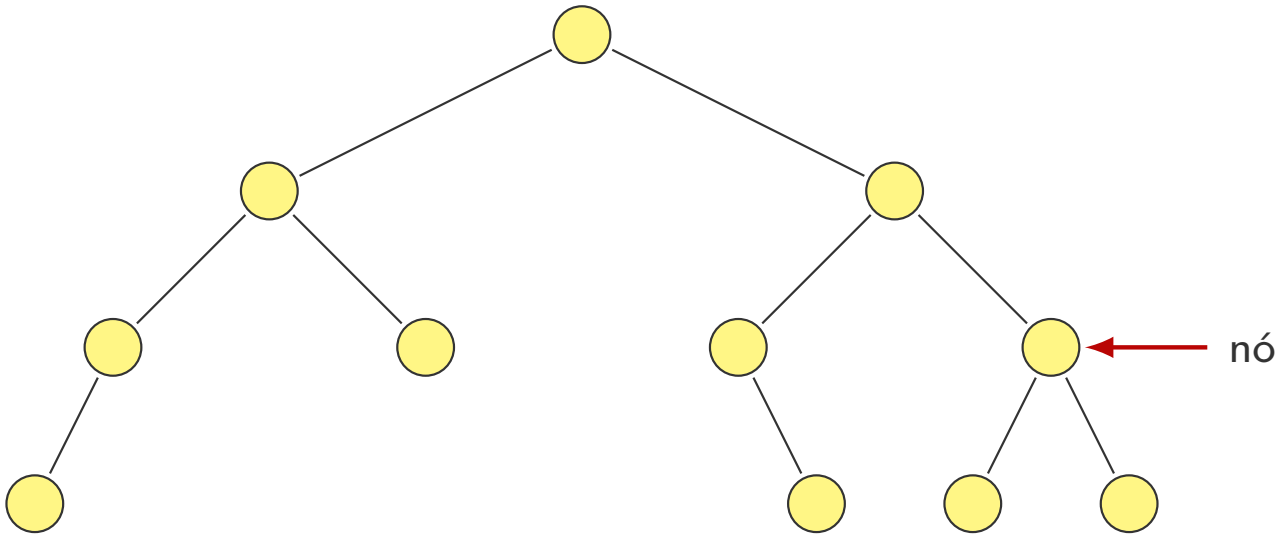
Árvores Binárias



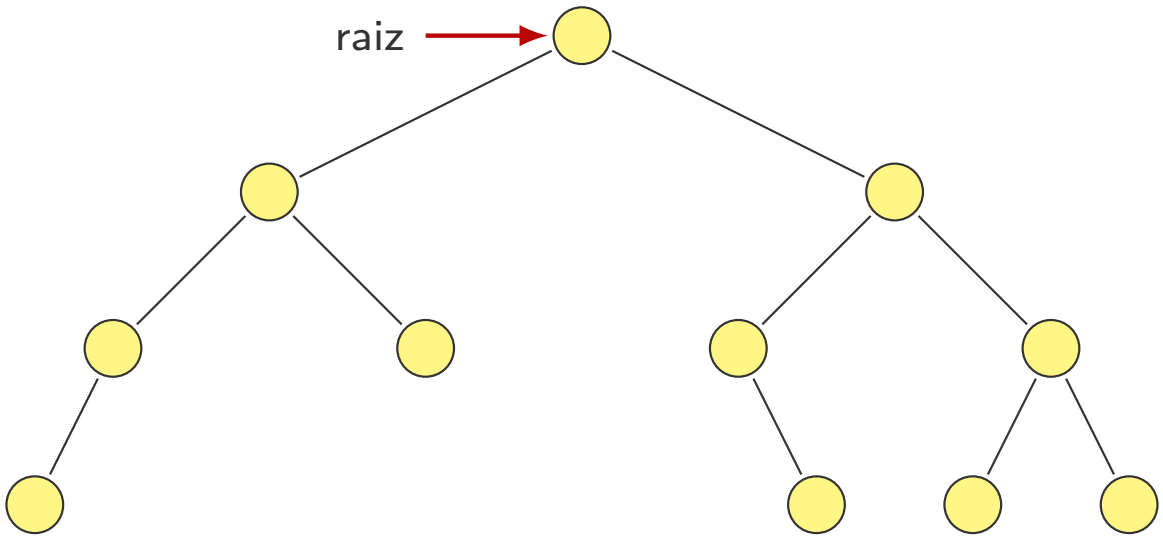
Uma árvore binária é:

- Ou o conjunto vazio
- Ou um nó raiz conectado a exatamente duas subárvores binárias, que podem ser vazias ou não.

