Árvores

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará

1º semestre/2020

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Introdução

Representando uma hierarquia

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

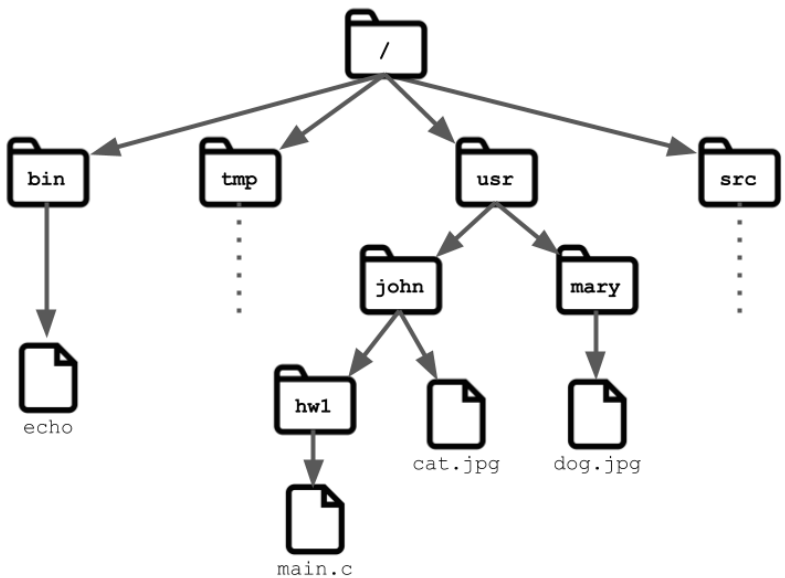


Figura: Hierarquia do sistema de arquivos de um PC Linux

Representando uma hierarquia

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

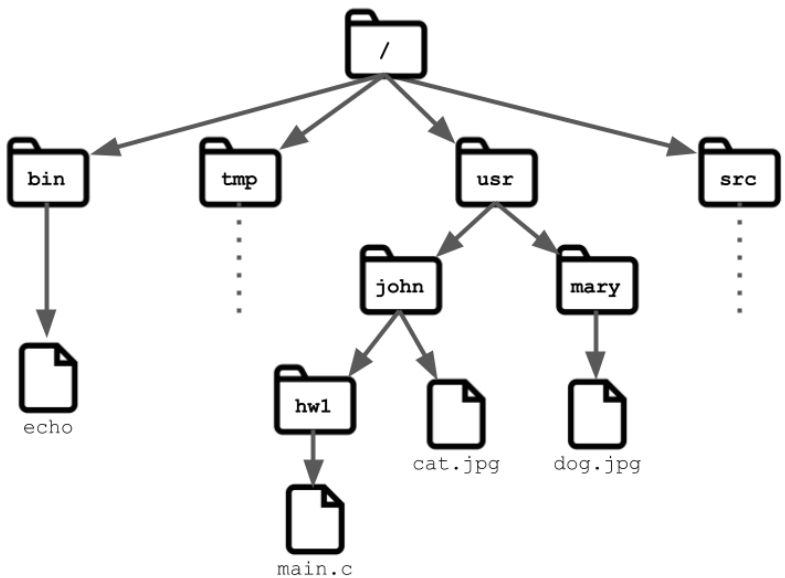


Figura: Hierarquia do sistema de arquivos de um PC Linux

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

Árvore — Definição Recursiva

Uma árvore 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

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

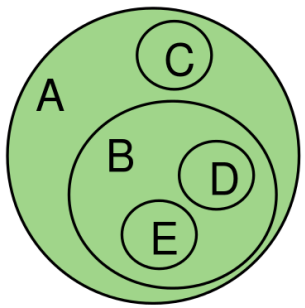
(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.

Árvore — Definição Recursiva

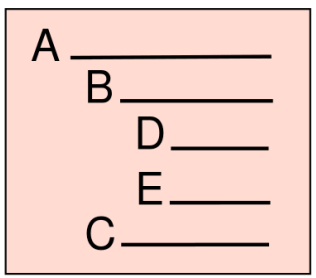
Uma árvore enraizada T, ou simplesmente árvore, é um conjunto finito de elementos denominados nós, tais que:

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

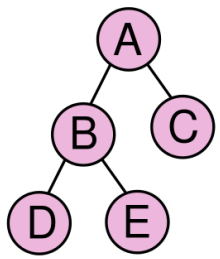
(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.



c)



b)



a)

1A; 1.1B; 1.1.1D; 1.1.2E; 1.2C

e)

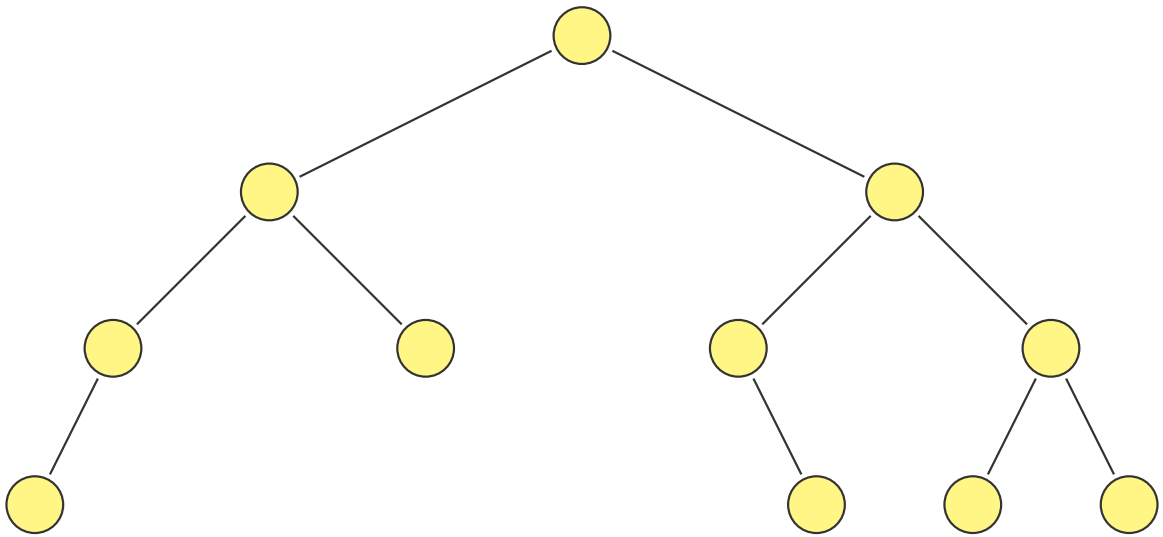
(A(B(D)(E))(C))

d)

Árvores Binárias

Universidade Federal do Ceará
Campus Quixadá

Exemplo de árvore binária:

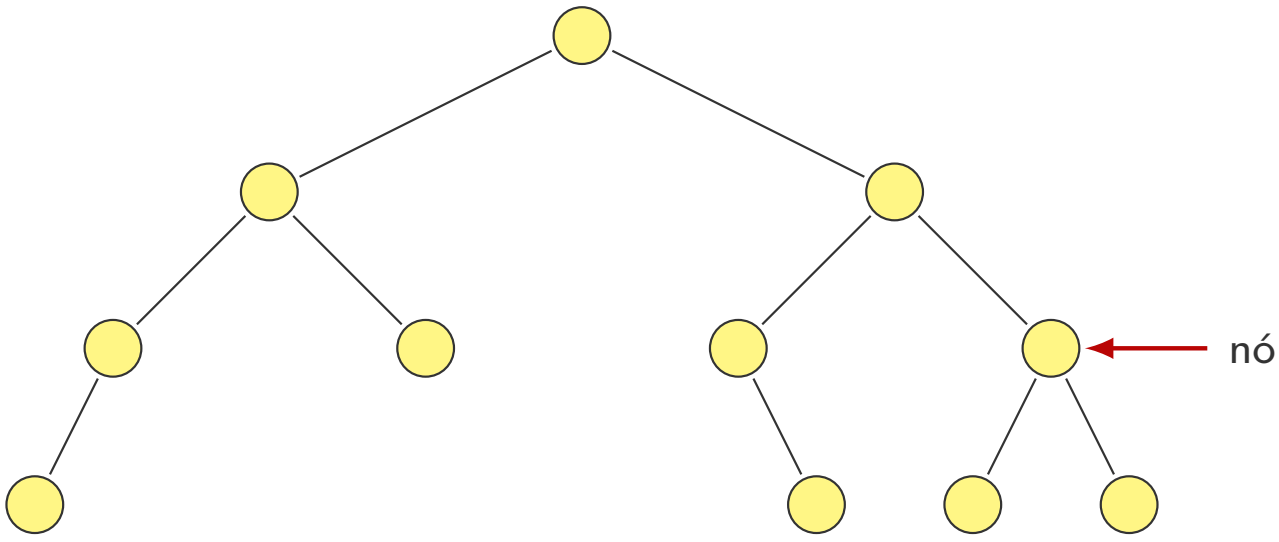


Uma árvore binária é:

- Ou o conjunto vazio
- Ou um nó conectado a no máximo duas árvores binárias.