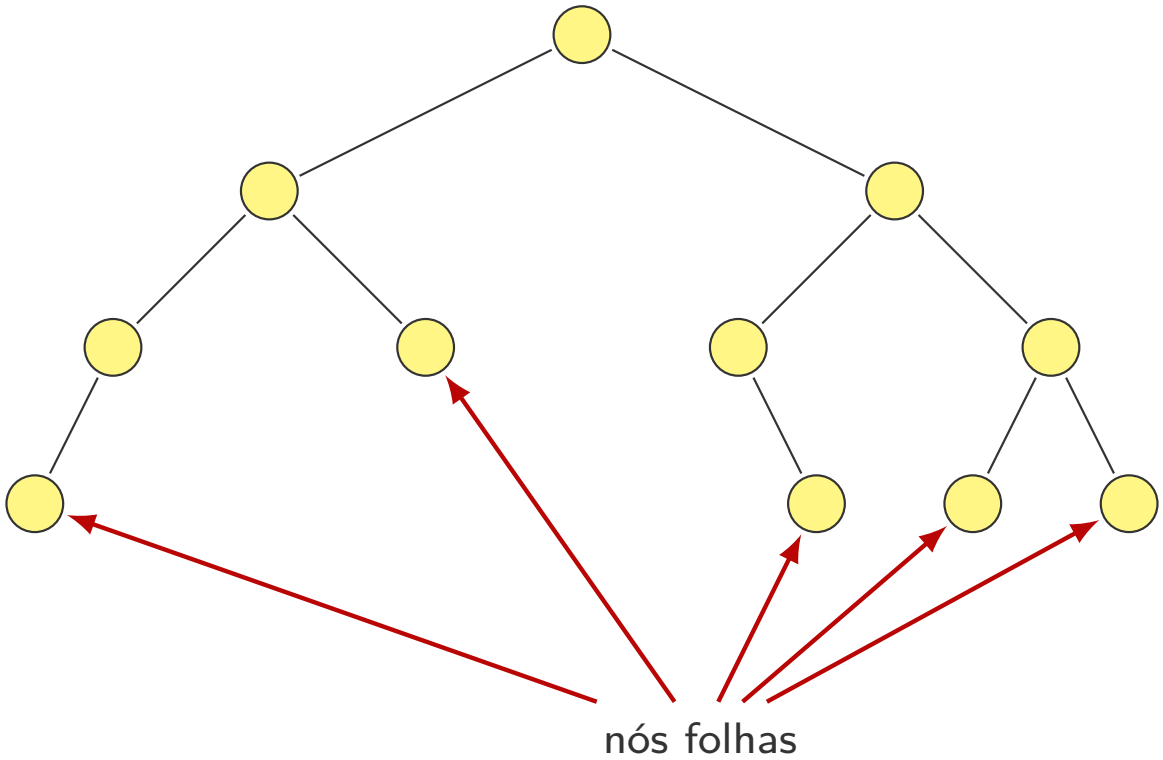
Árvores Binárias



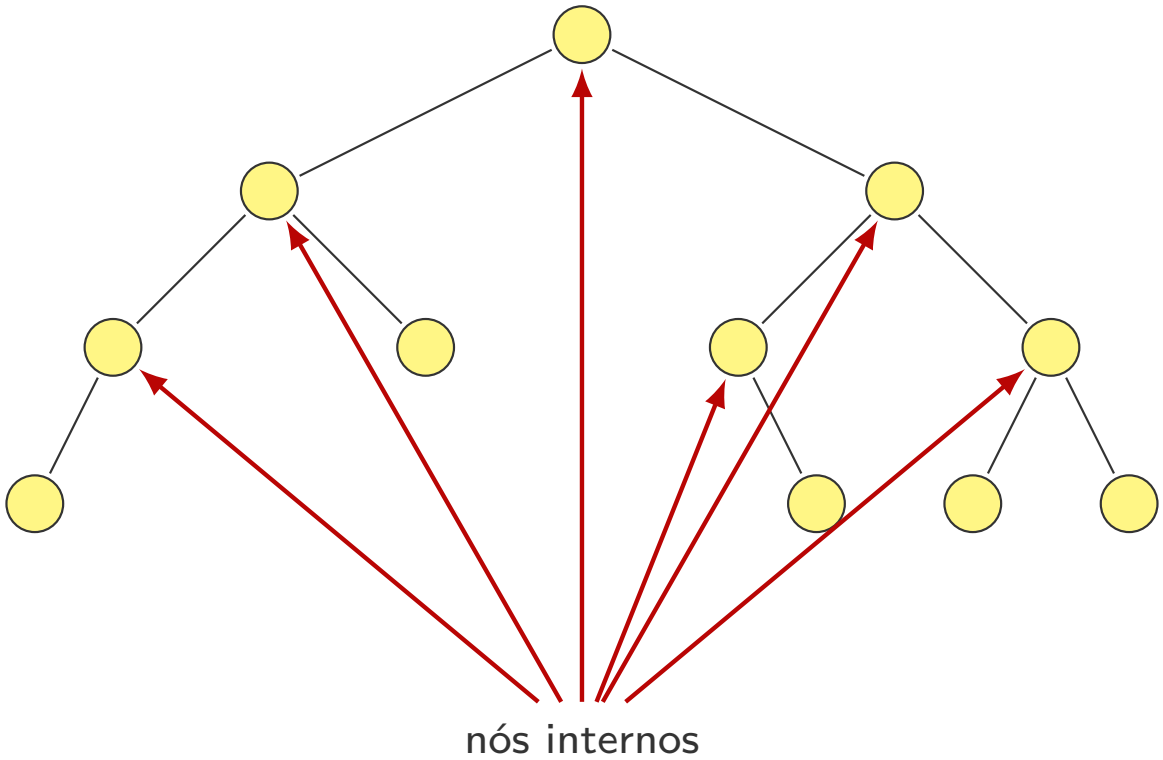
Árvores Binárias



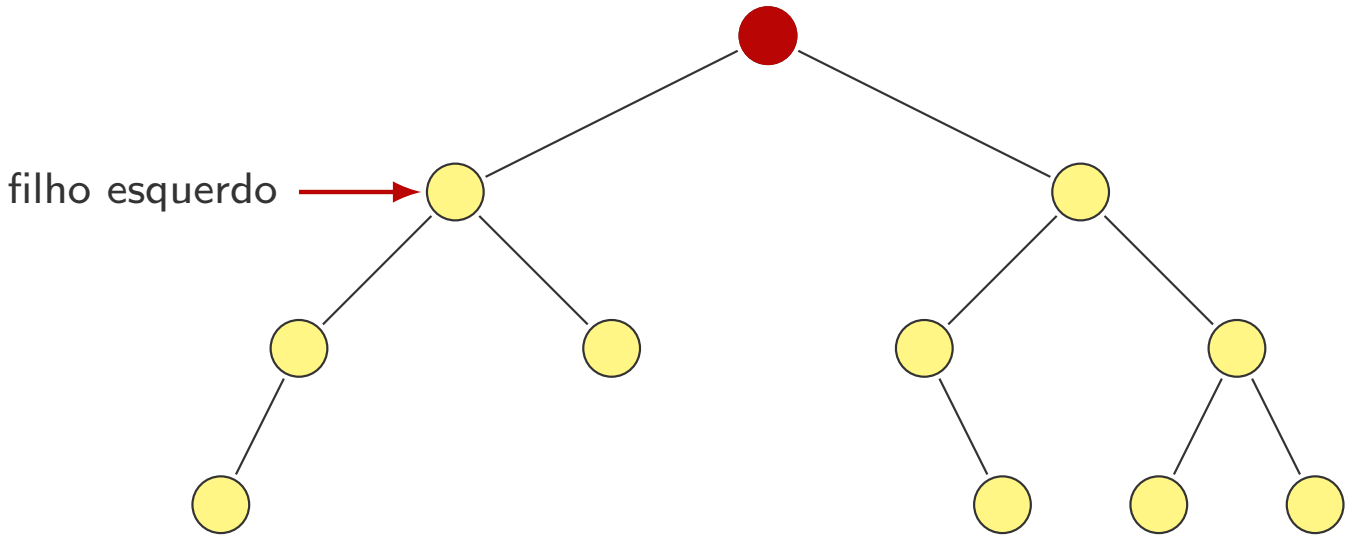
Árvores Binárias



Árvores Binárias

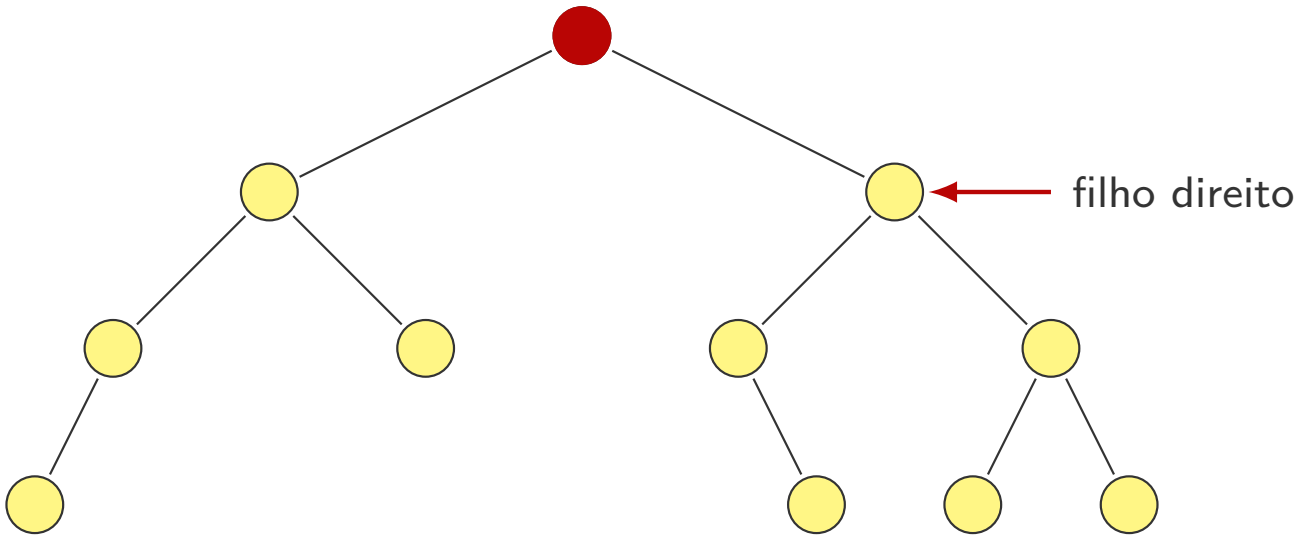


Árvores Binárias

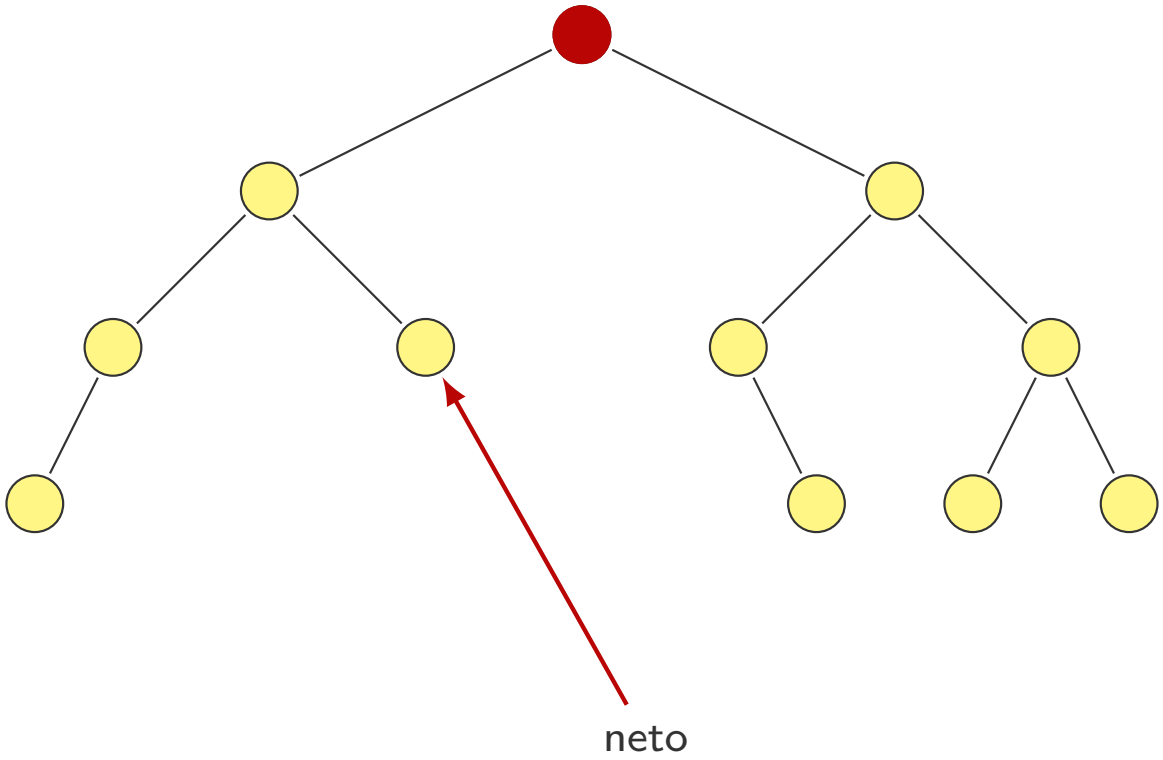


Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

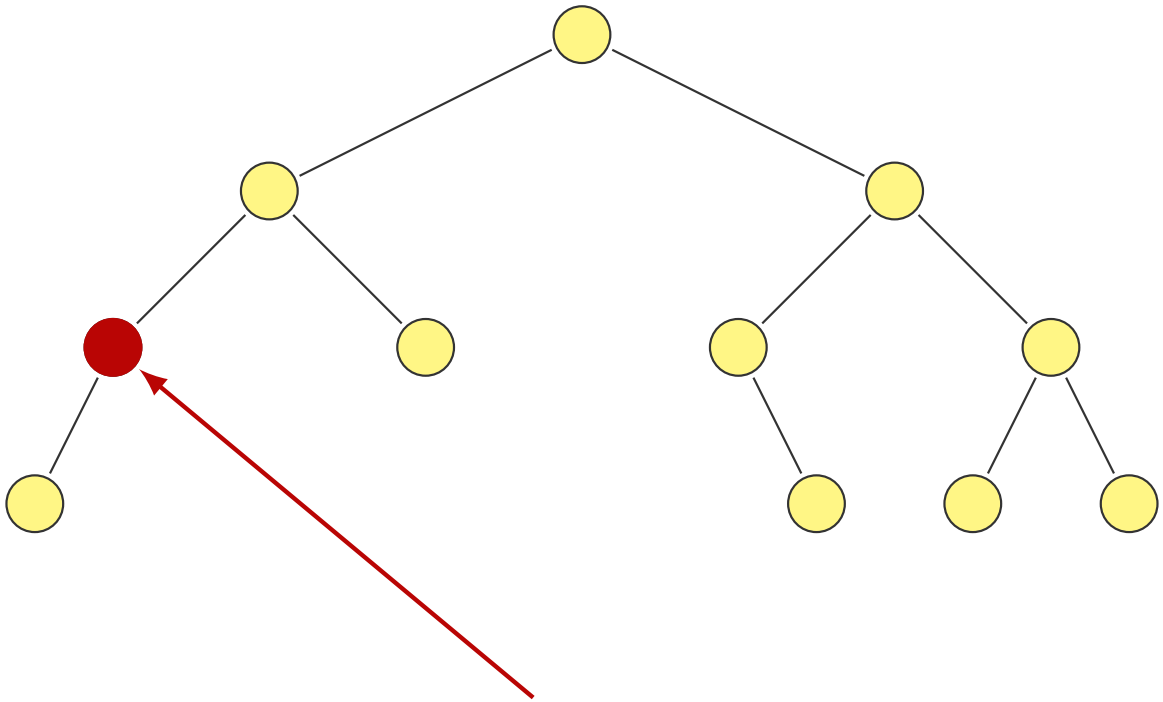


Árvores Binárias

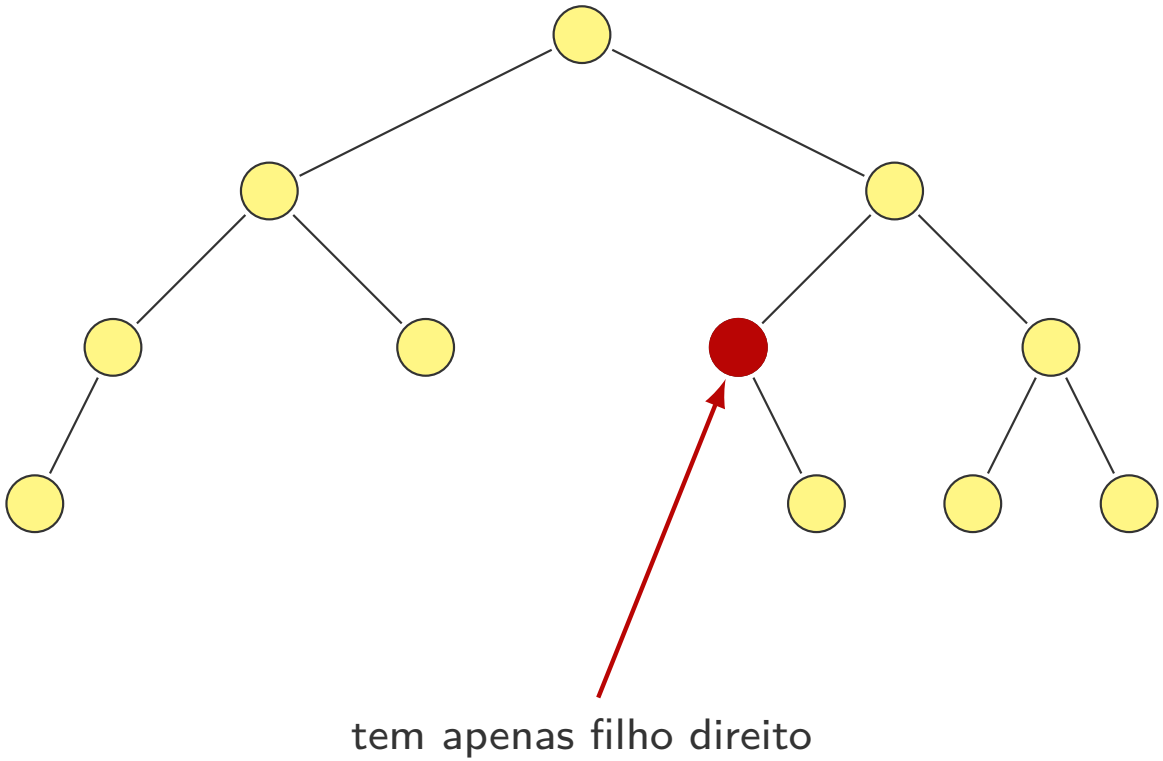


Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

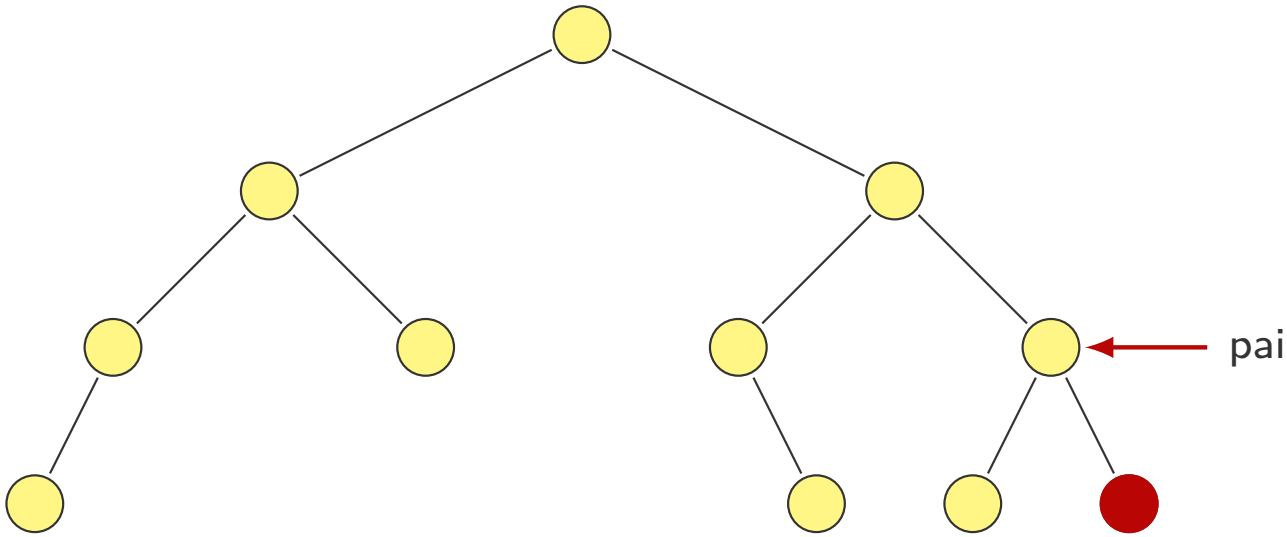


Árvores Binárias



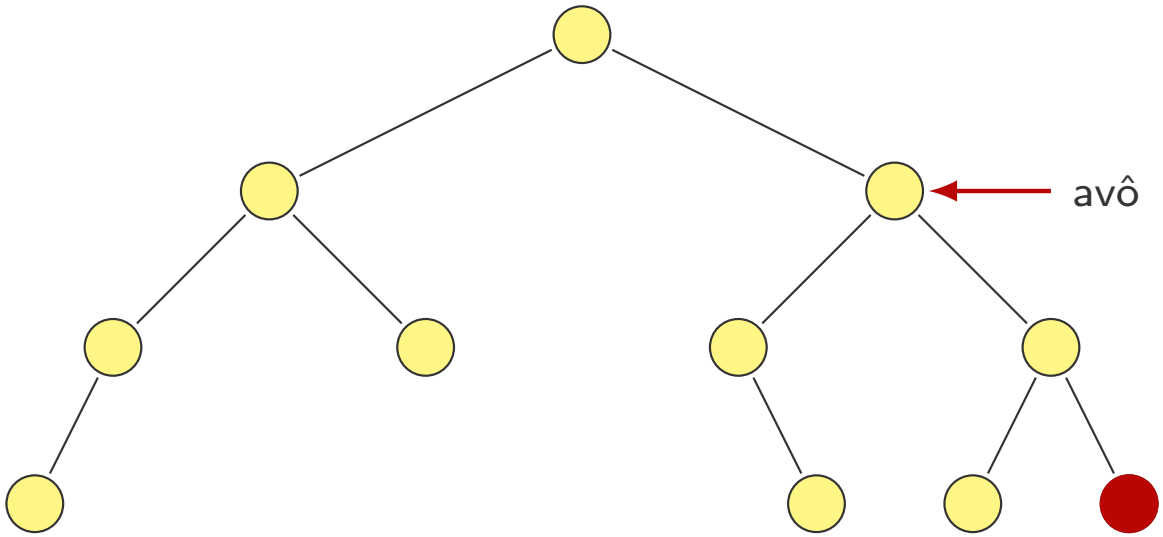
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



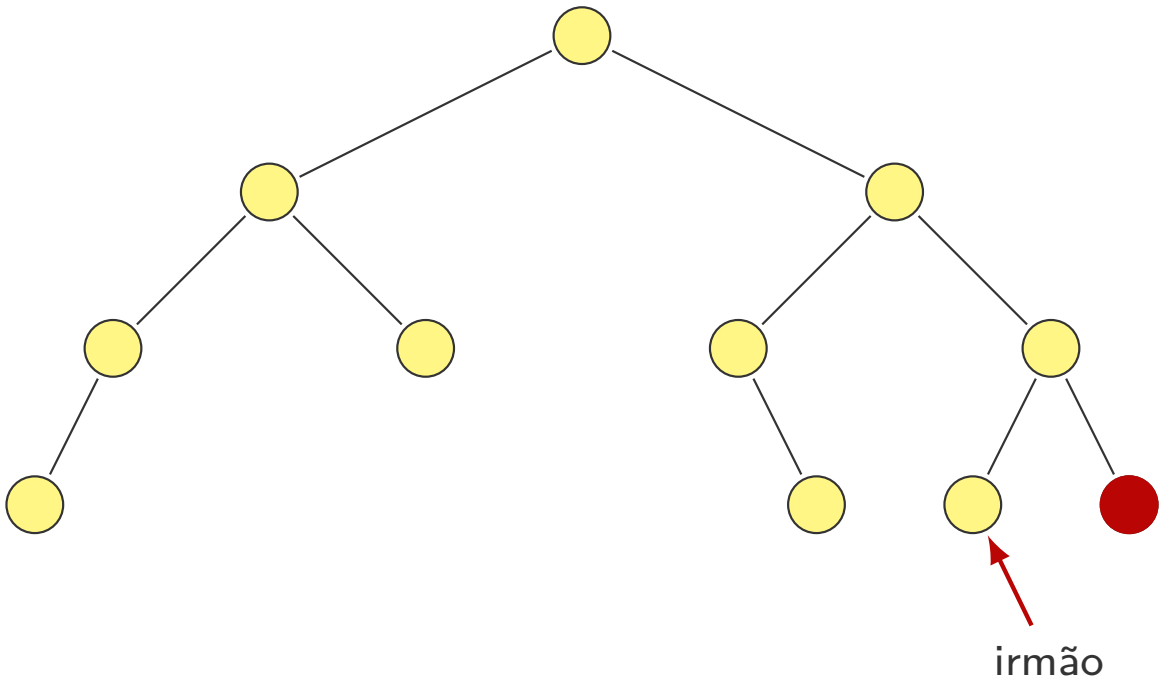
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