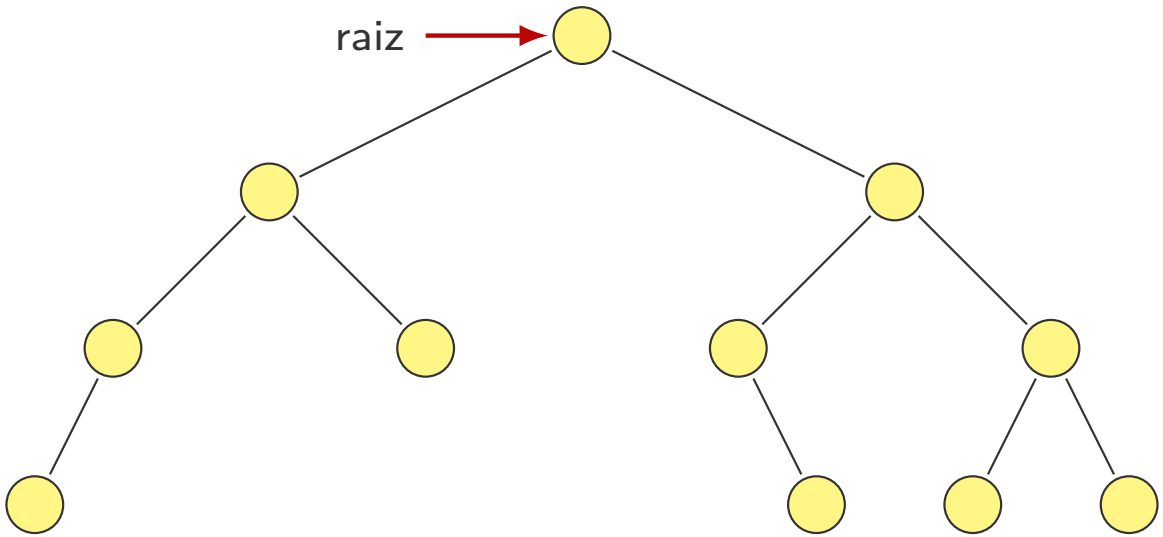
Árvores Binárias

Exemplo de árvore binária:



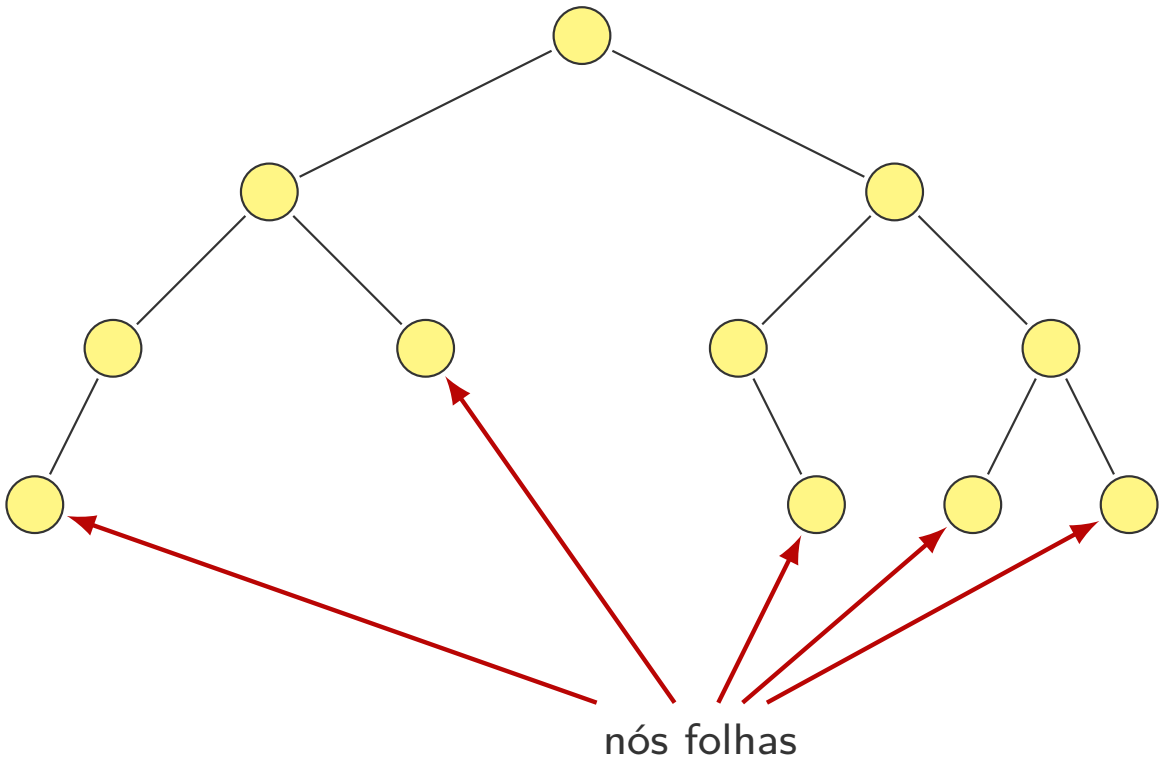
Árvores Binárias

Exemplo de árvore binária:



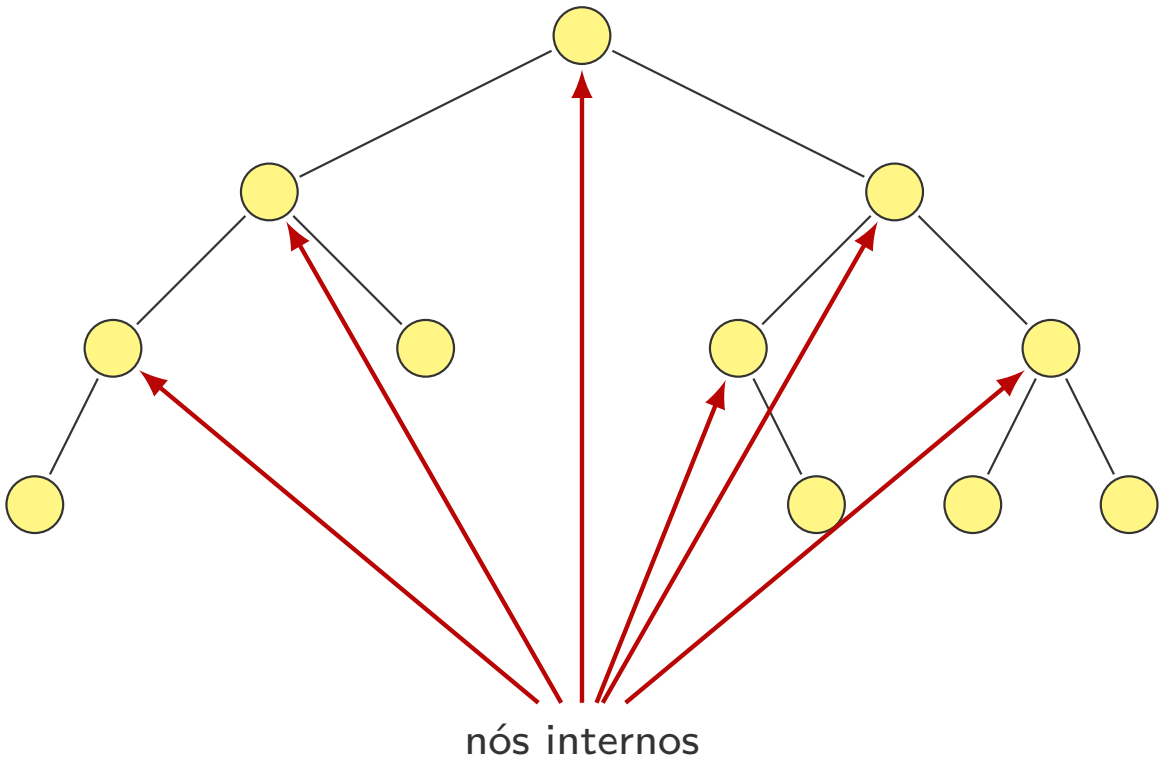
Árvores Binárias

Exemplo de árvore binária:



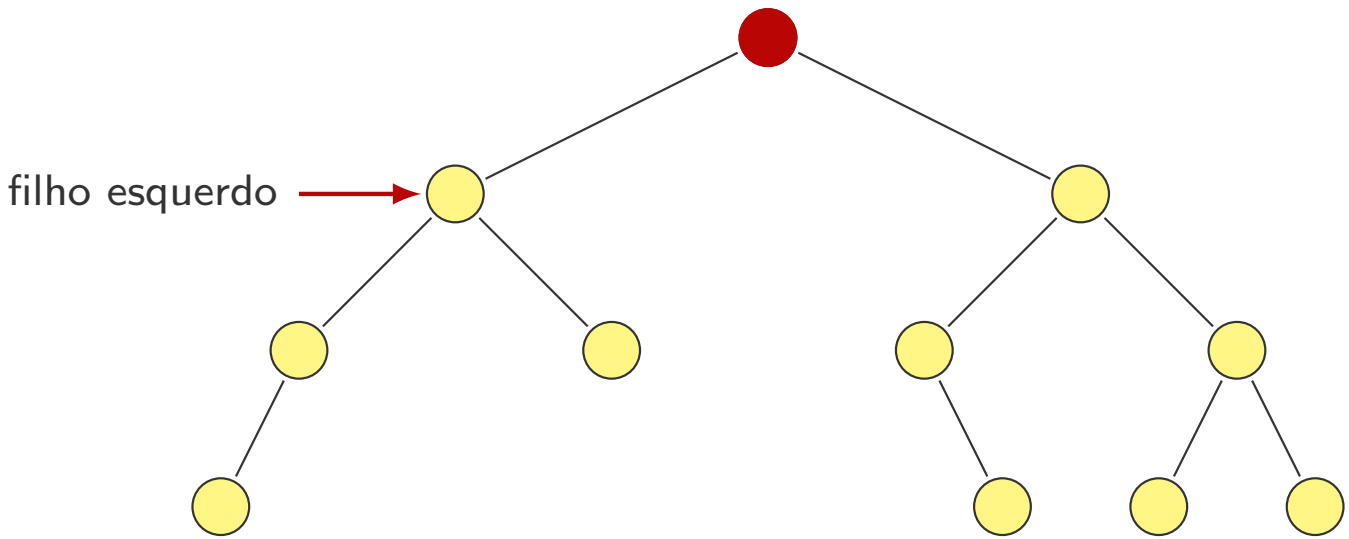
Árvores Binárias

Exemplo de árvore binária:



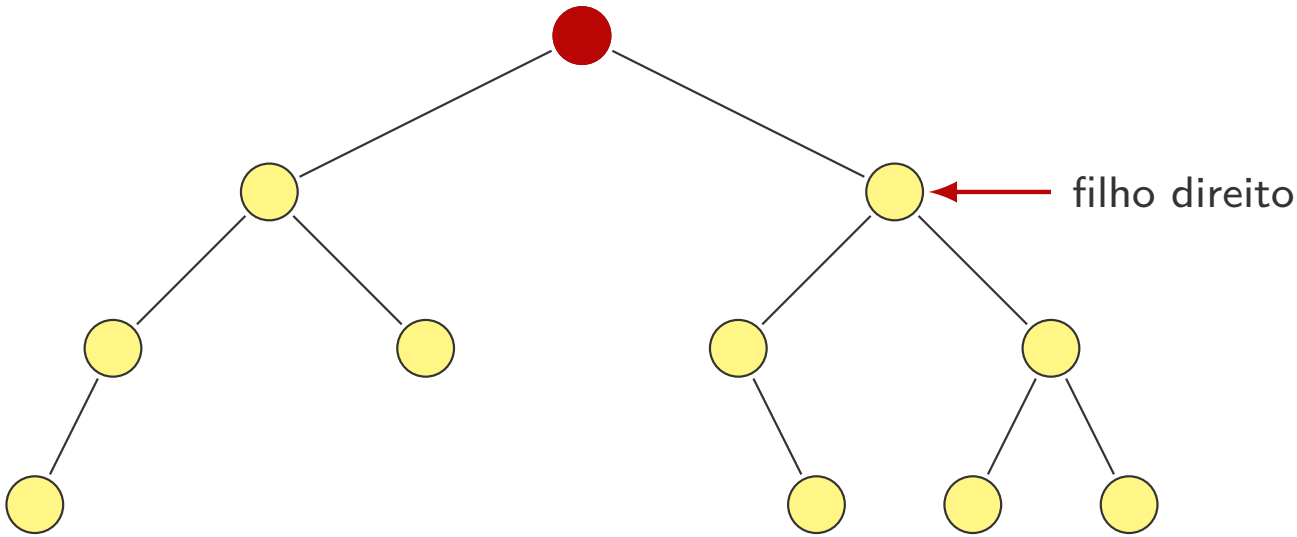
Árvores Binárias

Exemplo de árvore binária:



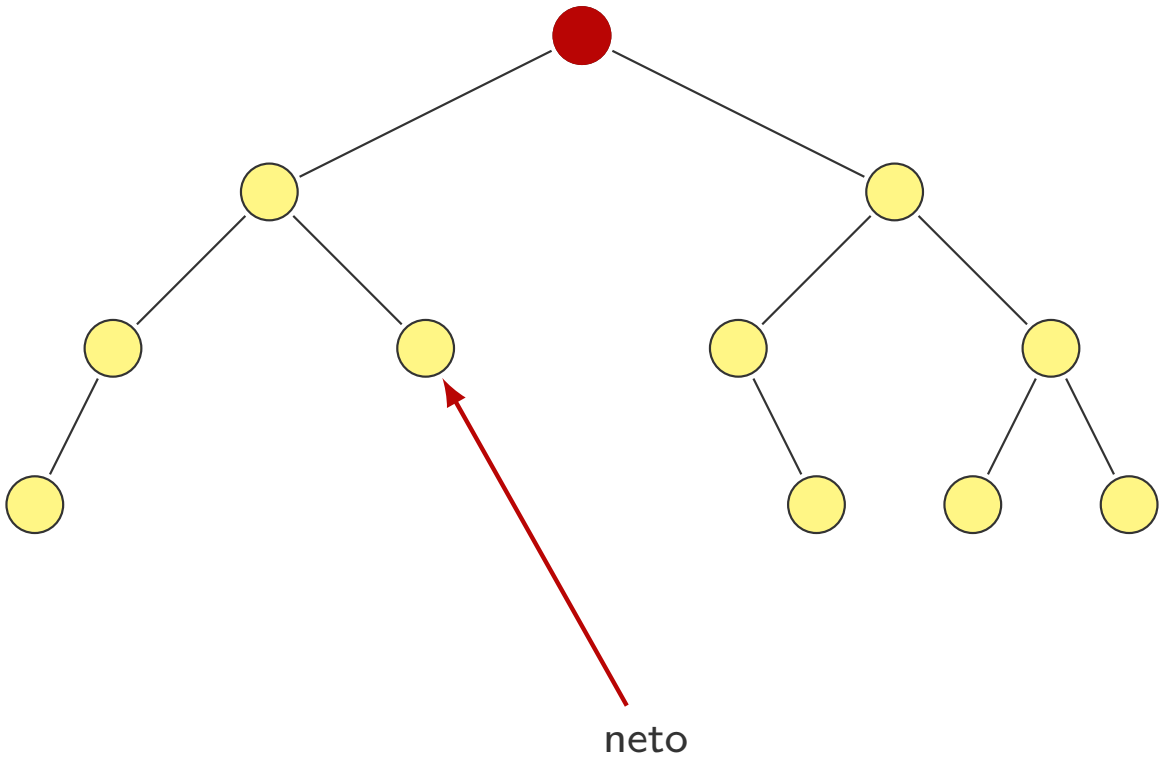
Árvores Binárias

Exemplo de árvore binária:



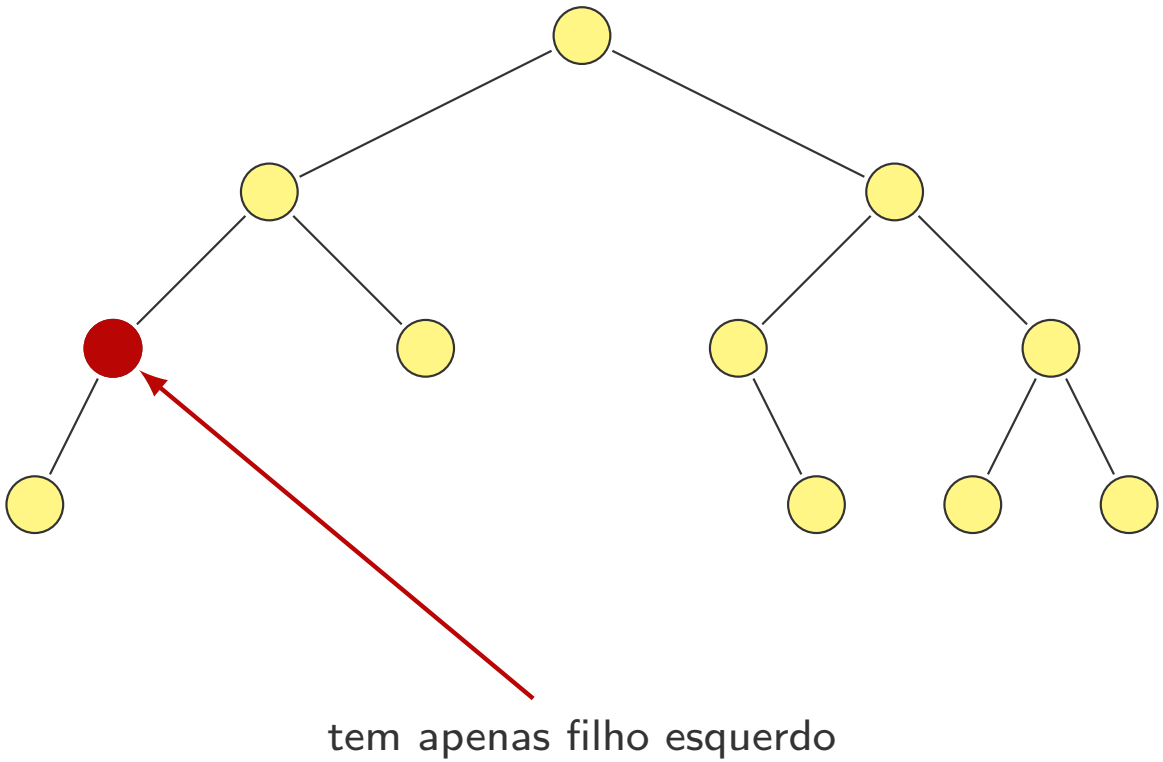
Árvores Binárias

Exemplo de árvore binária:



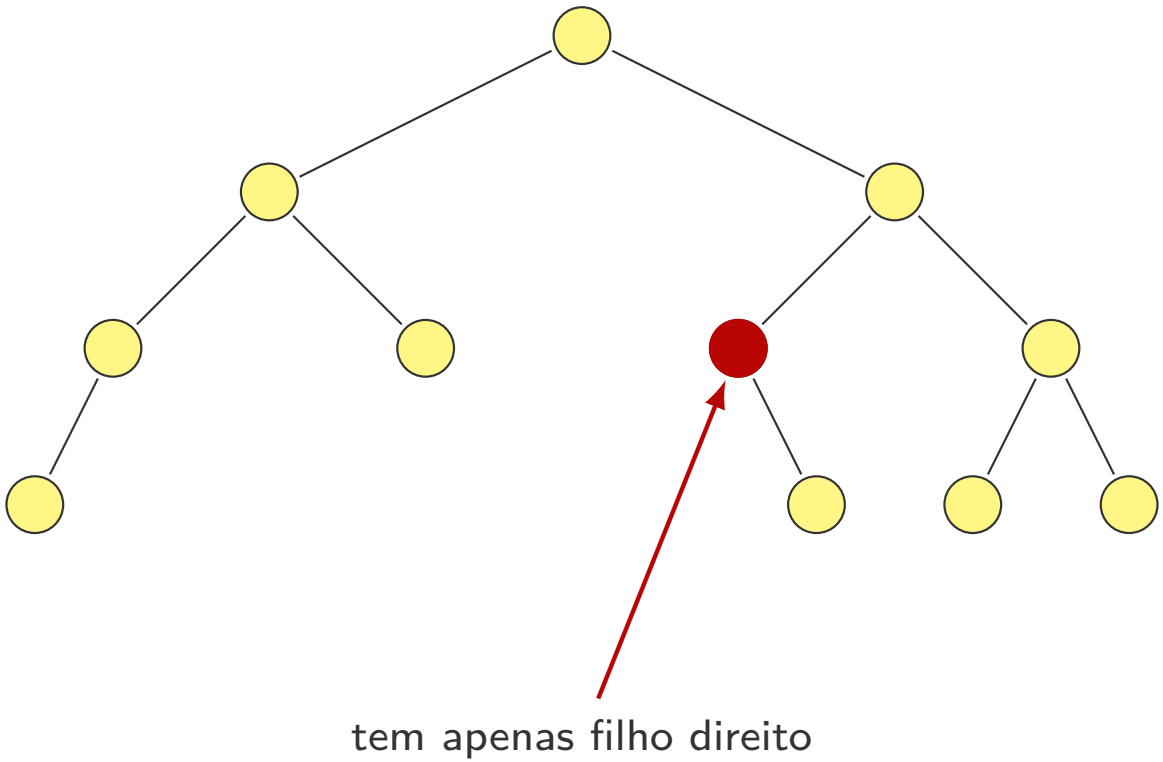
Árvores Binárias

Exemplo de árvore binária:



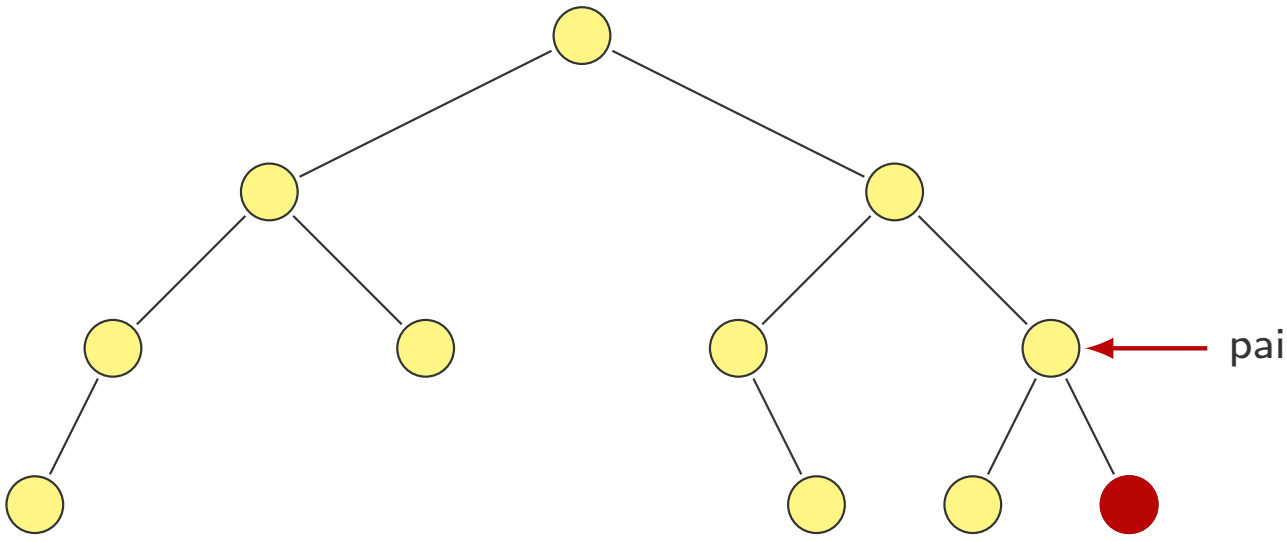
Árvores Binárias

Exemplo de árvore binária:



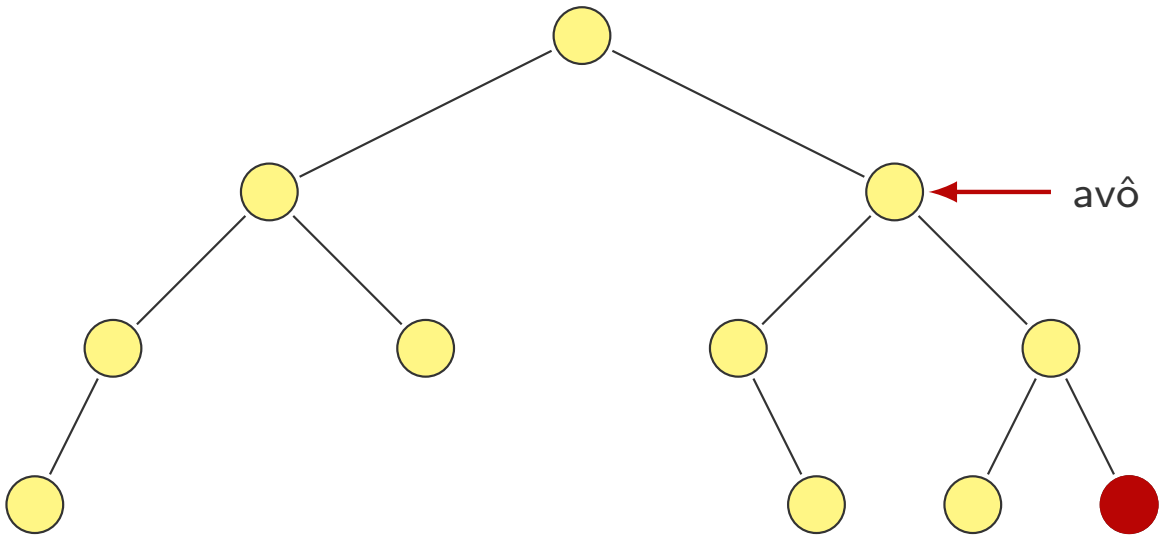
Árvores Binárias

Exemplo de árvore binária:



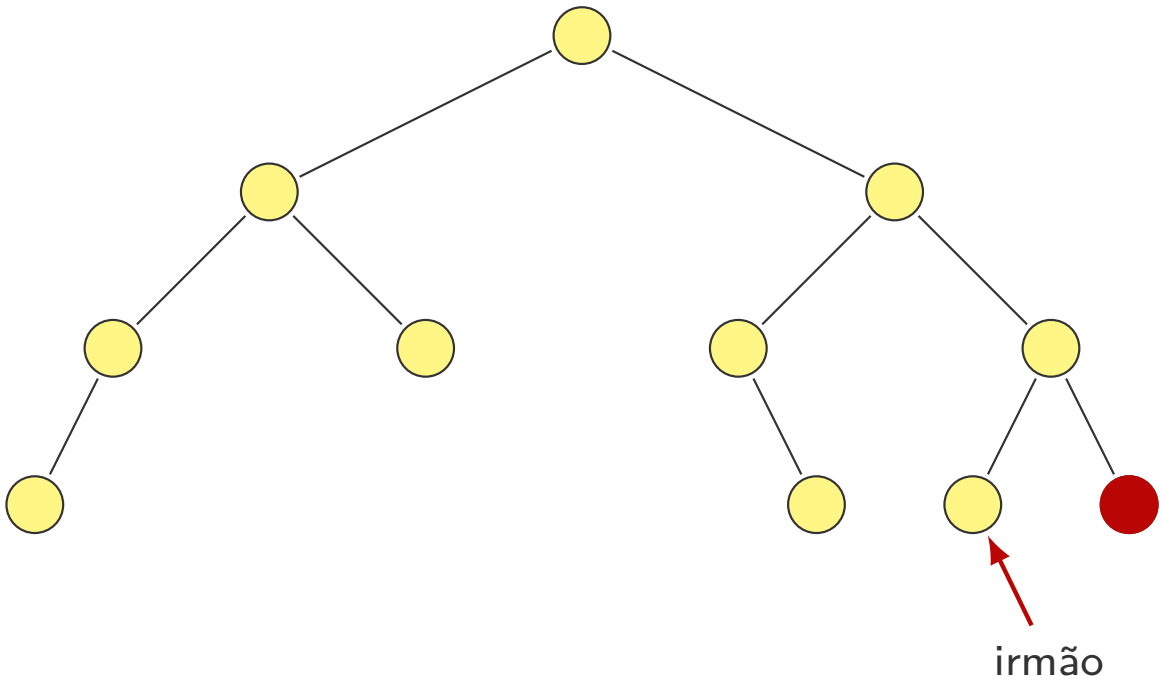
Árvores Binárias

Exemplo de árvore binária:



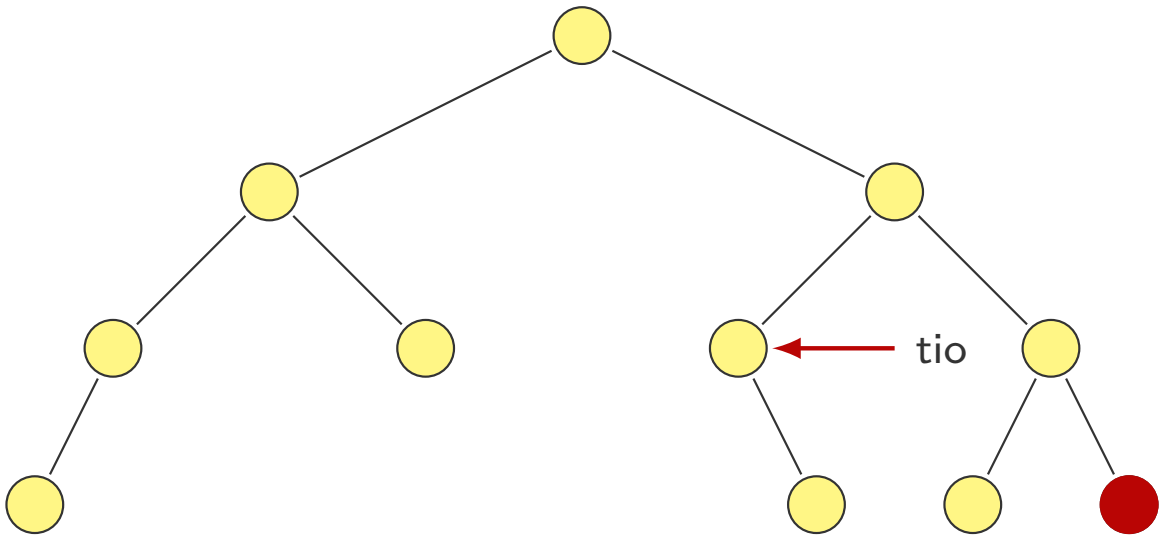
Árvores Binárias

Exemplo de árvore binária:



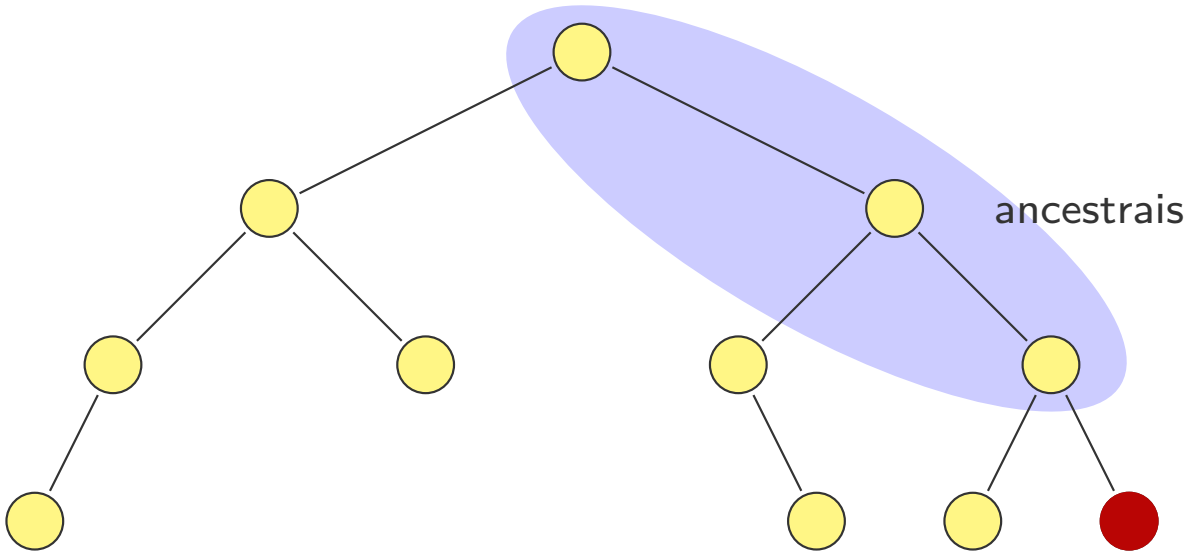
Árvores Binárias

Exemplo de árvore binária:



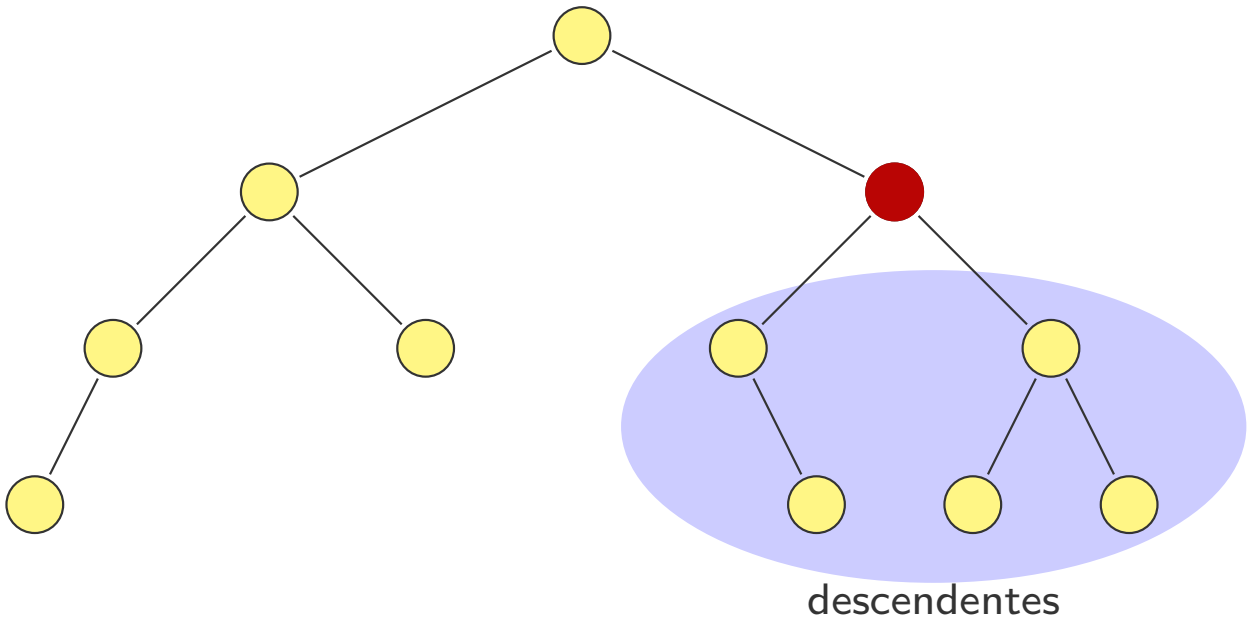
Árvores Binárias

Exemplo de árvore binária:



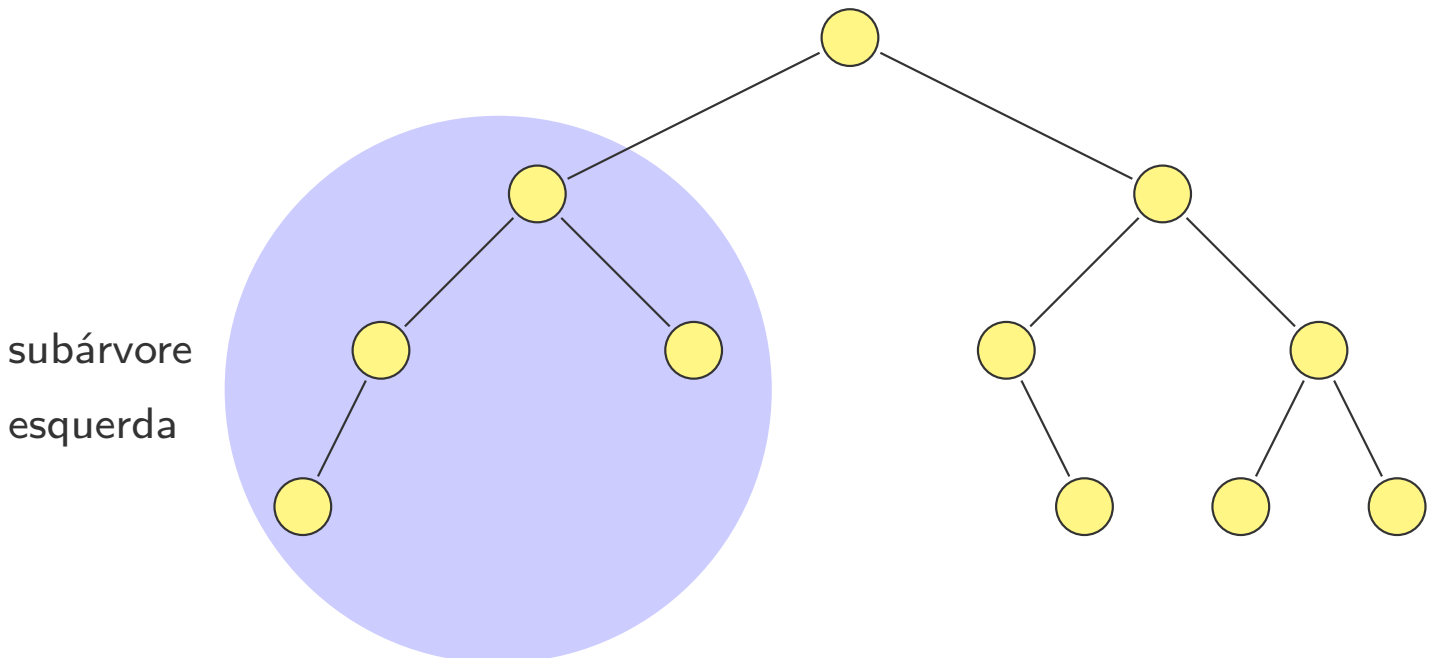
Árvores Binárias

Exemplo de árvore binária:



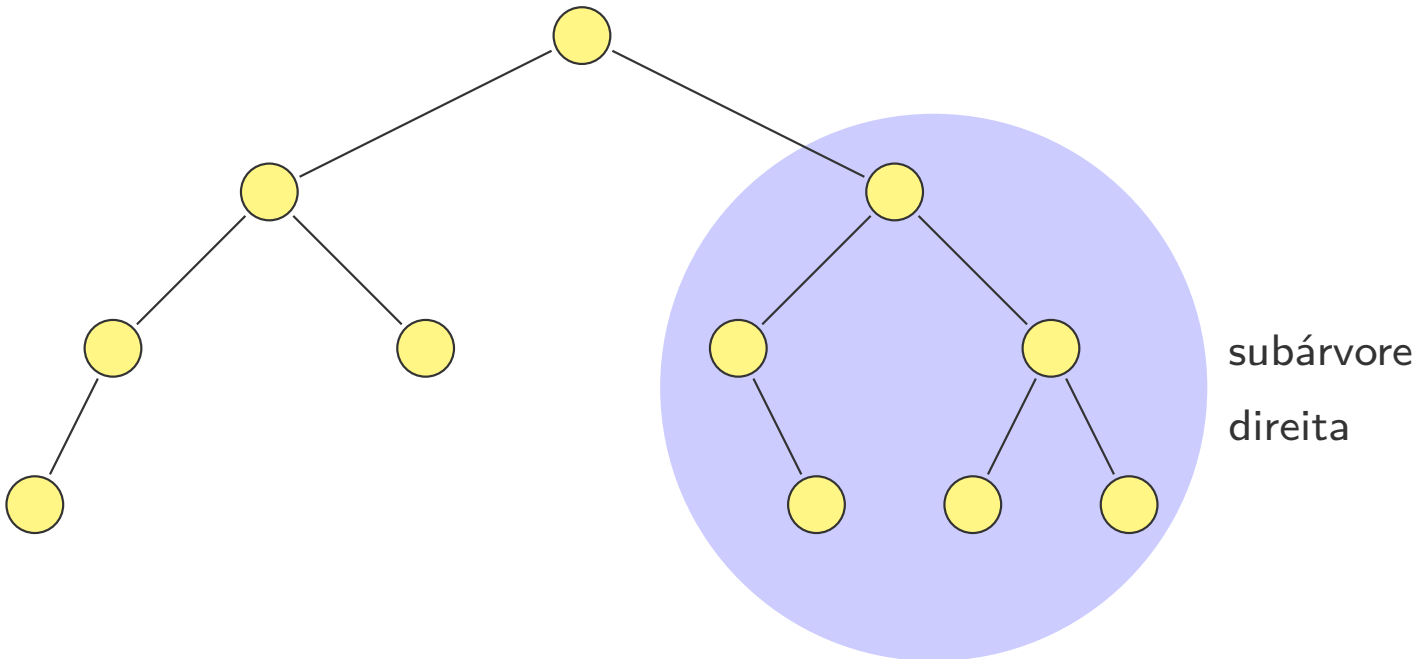
Árvores Binárias

Exemplo de árvore binária:



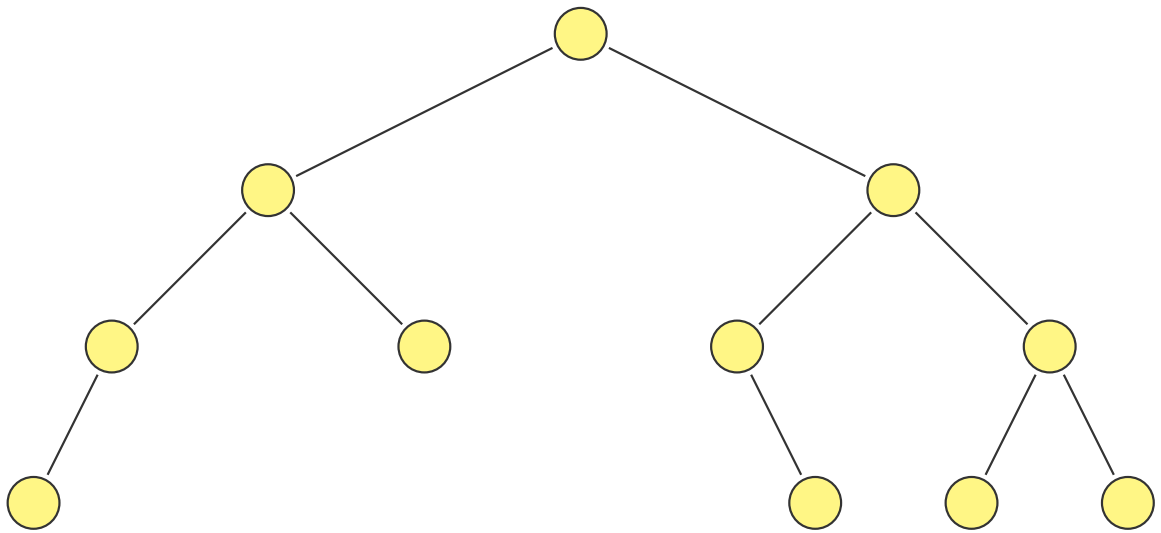
Árvores Binárias

Exemplo de árvore binária:



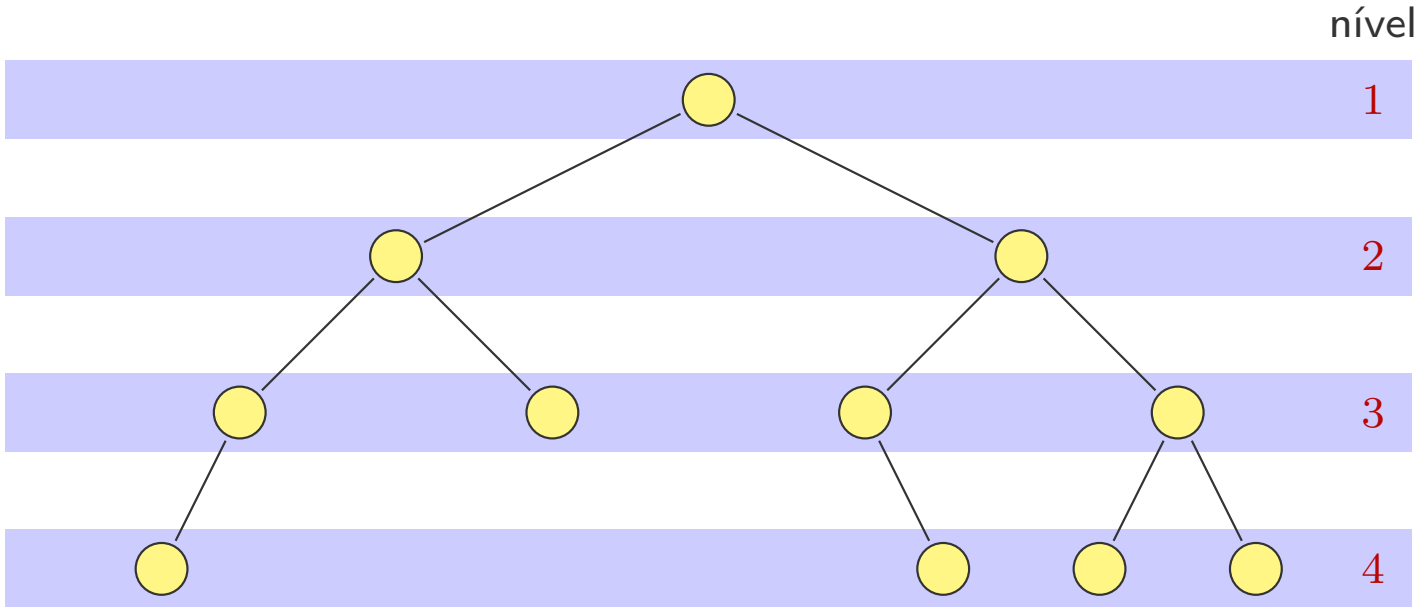
Árvores Binárias — Nível e Altura

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Árvores Binárias — Nível e Altura

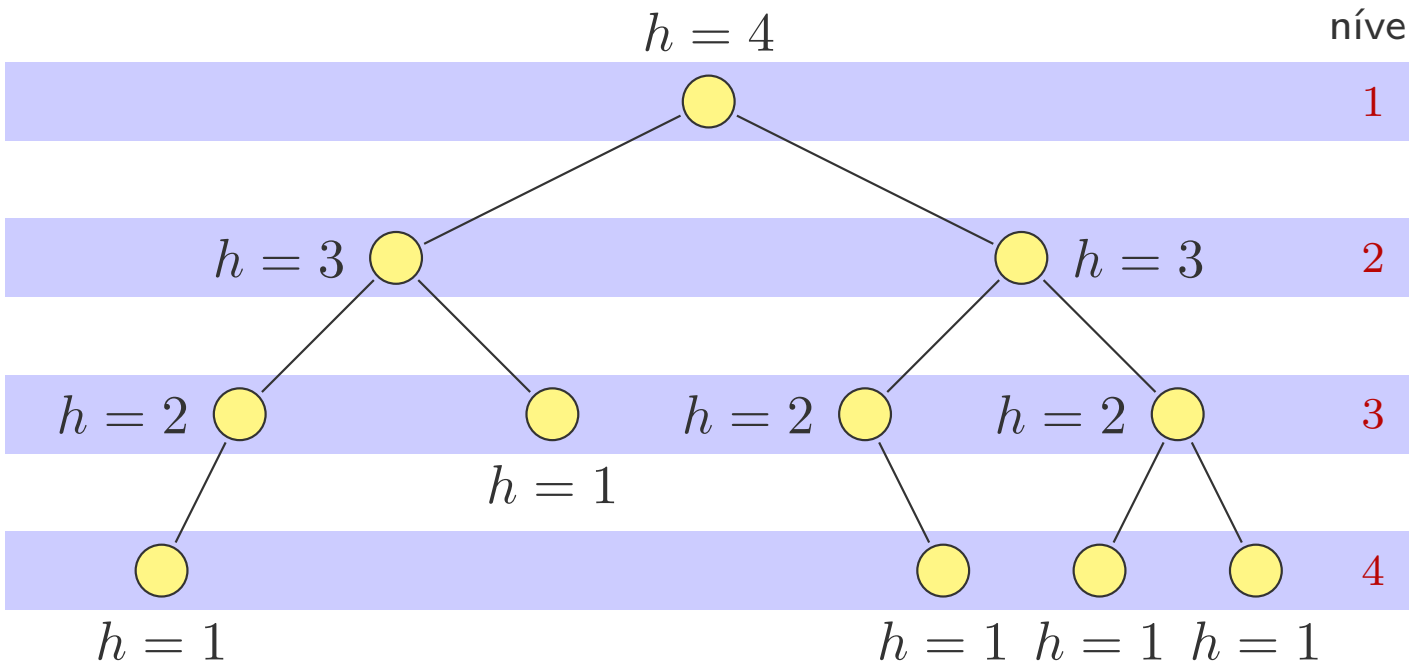
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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 — Nível e Altura

Universidade Federal do Ceará
Campus Quixadá

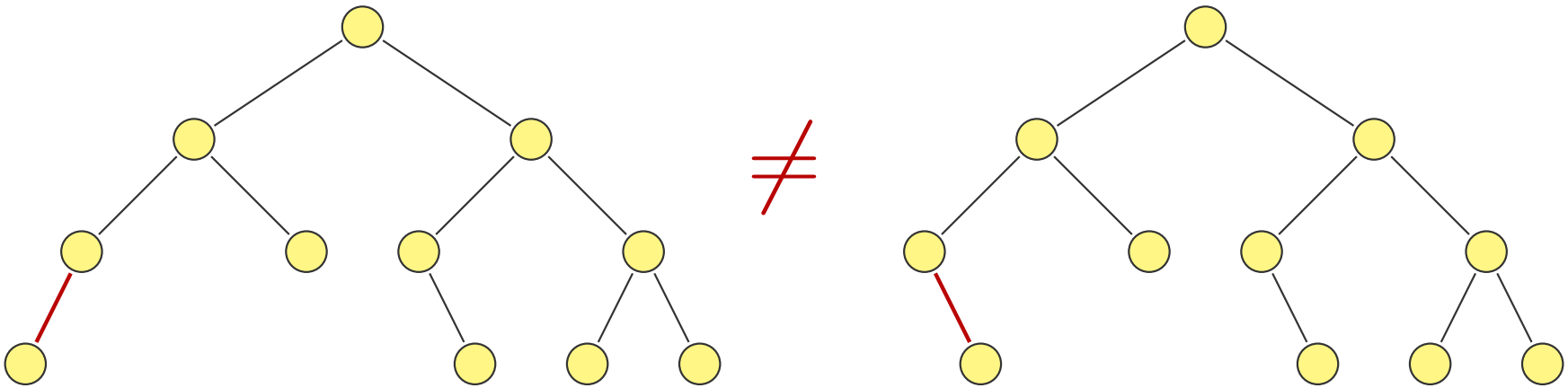


Profundidade de um nó v: Número de nós no caminho de v até a raiz. Dizemos que todos os nós com profundidade i estão no nível i.

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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.

Tipos específicos de árvores binárias

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- 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.

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

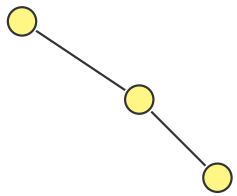
Relação entre altura e número de nós

Se a altura é h, então a árvore:

Relação entre altura e número de nós

Se a altura é h, então a árvore:

- tem no mínimo h nós

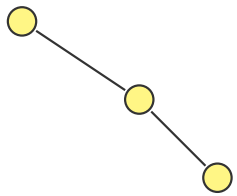


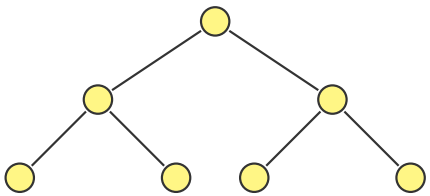
Relação entre altura e número de nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Se a altura é h, então a árvore:

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

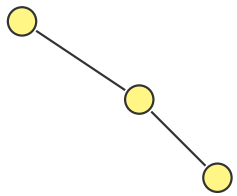


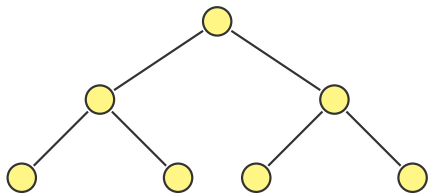


Relação entre altura e número de nós

Se a altura é h, então a árvore:

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





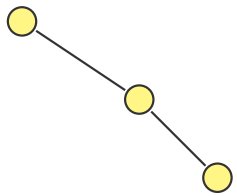
Se a árvore tem $n \geq 1$ nós, então:

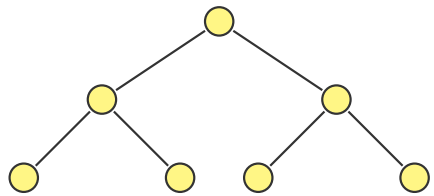
Relação entre altura e número de nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Se a altura é h, então a árvore:

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





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

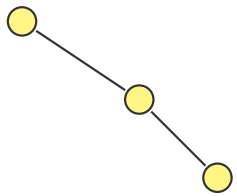
- a altura é no mínimo $\lceil \lg(n+1) \rceil$

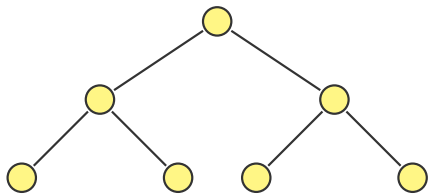
Relação entre altura e número de nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Se a altura é h, então a árvore:

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





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

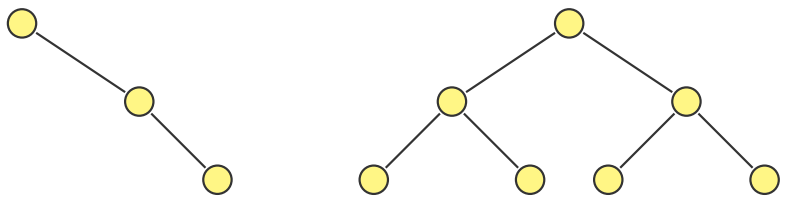
- a altura é no mínimo $\lceil \lg(n+1) \rceil$
  - quando a árvore é completa

Relação entre altura e número de nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Se a altura é h, então a árvore:

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



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

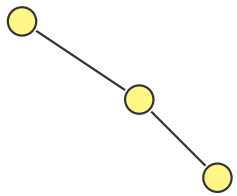
- a altura é no mínimo $\lceil \lg(n+1) \rceil$
  - quando a árvore é completa
- a altura é no máximo n

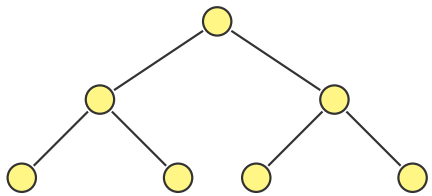
Relação entre altura e número de nós

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Se a altura é h, então a árvore:

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





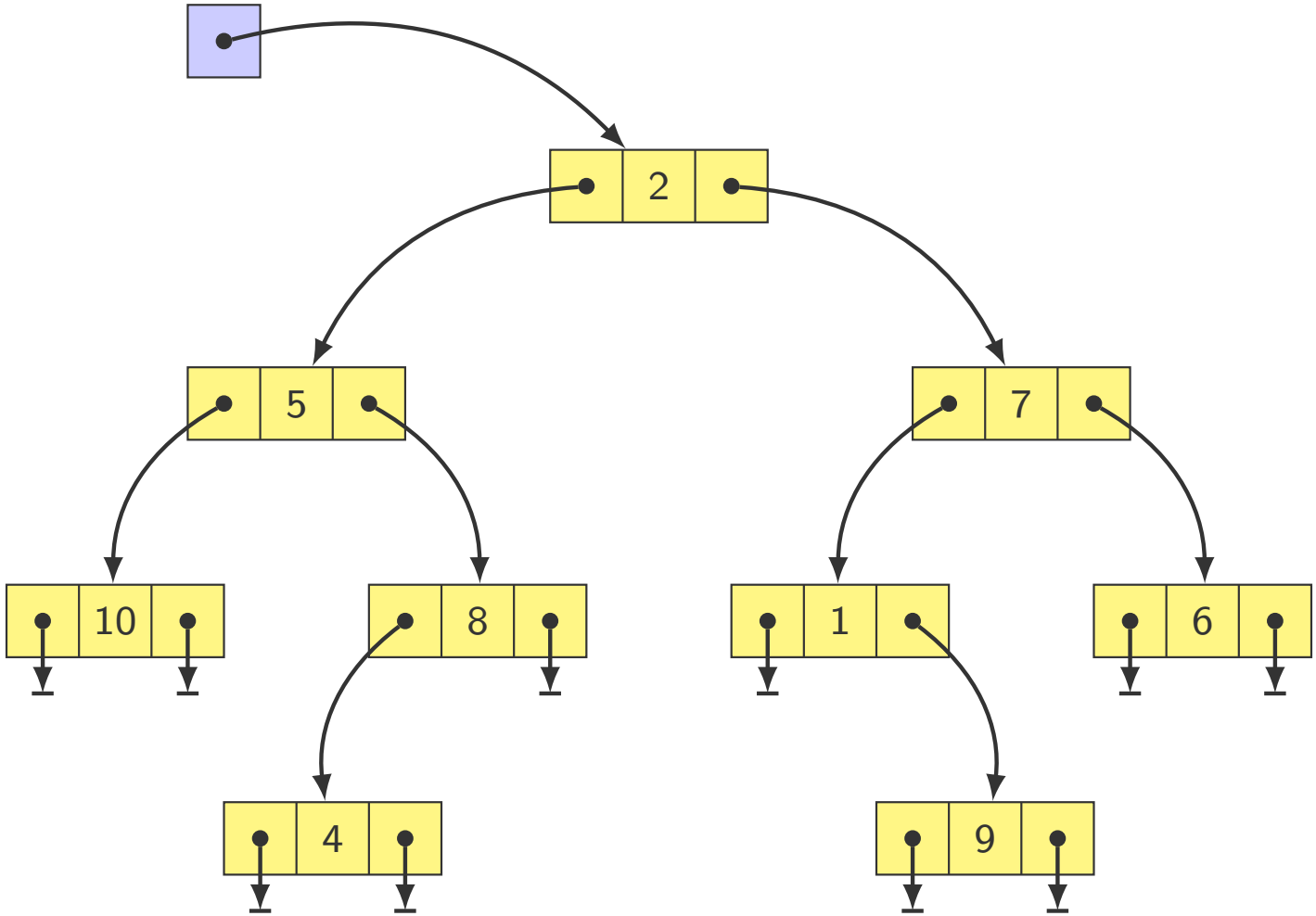
Se a árvore tem $n \geq 1$ nós, então:

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

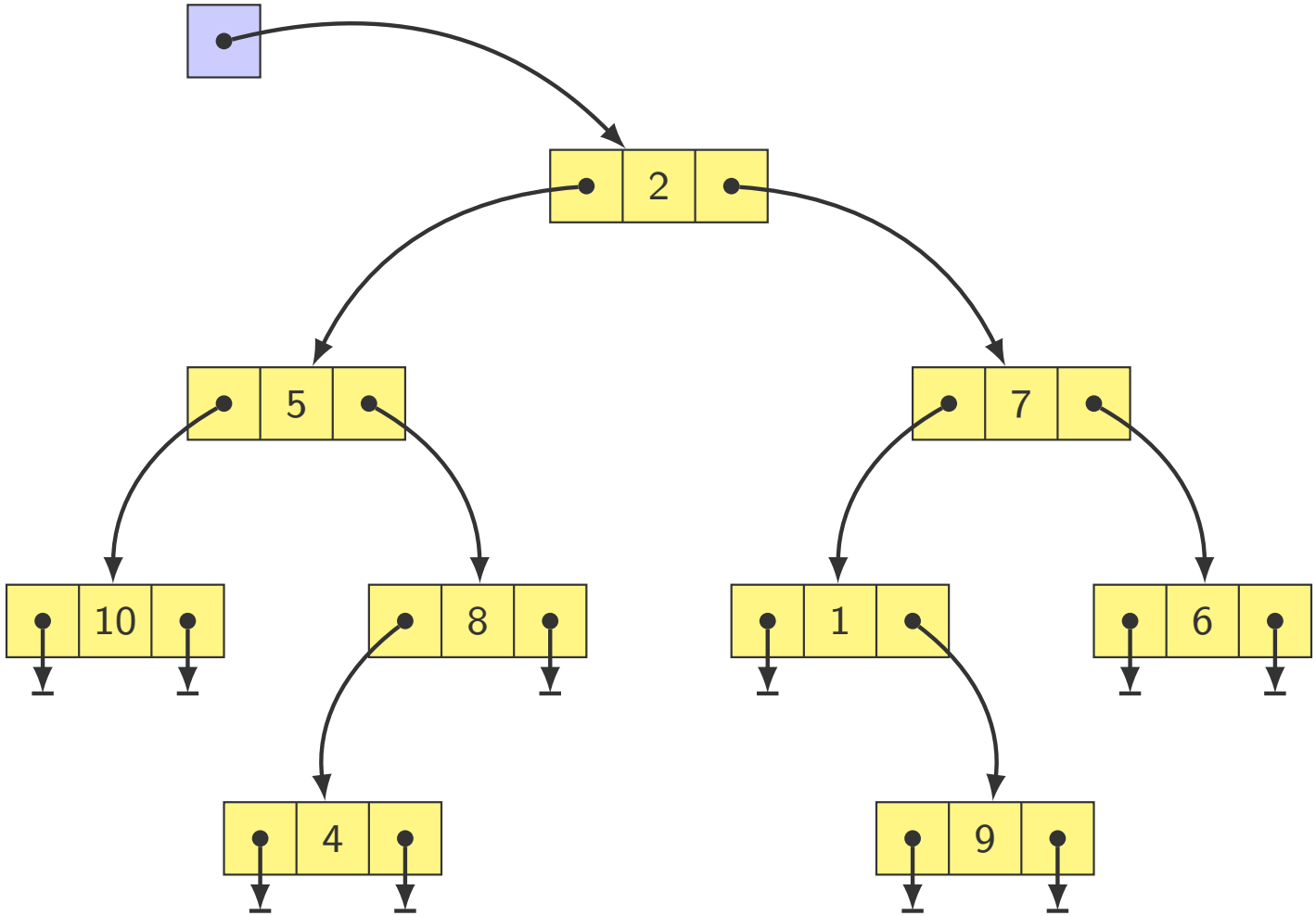
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Implementação

Implementação



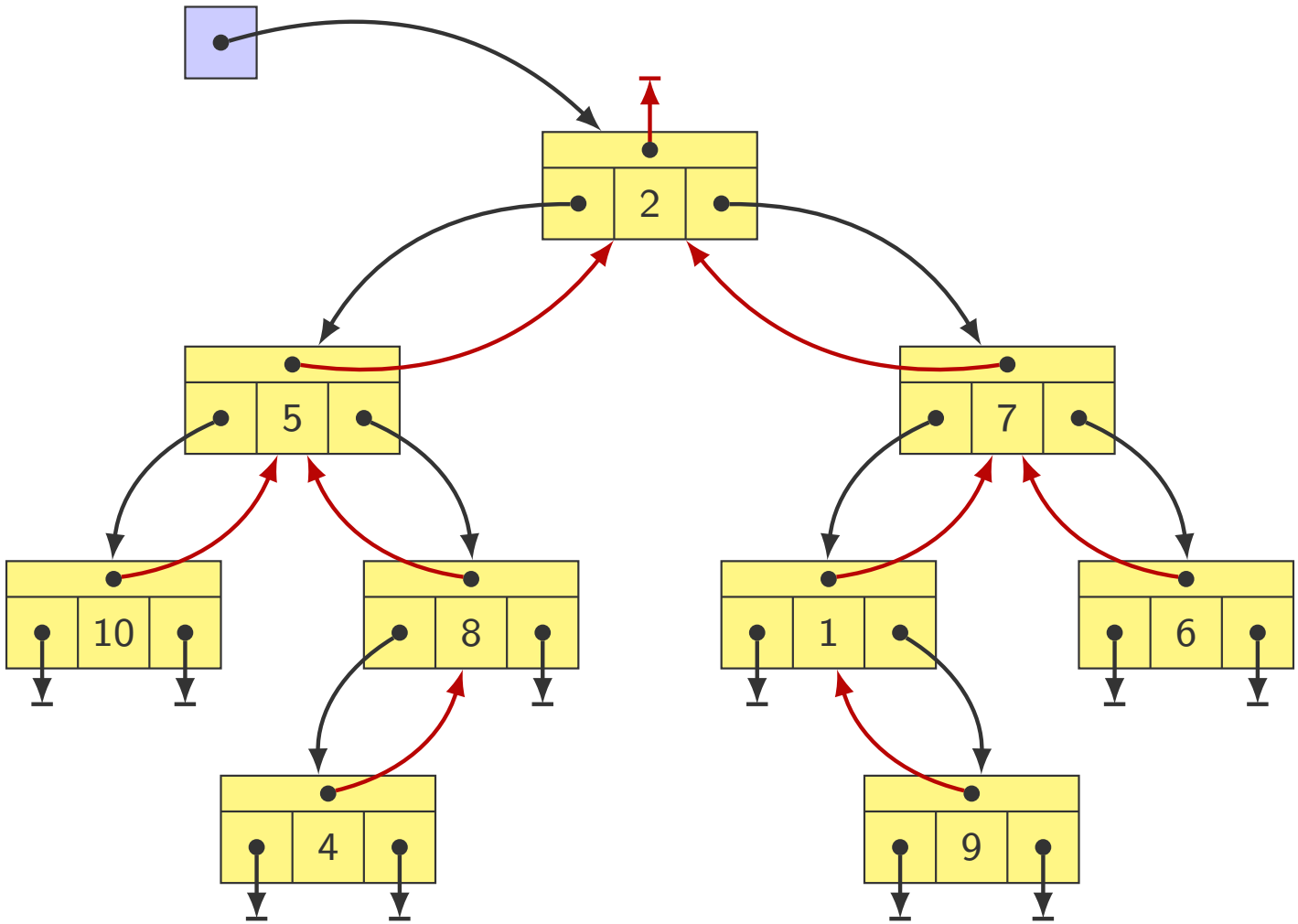
Implementação



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 — Decisões de projeto

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

Implementação — Decisões de projeto

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

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

Implementação — Decisões de projeto

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

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

Implementação — Decisões de projeto

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

- 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

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

- 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

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

- 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

```
1 struct Node { // sem ponteiro para no pai
2     int key;
3     Node *left; // subarvore esquerda
4     Node *right; // subarvore direita
5 };
```

Implementação — Decisões de projeto

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

- 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

```
1 struct Node { // sem ponteiro para no pai
2     int key;
3     Node *left; // subarvore esquerda
4     Node *right; // subarvore direita
5 };
```

- 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.

BinaryTree.h — Declaração dos protótipos das funções e dos novos tipos


```
1  #ifndef ARVOREBIN_H
2  #define ARVOREBIN_H
3
4  struct Node; // Cada no eh do tipo Node
5
6  Node* bt_emptyTree(); // retorna nulo, indicando arvore vazia
7
8  // cria no com chave 'key'
9  Node* bt_create(int key, Node* l, Node* r);
10
11 // arvore enraizada em no esta vazia?
12 bool bt_empty(Node* node);
13
14 void bt_print(Node* node); // imprime as chaves da arvore
15
16 bool bt_contains(Node* node, int key); // essa chave pertence?
17
18 Node* bt_destroy(Node* node); // libera todos os nos alocados
19
20 #endif
```

BinaryTree.cpp — Implementação das funções

Declaração do `struct Node`:

```
1  #include <iostream>
2  #include "BinaryTree.h"
3  using std::cout;
4  using std::endl;
5
6  struct Node { // sem ponteiro para no pai
7      int key;
8      Node *left; // subarvore esquerda
9      Node *right; // subarvore direita
10 };
```

BinaryTree.cpp — Implementação das funções

Cria uma árvore vazia:

```
1 Node* bt_emptyTree() {
2     return nullptr;
3 }
```

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Cria uma árvore vazia:

```
1 Node* bt_emptyTree() {
2     return nullptr;
3 }
```

Cria um nó com certa chave e dois filhos:

```
1 Node* bt_create(int key, Node* l, Node* r) {
2     Node* novo = new Node{};
3     novo->key = key;
4     novo->left = l;
5     novo->right = r;
6     return novo;
7 }
```

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Cria uma árvore vazia:

```
1 Node* bt_emptyTree() {
2     return nullptr;
3 }
```

Cria um nó com certa chave e dois filhos:

```
1 Node* bt_create(int key, Node* l, Node* r) {
2     Node* novo = new Node{};
3     novo->key = key;
4     novo->left = l;
5     novo->right = r;
6     return novo;
7 }
```

Árvores são estruturas definidas recursivamente

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

`BinaryTree.cpp` — Implementação das funções

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Saber se a árvore é vazia:

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 bool bt_empty(Node* node) {
2     return (node == nullptr);
3 }
```

BinaryTree.cpp — Implementação das funções

Saber se a árvore é vazia:

```
1 bool bt_empty(Node* node) {
2     return (node == nullptr);
3 }
```

Percorrendo e imprimindo a árvore:

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 bool bt_empty(Node* node) {
2     return (node == nullptr);
3 }
```

Percorrendo e imprimindo a árvore:

```
1 void bt_print(Node* node) {
2     if ( ! bt_empty(node) ) {
3         cout << node->key << endl;
4         bt_print(node->left);
5         bt_print(node->right);
6     }
7 }
```

`BinaryTree.cpp` — Implementação das funções

Buscando uma chave na árvore:

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

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

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

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

Observações:

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

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

Observações:

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

BinaryTree.cpp — Implementação das funções

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

```
1 bool bt_contains(Node* node, int key) {
2     if (node == nullptr)
3         return false; // Sub-arvore vazia
4     else
5         return node->key == key ||
6                bt_contains(node->left, key) ||
7                bt_contains(node->right, key);
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 das funções

Universidade Federal do Ceará
Campus Quixadá

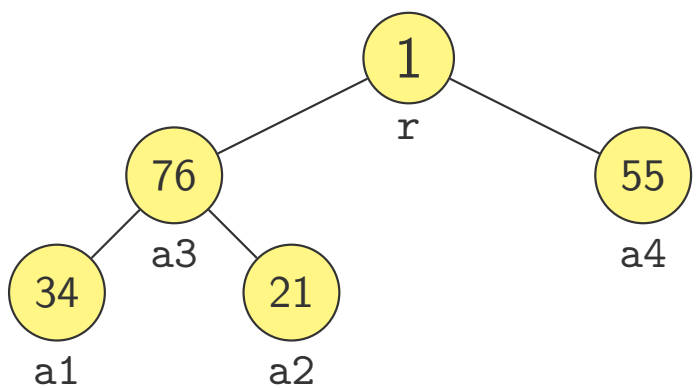
Liberando memória alocada para a árvore:

```
1
2  Node* bt_destroy(Node* node) {
3      if (node != nullptr) {
4          node->left = bt_destroy(node->left);
5          node->right = bt_destroy(node->right);
6          cout << "Deleting " << node->key << endl;
7          delete node;
8      }
9      return nullptr;
10 }
```

main.cpp — Exemplo de programa cliente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "BinaryTree.h"
3
4  // Cria arvore com 5 nos, imprime chaves e finaliza
5  // liberando a memoria que foi alocada para a arvore
6  int main() {
7      Node* a1 = bt_create(34, nullptr, nullptr);
8      Node* a2 = bt_create(21, nullptr, nullptr);
9      Node* a3 = bt_create(76, a1, a2);
10     Node* a4 = bt_create(55, nullptr, nullptr);
11     Node* raiz = bt_create(1, a3, a4);
12
13     bt_print(raiz);
14
15     raiz = bt_destroy(raiz);
16
17     return 0;
18 }
```



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