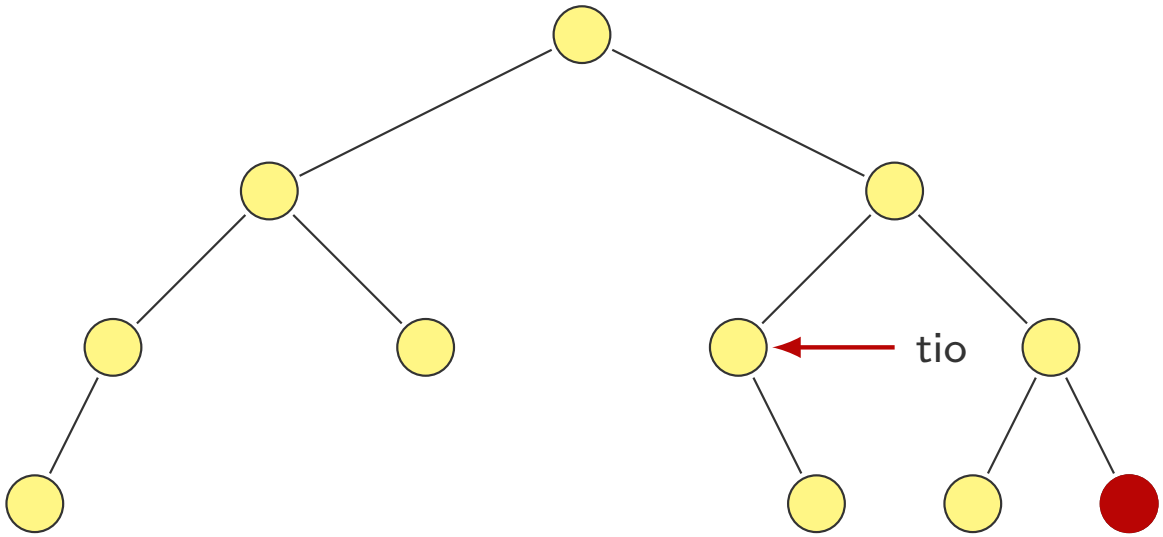


Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

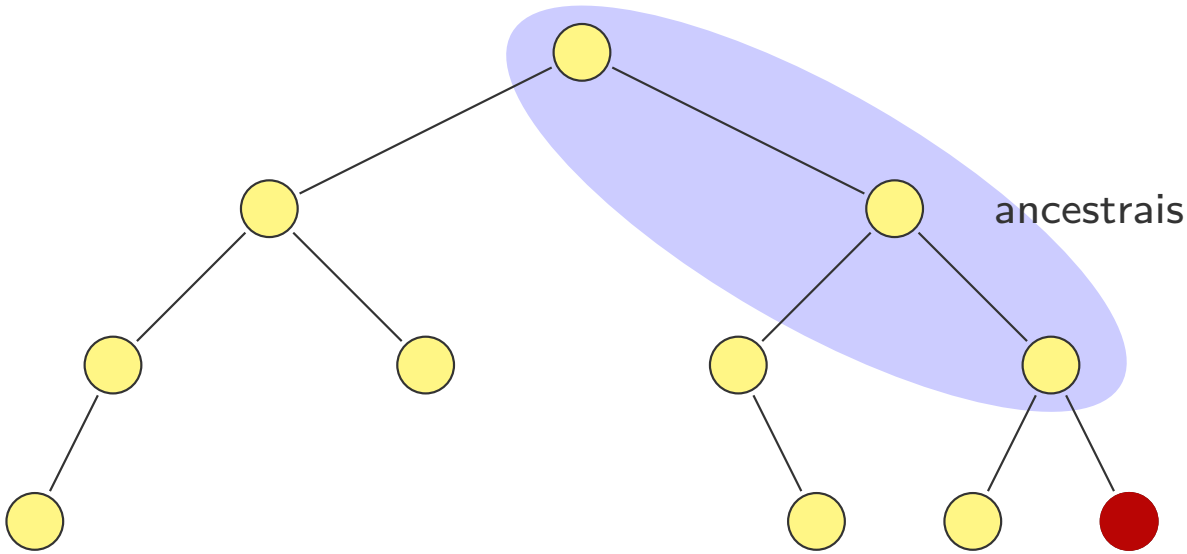


Árvores Binárias



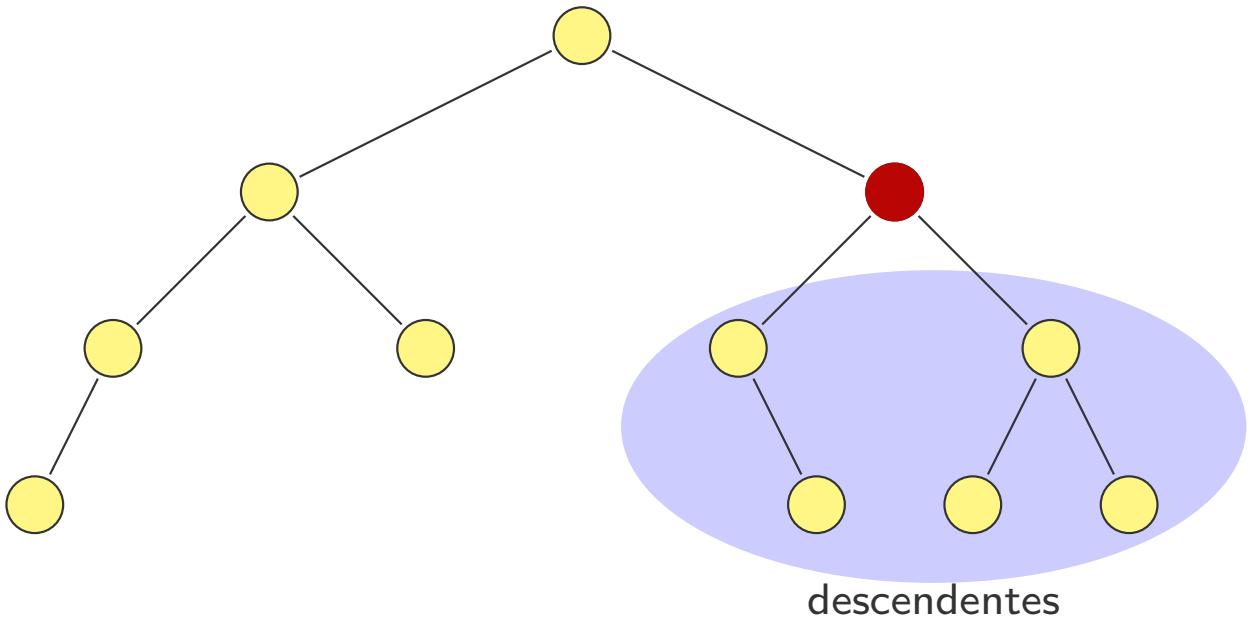
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



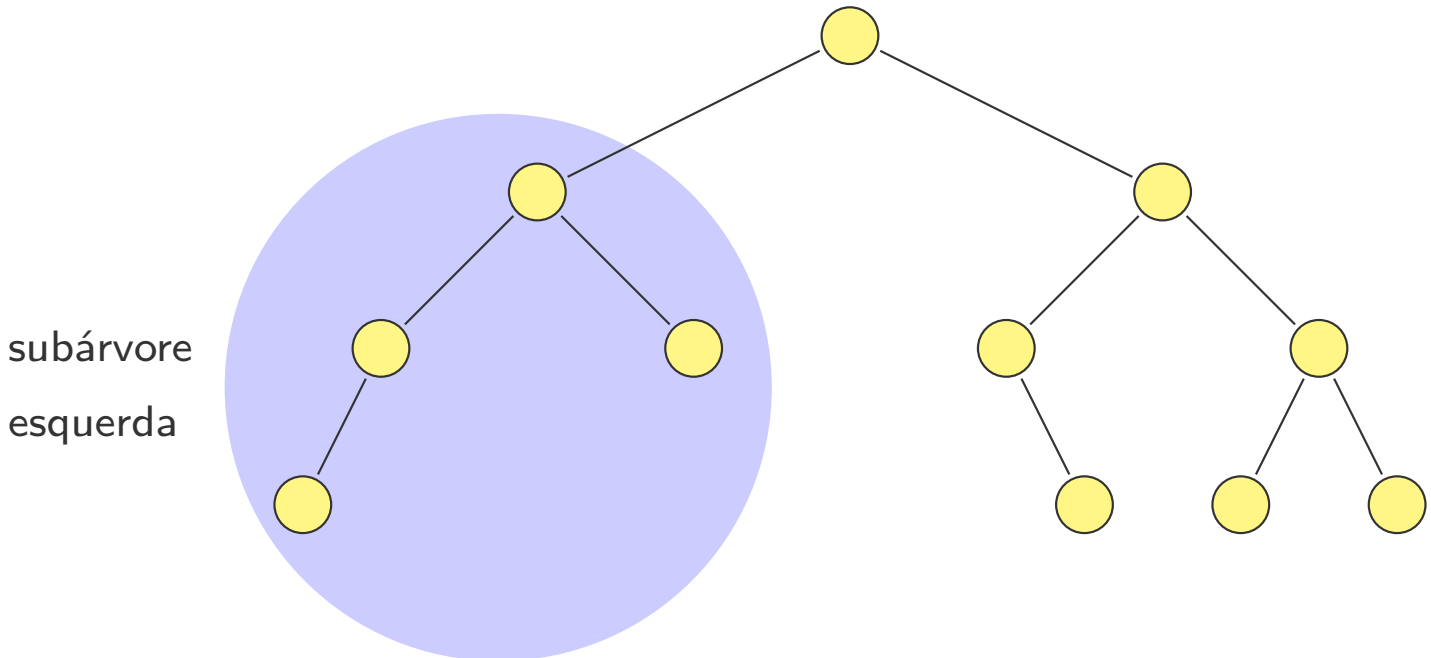
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



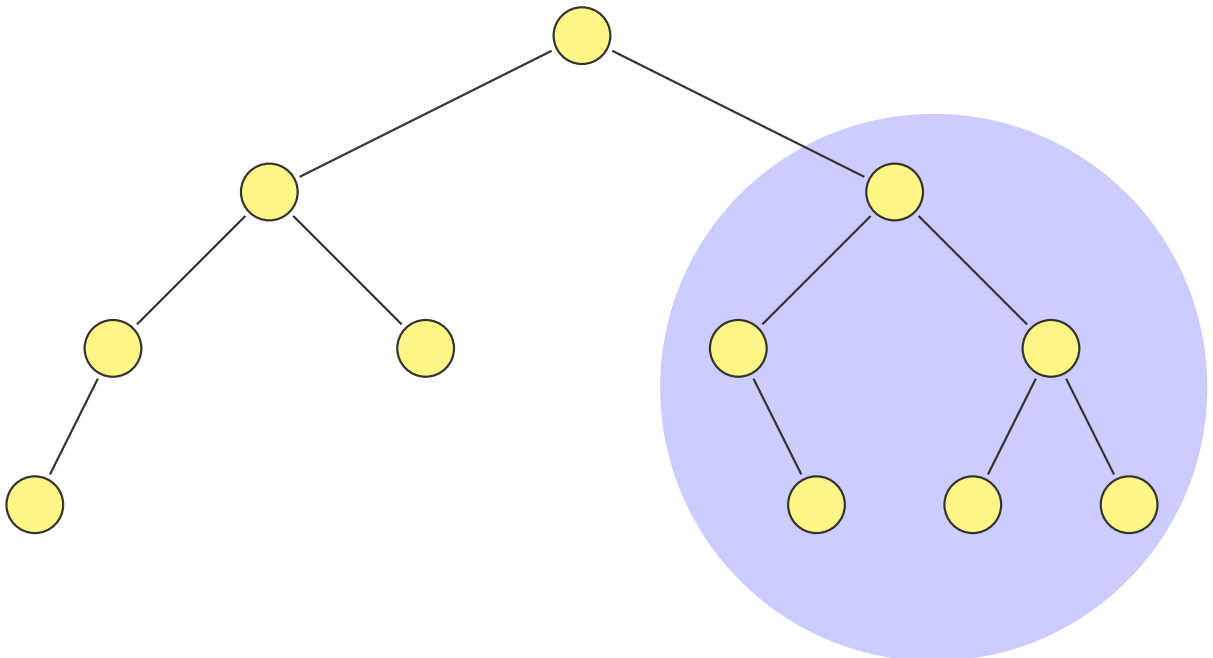
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



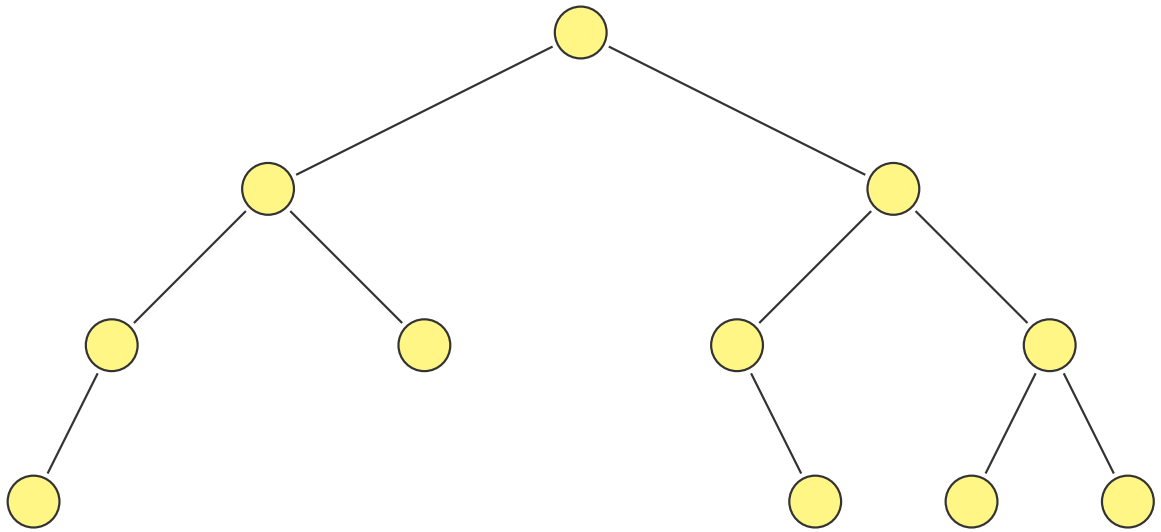
Árvores Binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



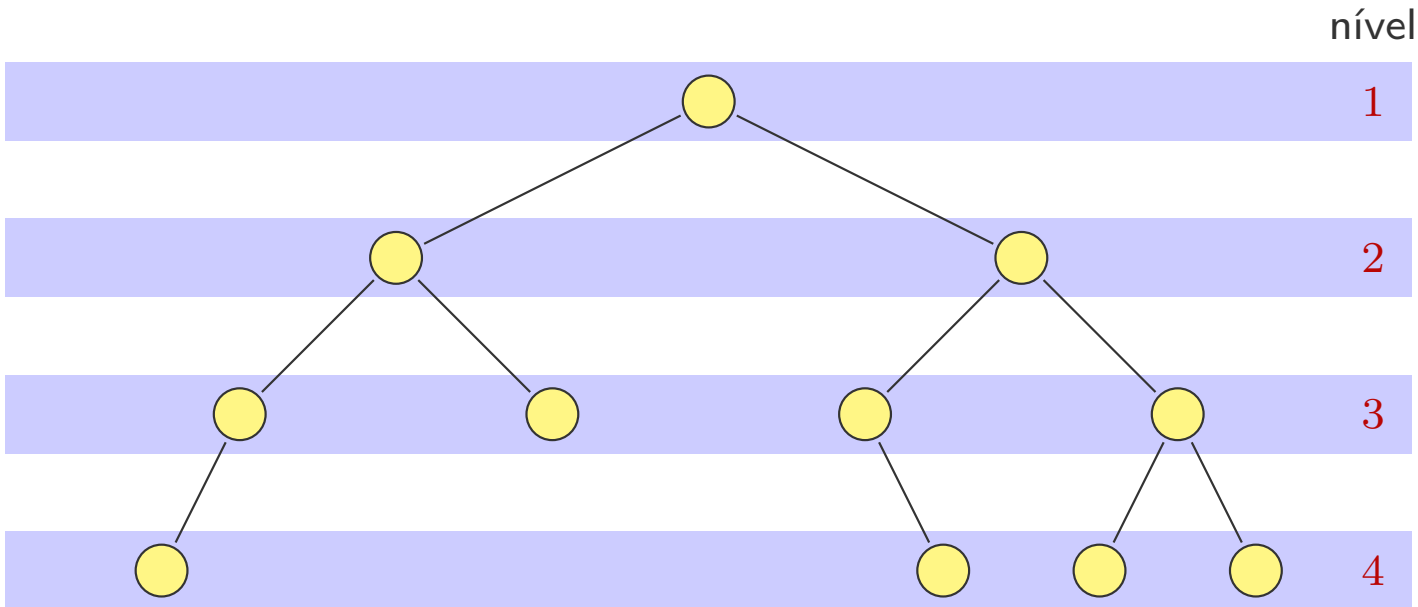
Árvores Binárias — Profundidade, Nível e Altura

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Árvores Binárias — Profundidade, Nível e Altura

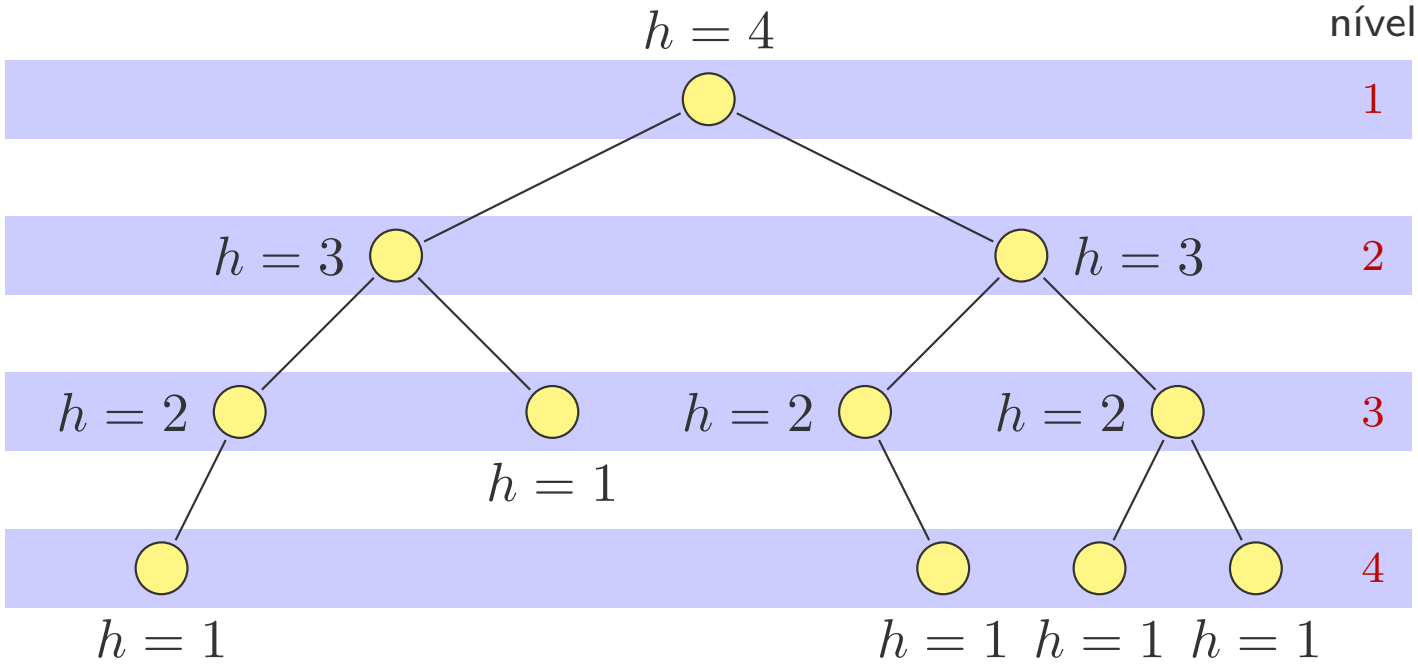
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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.

Árvores Binárias — Profundidade, Nível e Altura

Universidade Federal do Ceará
Campus Quixadá

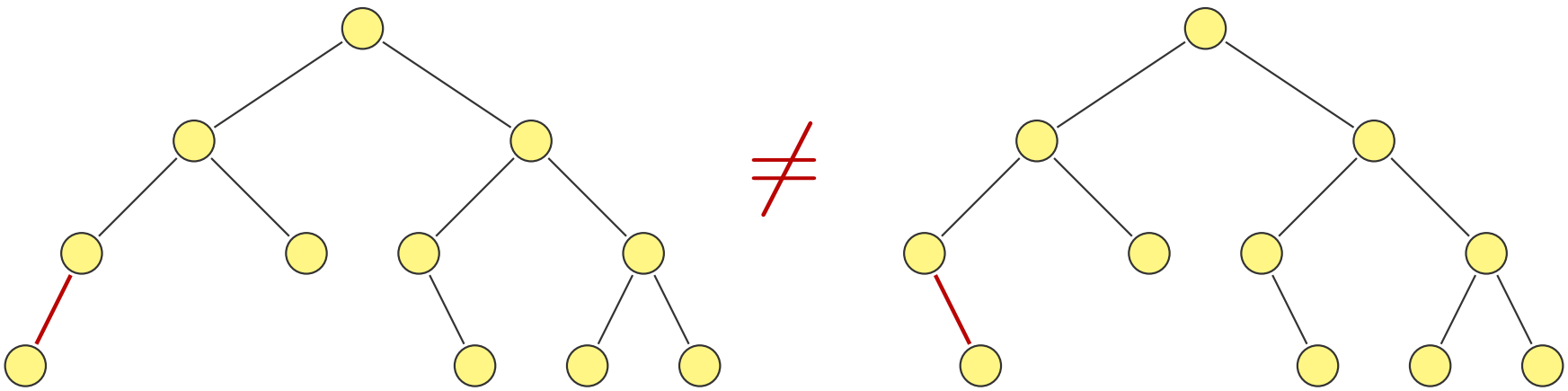


Profundidade de um nó v: Número de nós no caminho de v até a raiz. Dizemos que todos os nós com profundidade i estão no nível i.

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

Comparando com atenção

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Ordem dos filhos é relevante!

Tipos específicos de árvores binárias

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

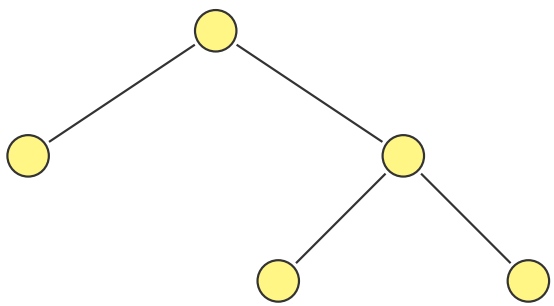
Tipos específicos de árvores binárias

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

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.

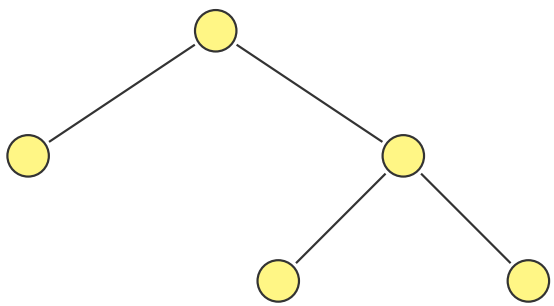


binária completa

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.

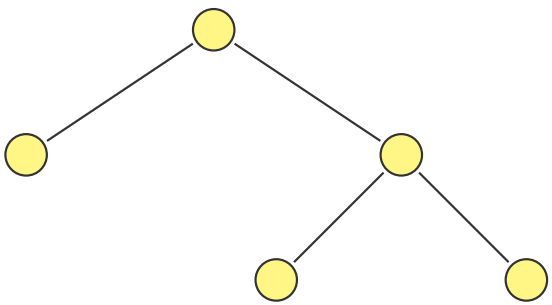


binária completa

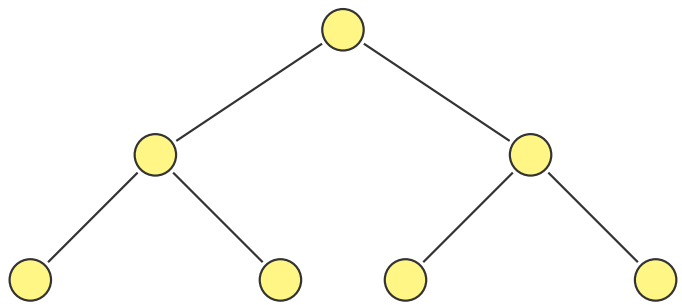
Tipos específicos de árvores binárias

- 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

Universidade Federal do Ceará
Campus Quixadá

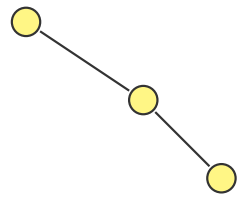
Se a altura é h, então a árvore binária:

Relação entre altura e número de nós da Árvore Binária

Universidade Federal do Ceará
Campus Quixadá

Se a altura é h, então a árvore binária:

- tem no mínimo h nós

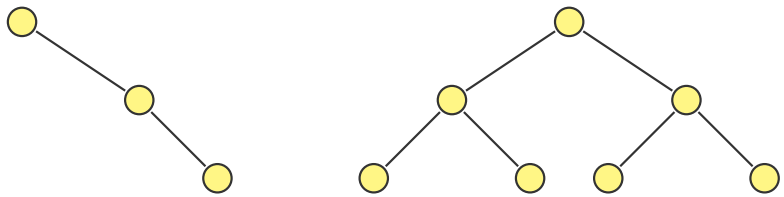


Relação entre altura e número de nós da Árvore Binária

Universidade Federal do Ceará
Campus Quixadá

Se a altura é h, então a árvore binária:

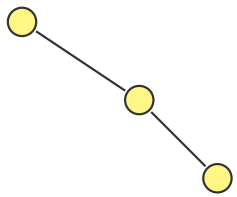
- tem no mínimo h nós
- tem no máximo $2^h - 1$ nós

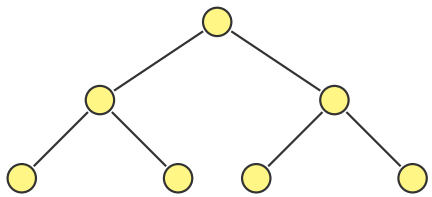


Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



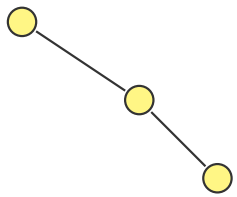


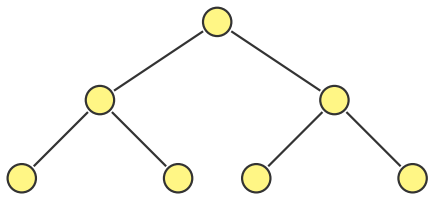
Se a árvore binária tem $n \geq 1$ nós, então:

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





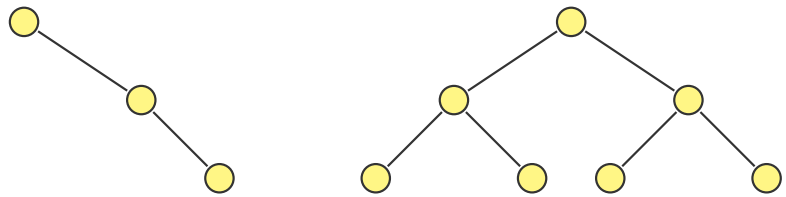
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



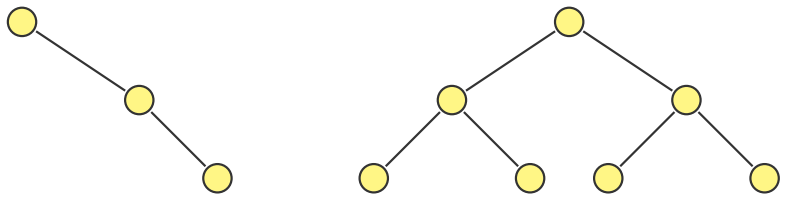
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



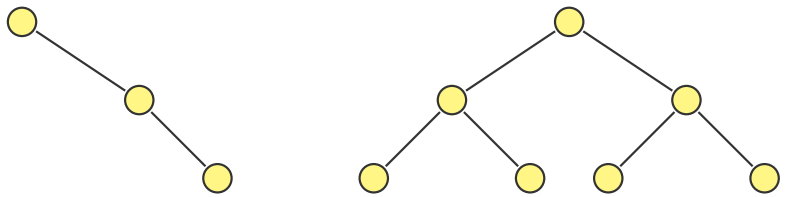
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



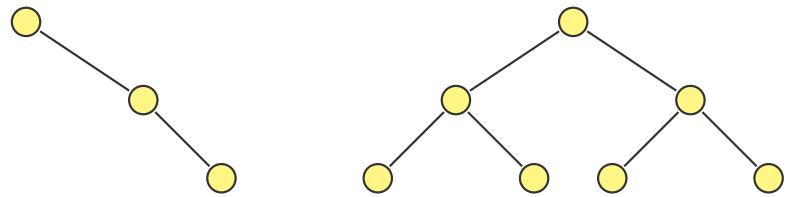
Se a árvore binária tem $n \geq 1$ nós, então:

- 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)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Implementação

Implementação — Decisões de projeto

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Cada nó da árvore será uma estrutura (struct) contendo três campos:

Implementação — Decisões de projeto

- Cada nó da árvore será uma estrutura (struct) contendo três campos:
  - um valor inteiro (chave a ser guardada)

Implementação — Decisões de projeto

- Cada nó da árvore será uma estrutura (struct) contendo três campos:
  - um valor inteiro (chave a ser guardada)
  - um ponteiro para o filho esquerdo do nó

Implementação — Decisões de projeto

- Cada nó da árvore será uma estrutura (struct) contendo três campos:
  - um valor inteiro (chave a ser guardada)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó

Implementação — Decisões de projeto

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Cada nó da árvore será uma estrutura (struct) contendo três campos:
  - um valor inteiro (chave a ser guardada)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó

- Para acessar qualquer nó da árvore, basta termos o endereço do nó raiz. Portanto, a única informação necessária é um ponteiro para a raiz da árvore.

Implementação — Decisões de projeto

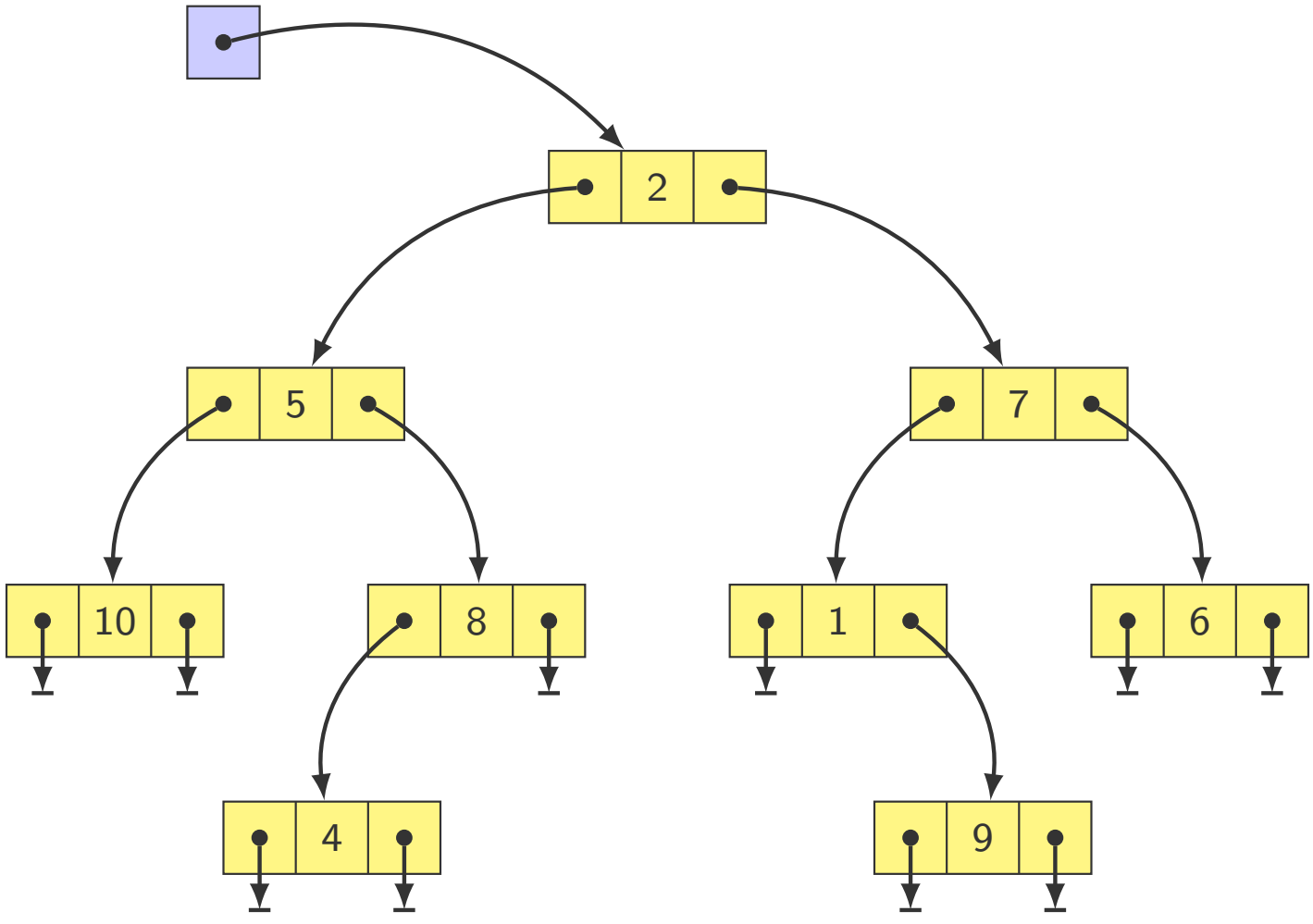
- Cada nó da árvore será uma estrutura (struct) contendo três campos:
  - um valor inteiro (chave a ser guardada)
  - um ponteiro para o filho esquerdo do nó
  - um ponteiro para o filho direito do nó

- Para acessar qualquer nó da árvore, basta termos o endereço do nó raiz. Portanto, a única informação necessária é um ponteiro para a raiz da árvore.

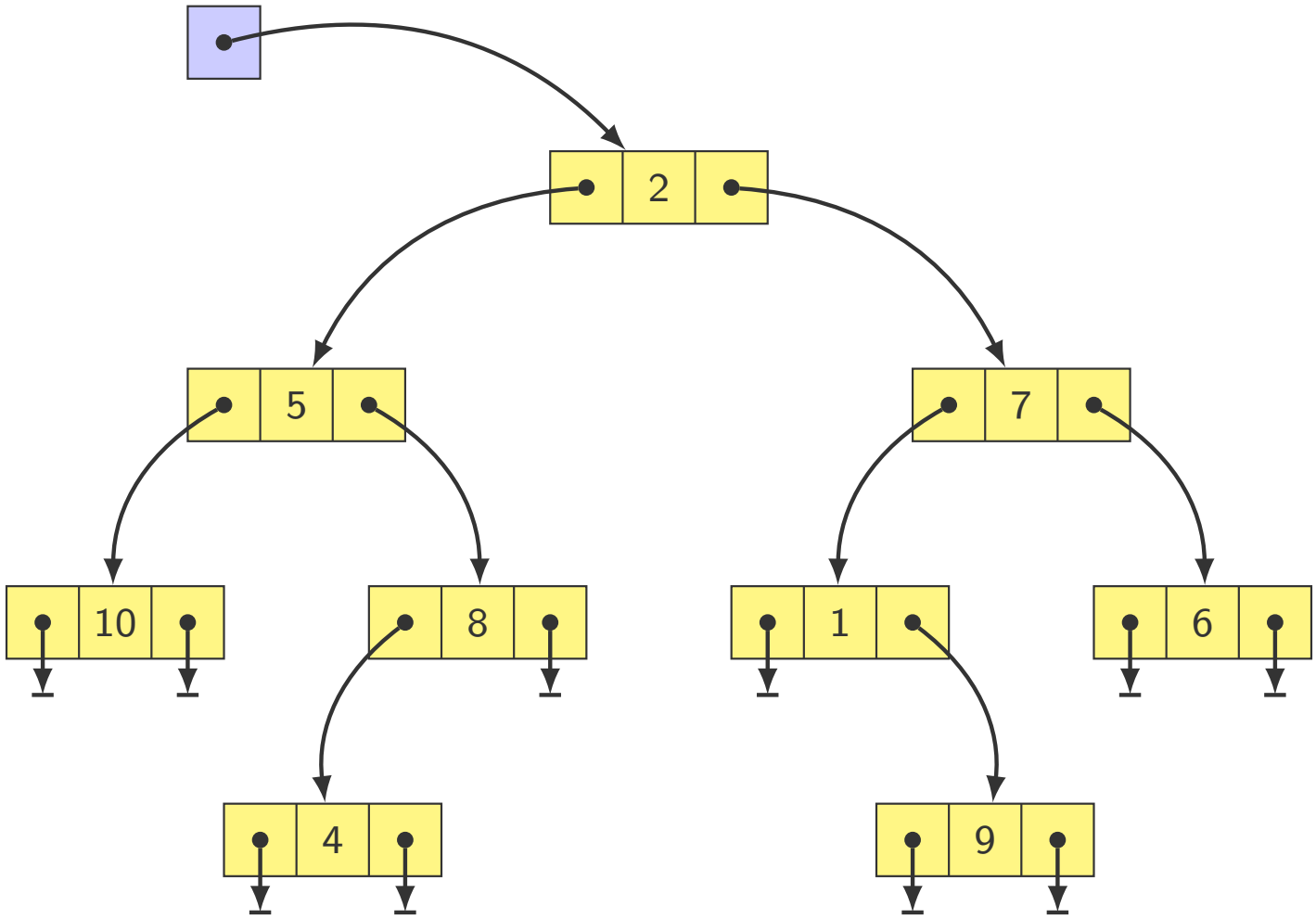
- **Obs.:** Estamos supondo que todas as chaves a serem armazenadas na árvore são distintas.

Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



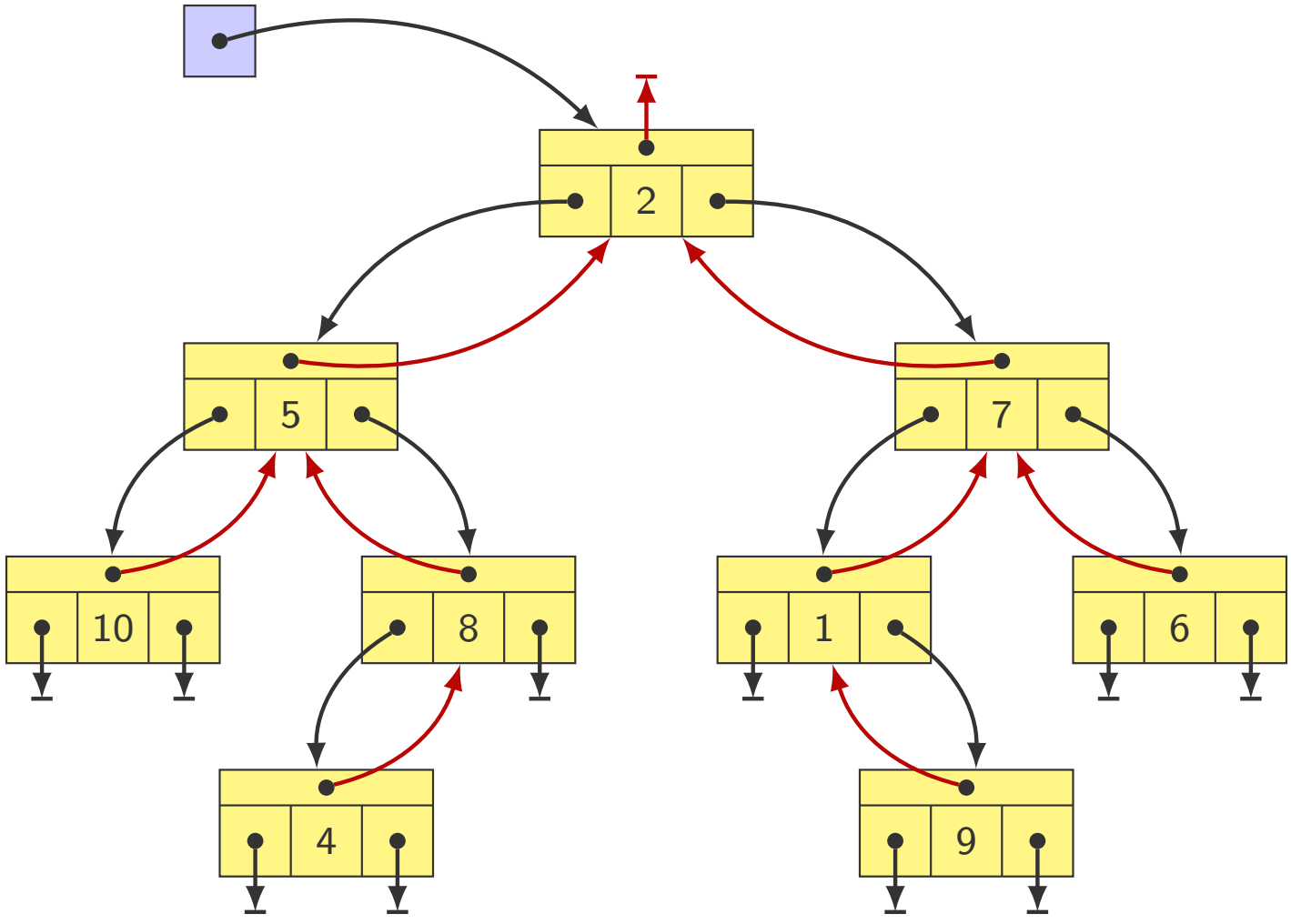
Implementação



E se quisermos saber o pai de um nó? É possível nesta estrutura?

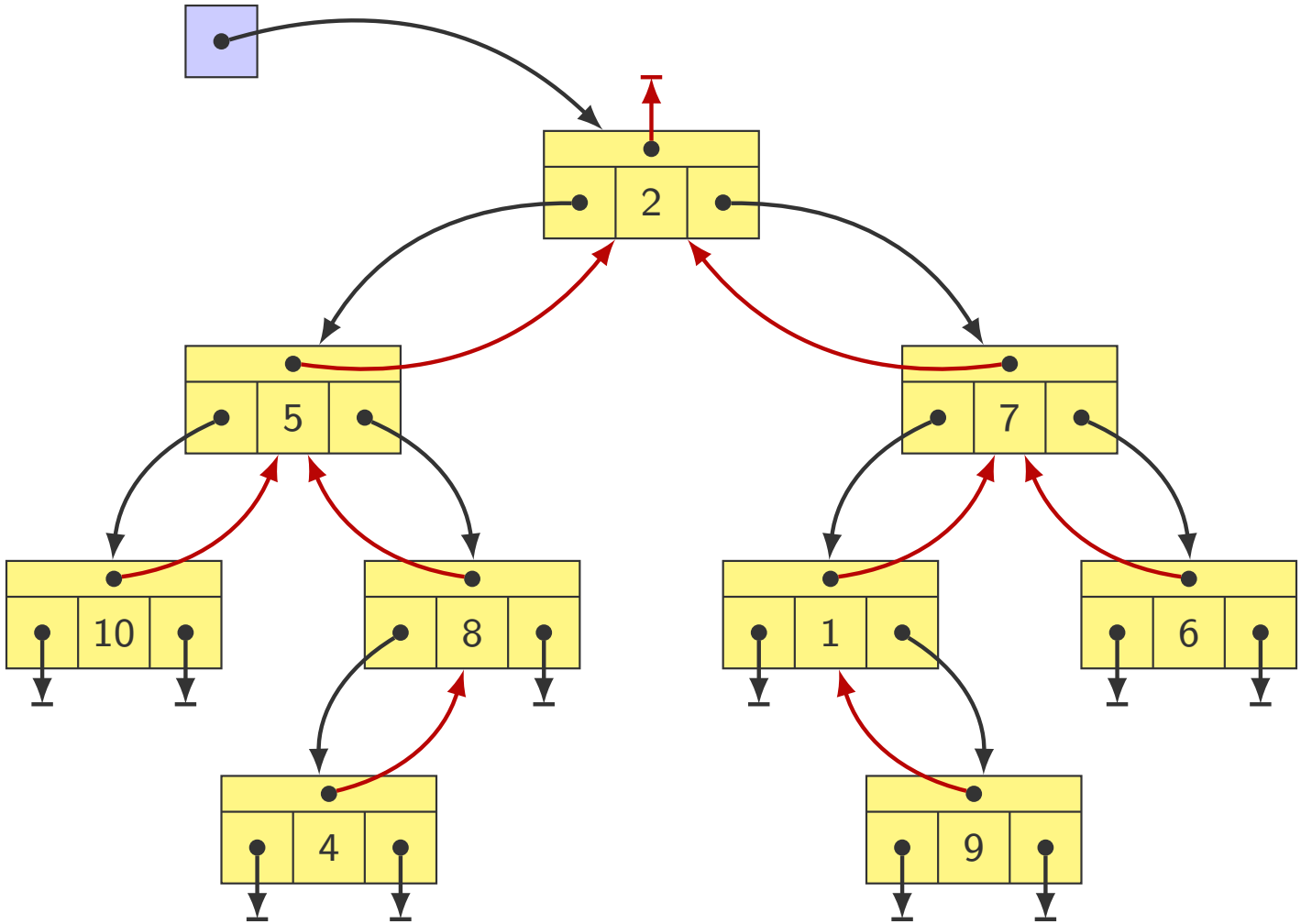
Implementação com ponteiro para pai

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Implementação com ponteiro para pai

Universidade Federal do Ceará
Campus Quixadá



Os nós da árvore implementada nesta aula não terão ponteiro para o pai (fica como exercício para casa).

Arquivo Node.h


```
1  #ifndef NODE_H
2  #define NODE_H
3  #include <iostream>
4
5  struct Node {
6      int    key;     // valor a ser guardado
7      Node* left;    // ponteiro para filho esquerdo
8      Node* right;   // ponteiro para filho direito
9
10     // Construtor do struct Node
11     Node(int k, Node *l, Node *r) {
12         key = k;
13         left = l;
14         right = r;
15     }
16
17     ~Node() {
18         std::cout << key << " deleted\n";
19     }
20 };
21
22 #endif
```

Arquivo BinaryTree.h

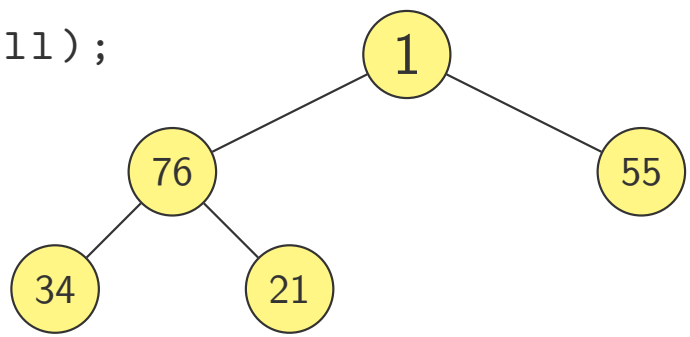
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #ifndef BINARYTREE_H
2  #define BINARYTREE_H
3  #include "Node.h"
4
5  class BinaryTree {
6  private:
7      Node* root; // Pointer to the root of the tree.
8  public:
9      BinaryTree(); // Constructs an empty binary tree.
10     // Constructs a binary tree with the given data
11     // at the root and the two given subtrees.
12     BinaryTree(int, BinaryTree&, BinaryTree&);
13     bool is_empty() const; // is the tree empty?
14     bool is_leaf() const; // is the tree a leaf.
15     bool contains(int) const; // does it contains the value?
16     void print_keys() const; // print all keys
17     void clear(); // leaves the tree empty
18     ~BinaryTree(); // destructor
19 };
20
21 #endif
```

main.cpp — Exemplo de programa cliente

Universidade Federal do Ceará
Campus Quixadá

```
1  #include <iostream>
2  #include "BinaryTree.h"
3  using namespace std;
4
5  int main() {
6      BinaryTree tnull;
7      BinaryTree t34(34, tnull, tnull);
8      BinaryTree t21(21, tnull, tnull);
9      BinaryTree t76(76, t34, t21);
10     BinaryTree t55(55, tnull, tnull);
11     BinaryTree t1(1, t76, t55);
12
13     t1.print_keys();
14 }
```



BinaryTree.cpp — Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Construtores:

```
1 BinaryTree::BinaryTree() {
2     root = nullptr;
3 }
```

```
1 BinaryTree::BinaryTree(int value, BinaryTree& lchild,
2     BinaryTree& rchild) {
3     root = new Node(value, lchild.root, rchild.root);
4     lchild.root = nullptr;
5     rchild.root = nullptr;
6 }
```

Saber se a árvore é uma folha:

BinaryTree.cpp — Implementação

Construtores:

```
1 BinaryTree::BinaryTree() {
2     root = nullptr;
3 }
```

```
1 BinaryTree::BinaryTree(int value, BinaryTree& lchild,
2     BinaryTree& rchild) {
3     root = new Node(value, lchild.root, rchild.root);
4     lchild.root = nullptr;
5     rchild.root = nullptr;
6 }
```

Saber se a árvore é uma folha:

```
1 bool BinaryTree::is_leaf() const {
2     return !is_empty() && root->left == nullptr && root->right
       == nullptr;
3 }
```

`BinaryTree.cpp` — Implementação

Saber se a árvore é vazia:

BinaryTree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 bool BinaryTree::is_empty() const {
2     return root == nullptr;
3 }
```

BinaryTree.cpp — Implementação

Saber se a árvore é vazia:

```
1 bool BinaryTree::is_empty() const {
2     return root == nullptr;
3 }
```

Percorrendo e imprimindo a árvore:

BinaryTree.cpp — Implementação

Saber se a árvore é vazia:

```
1 bool BinaryTree::is_empty() const {
2     return root == nullptr;
3 }
```

Percorrendo e imprimindo a árvore:

```
1 void print_keys_rec(Node *node) {
2     if(node != nullptr) {
3         cout << node->key << " ";
4         print_keys_rec(node->left);
5         print_keys_rec(node->right);
6     }
7 }
```

BinaryTree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 bool BinaryTree::is_empty() const {
2     return root == nullptr;
3 }
```

Percorrendo e imprimindo a árvore:

```
1 void print_keys_rec(Node *node) {
2     if(node != nullptr) {
3         cout << node->key << " ";
4         print_keys_rec(node->left);
5         print_keys_rec(node->right);
6     }
7 }
```

```
1 void BinaryTree::print_keys() const {
2     print_keys_rec(root);
3     cout << endl;
4 }
```

`BinaryTree.cpp` — Implementação

Buscando uma chave na árvore:

BinaryTree.cpp — Implementação

Buscando uma chave na árvore:

```
1 bool contains_rec(Node *node, int value) {
2     if(node == nullptr)
3         return false; // subárvore vazia
4     else
5         return node->key == value ||
6                contains_rec(node->left, value) ||
7                contains_rec(node->right, value);
8 }
```

BinaryTree.cpp — Implementação

Buscando uma chave na árvore:

```
1 bool contains_rec(Node *node, int value) {
2     if(node == nullptr)
3         return false; // subárvore vazia
4     else
5         return node->key == value ||
6                contains_rec(node->left, value) ||
7                contains_rec(node->right, value);
8 }
```

Observações:

BinaryTree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

```
1 bool contains_rec(Node *node, int value) {
2     if(node == nullptr)
3         return false; // subárvore vazia
4     else
5         return node->key == value ||
6                contains_rec(node->left, value) ||
7                contains_rec(node->right, value);
8 }
```

Observações:

- se o resultado da condição (`node->key == key`) for true, as outras duas expressões não chegam a ser avaliadas.

BinaryTree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

```
1 bool contains_rec(Node *node, int value) {
2     if(node == nullptr)
3         return false; // subárvore vazia
4     else
5         return node->key == value ||
6                contains_rec(node->left, value) ||
7                contains_rec(node->right, value);
8 }
```

Observações:

- se o resultado da condição (`node->key == key`) for true, as outras duas expressões não chegam a ser avaliadas.
  - por sua vez, se a chave for encontrada na subárvore esquerda, a busca não prossegue na subárvore da direita.

BinaryTree.cpp — Implementação

Buscando uma chave na árvore (função pública):

```
1 bool BinaryTree::contains(int value) const {
2     return contains_rec(root, value);
3 }
```

BinaryTree.cpp — Implementação

Liberando memória alocada para a árvore:

```
1 Node *clear_rec(Node *node) {
2     if(node != nullptr) {
3         node->left = clear_rec(node->left);
4         node->right = clear_rec(node->right);
5         delete node;
6     }
7     return nullptr;
8 }
```

BinaryTree.cpp — Implementação

Liberando memória alocada para a árvore:

```
1 Node *clear_rec(Node *node) {
2     if(node != nullptr) {
3         node->left = clear_rec(node->left);
4         node->right = clear_rec(node->right);
5         delete node;
6     }
7     return nullptr;
8 }
```

```
1 void BinaryTree::clear() {
2     root = clear_rec(root);
3 }
```

BinaryTree.cpp — Implementação

Destrutor:

```
1 BinaryTree::~BinaryTree() {
2     clear();
3 }
```

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.

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM