Árvores Binárias

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2021

Introdução

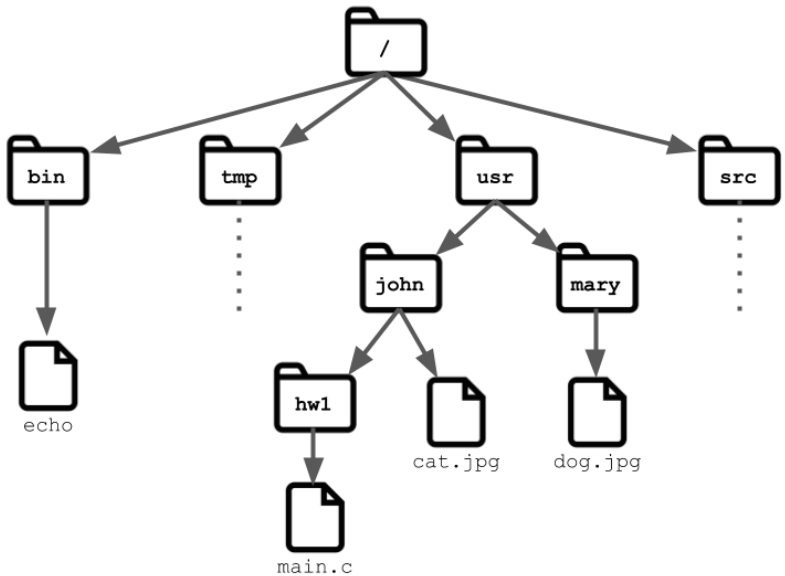
Representando uma hierarquia

- 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

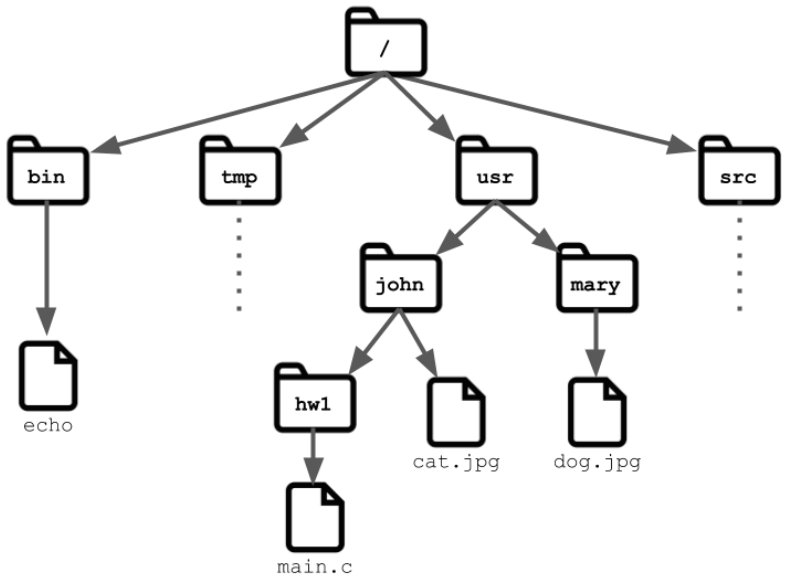
- Vetores e filas são estruturas lineares.
- A importância dessas estruturas é inegável, mas elas não são adequadas para representar dados dispostos de maneira hierárquica.



Hierarquia do sistema de arquivos de um PC Linux

Representando uma hierarquia

- Vetores e filas 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

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

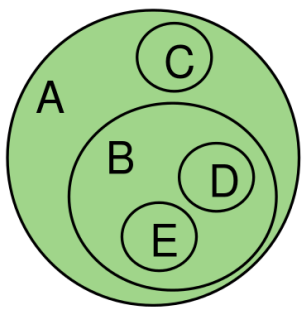
(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.

Árvore — Definição Recursiva

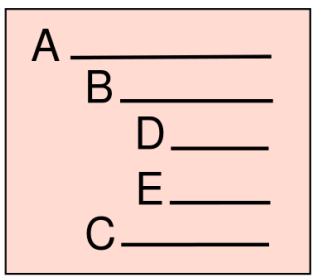
Uma árvore enraizada T, ou simplesmente árvore, é um conjunto finito de elementos denominados nós, tais que:

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

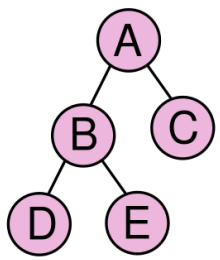
(b) $T \neq \emptyset$ e ele possui um nó especial r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em $m \geq 1$ conjuntos disjuntos não vazios, as subárvores de r, cada qual por sua vez um árvore.



c)



b)



a)

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

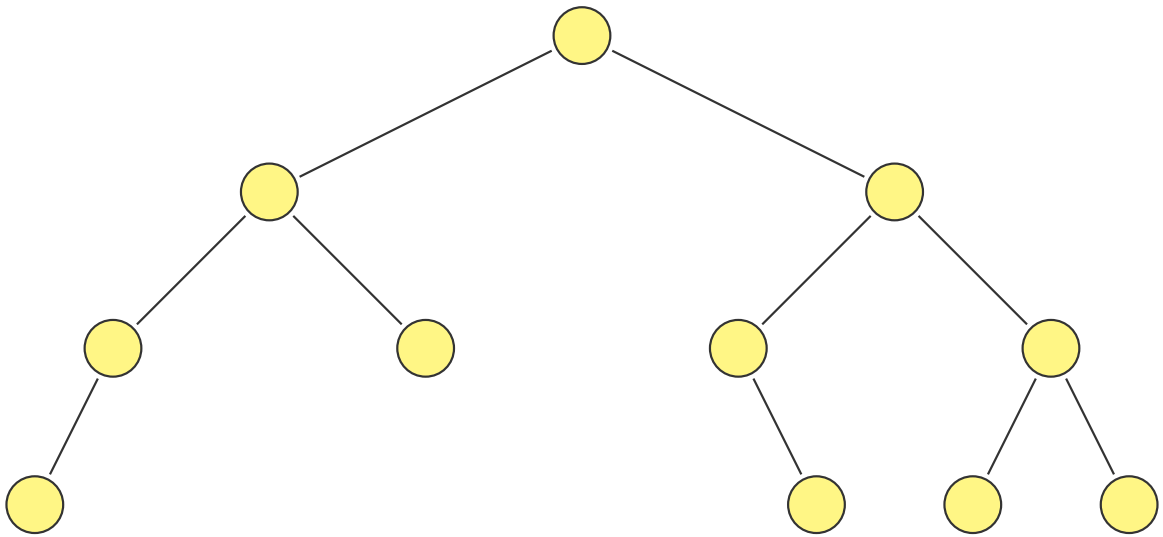
e)

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

d)

Árvores Binárias

Exemplo de árvore binária:

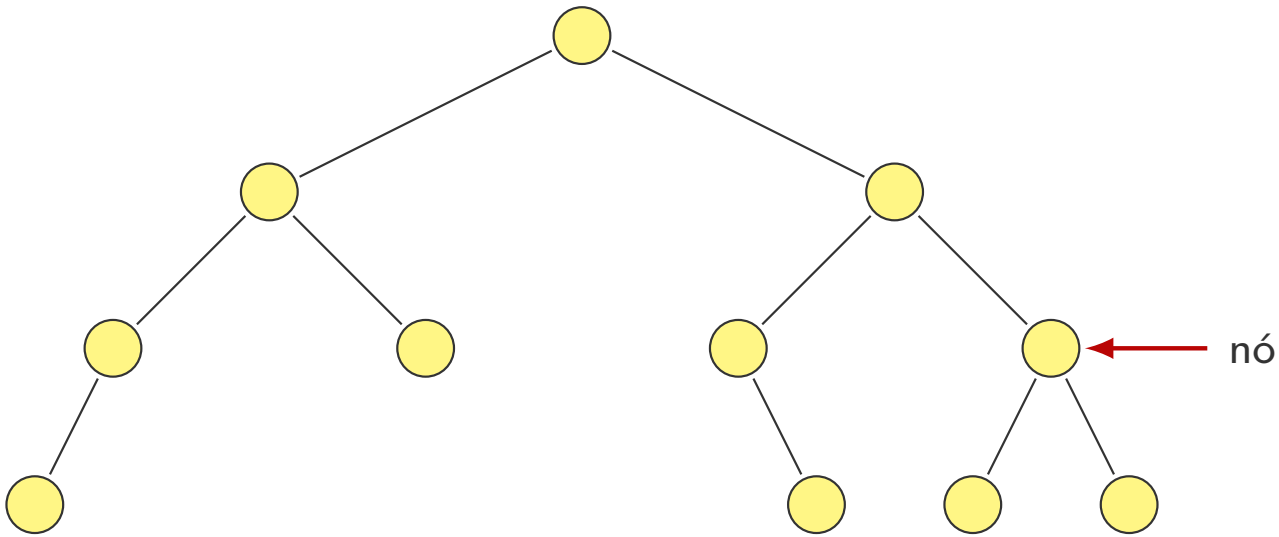


Uma árvore binária é:

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

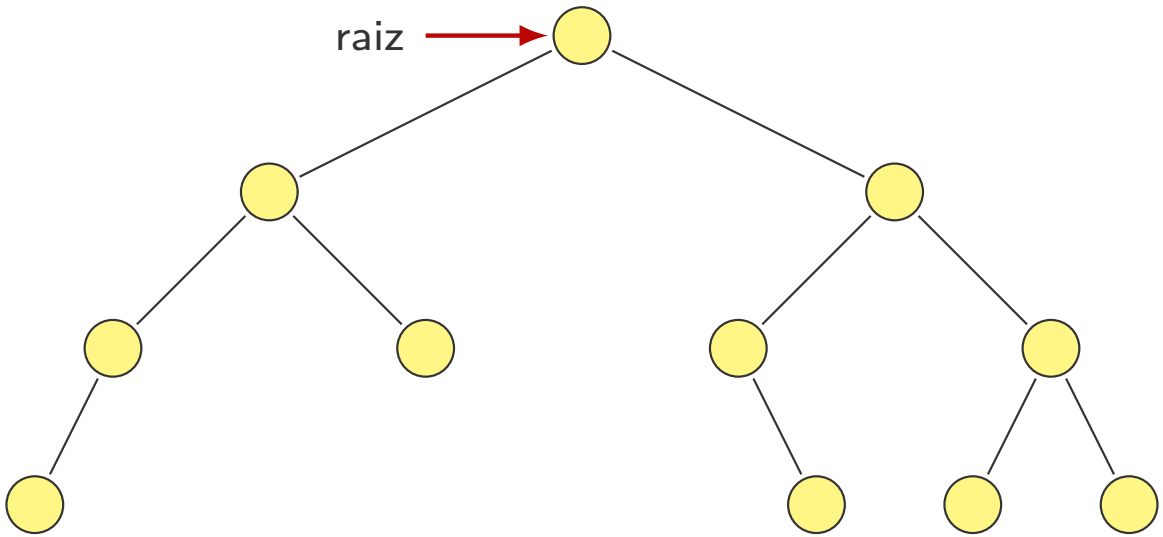
Árvores Binárias

Exemplo de árvore binária:



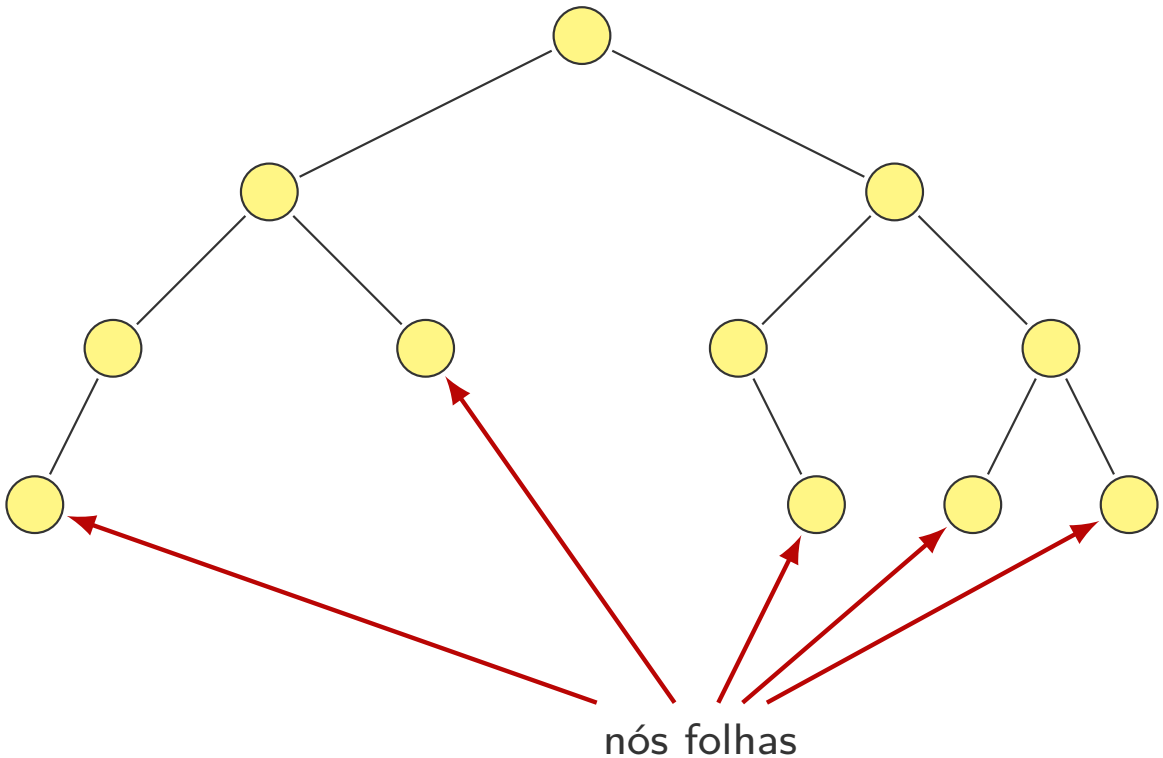
Árvores Binárias

Exemplo de árvore binária:



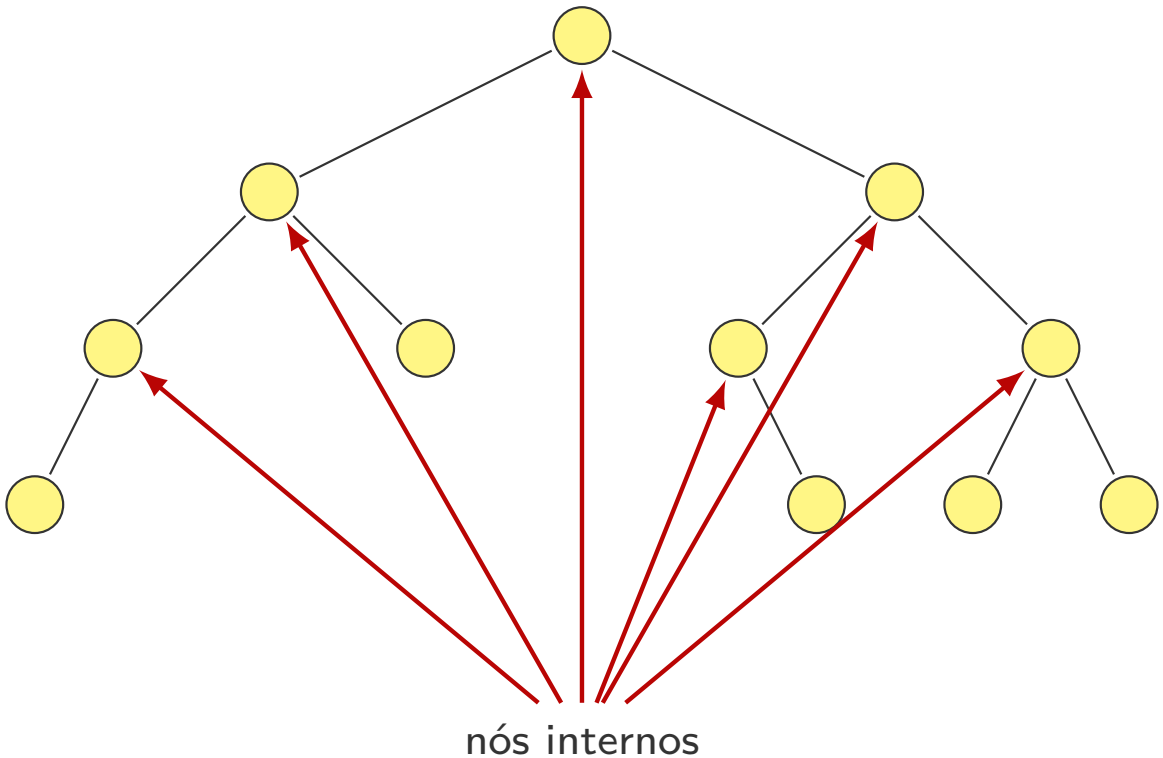
Árvores Binárias

Exemplo de árvore binária:



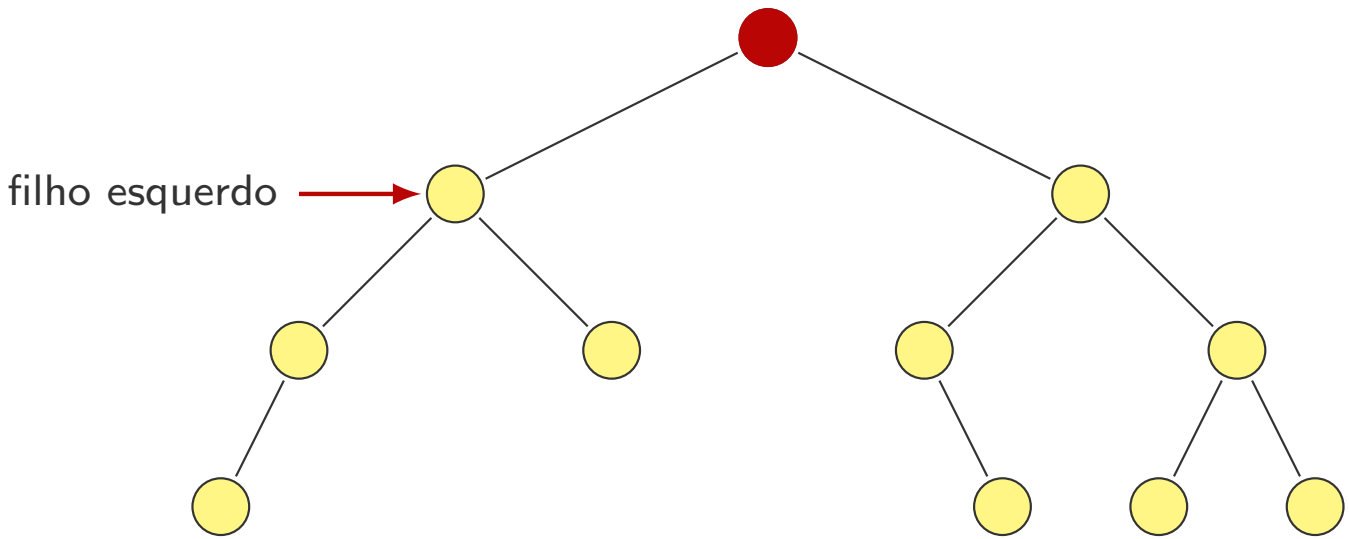
Árvores Binárias

Exemplo de árvore binária:



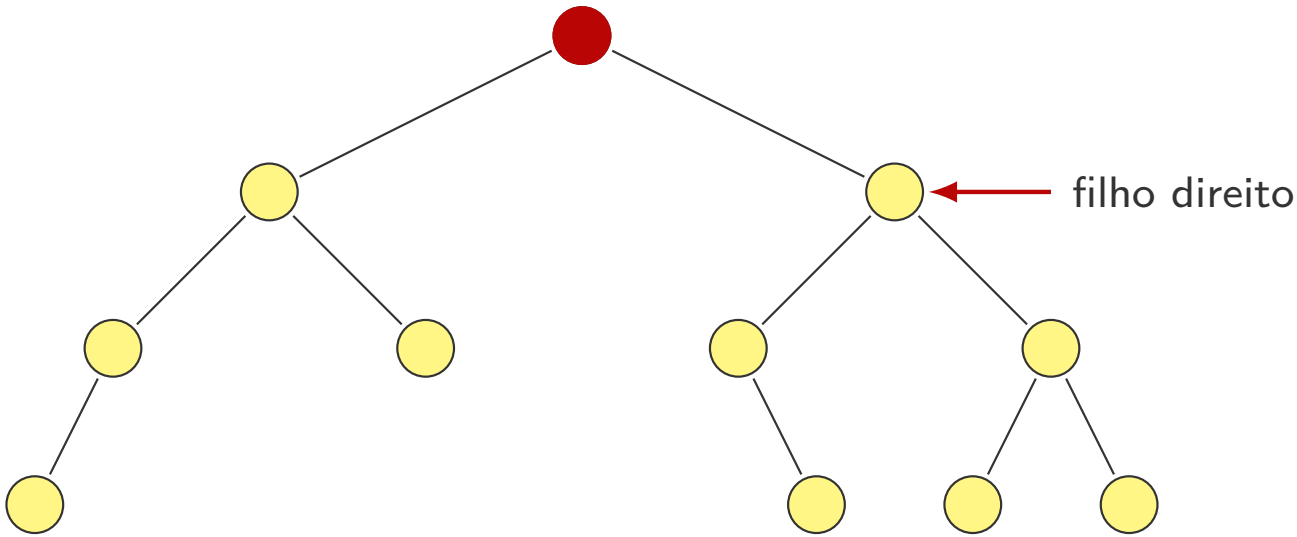
Árvores Binárias

Exemplo de árvore binária:



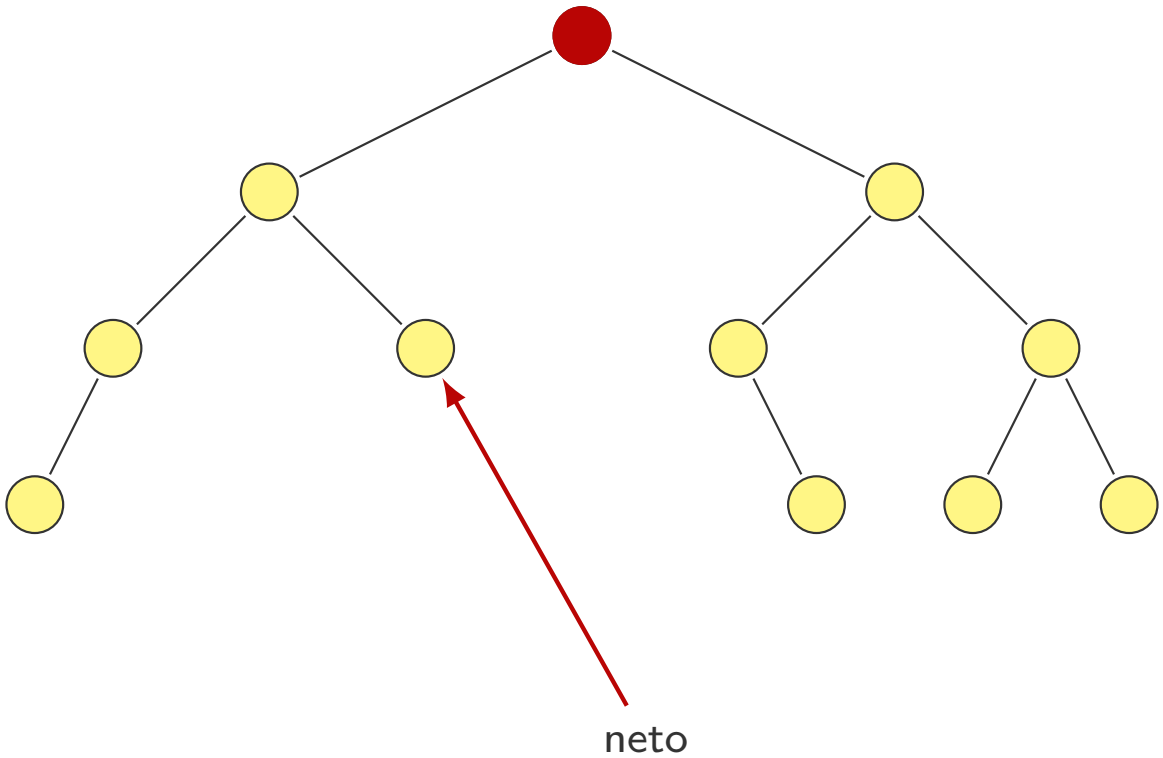
Árvores Binárias

Exemplo de árvore binária:



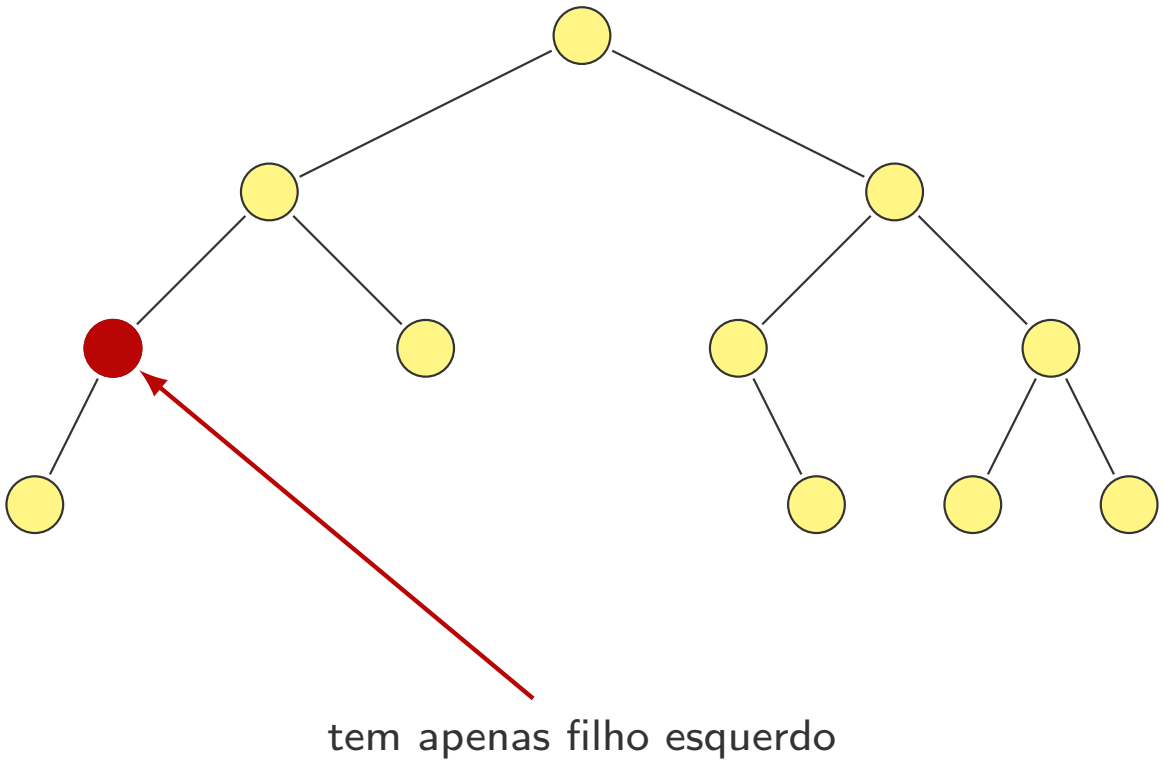
Árvores Binárias

Exemplo de árvore binária:



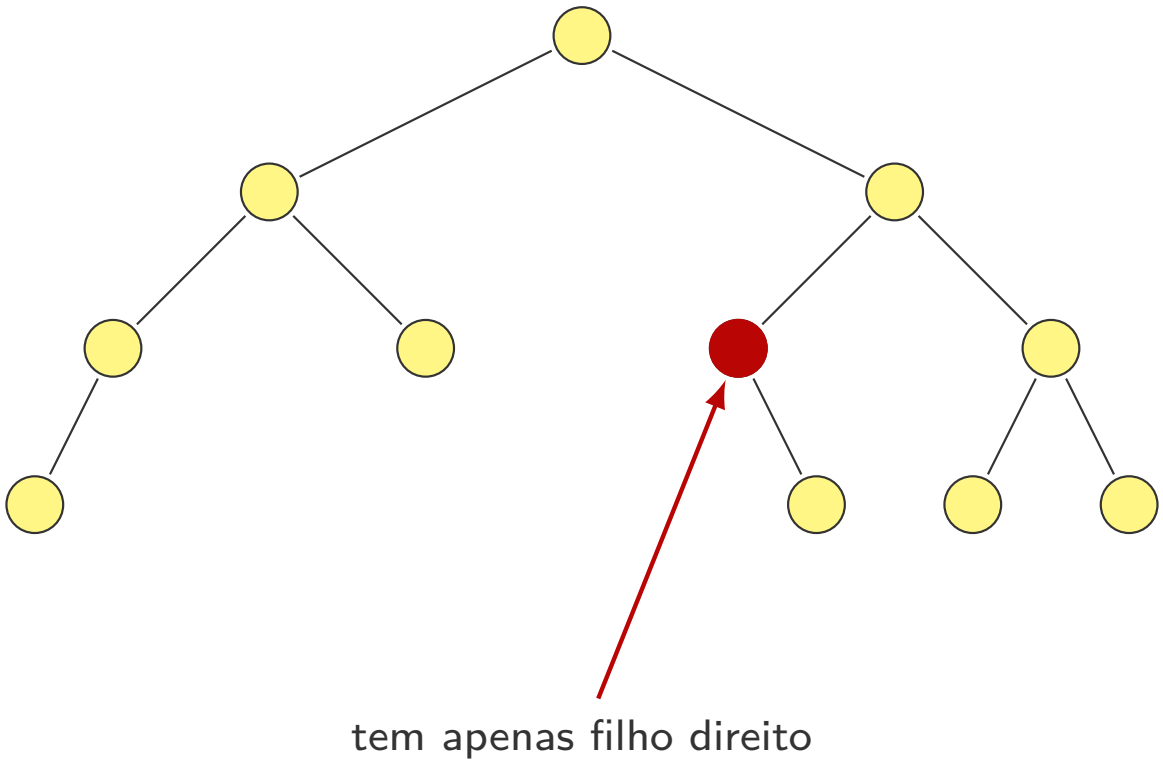
Árvores Binárias

Exemplo de árvore binária:



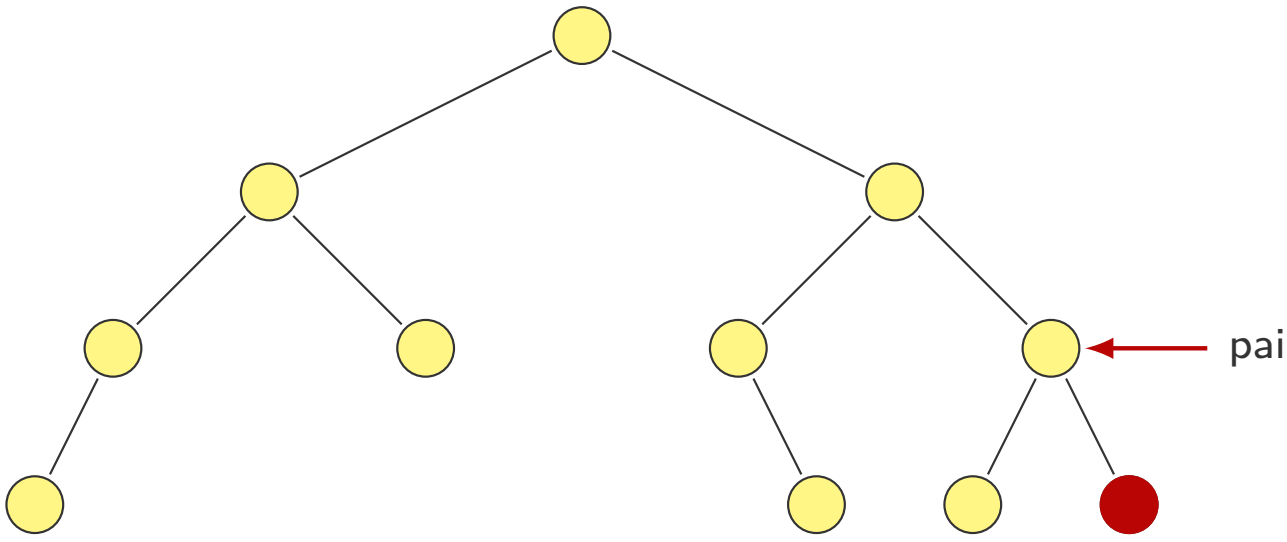
Árvores Binárias

Exemplo de árvore binária:



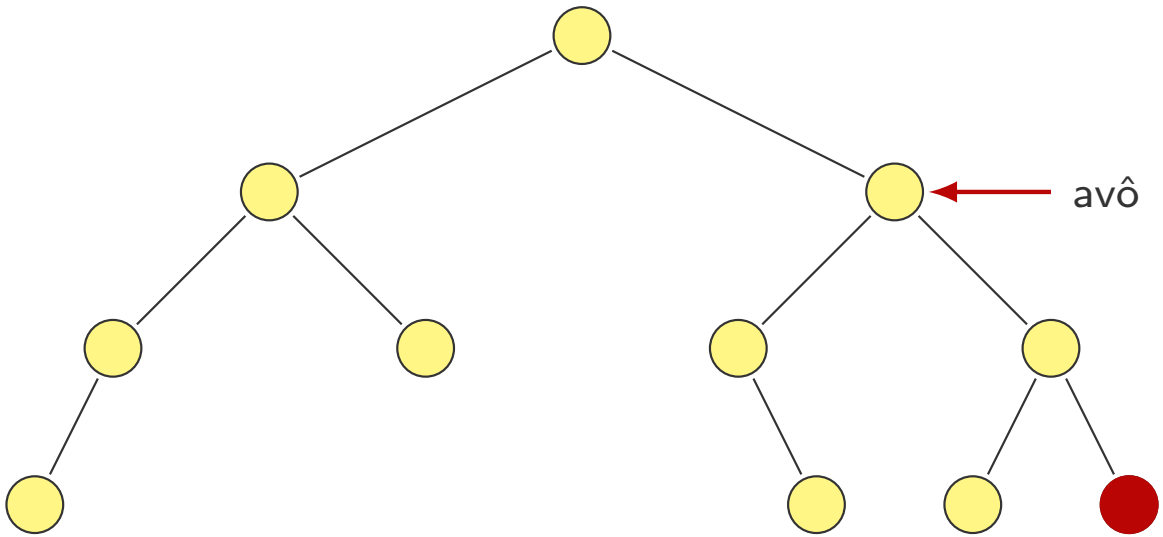
Árvores Binárias

Exemplo de árvore binária:



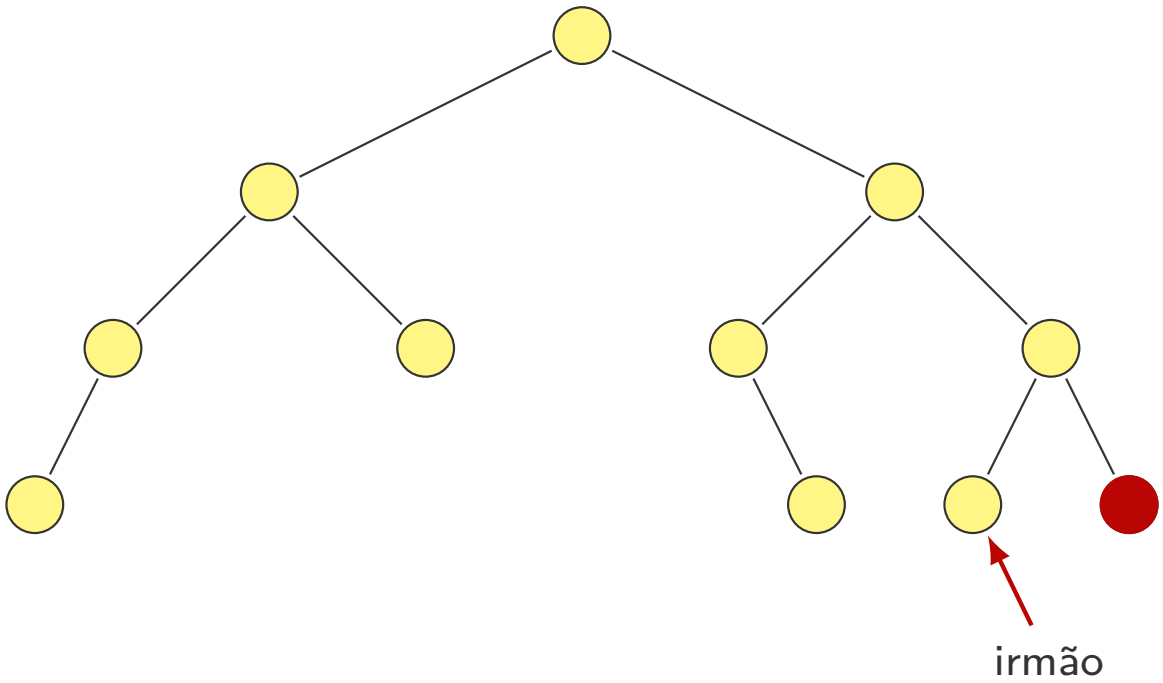
Árvores Binárias

Exemplo de árvore binária:



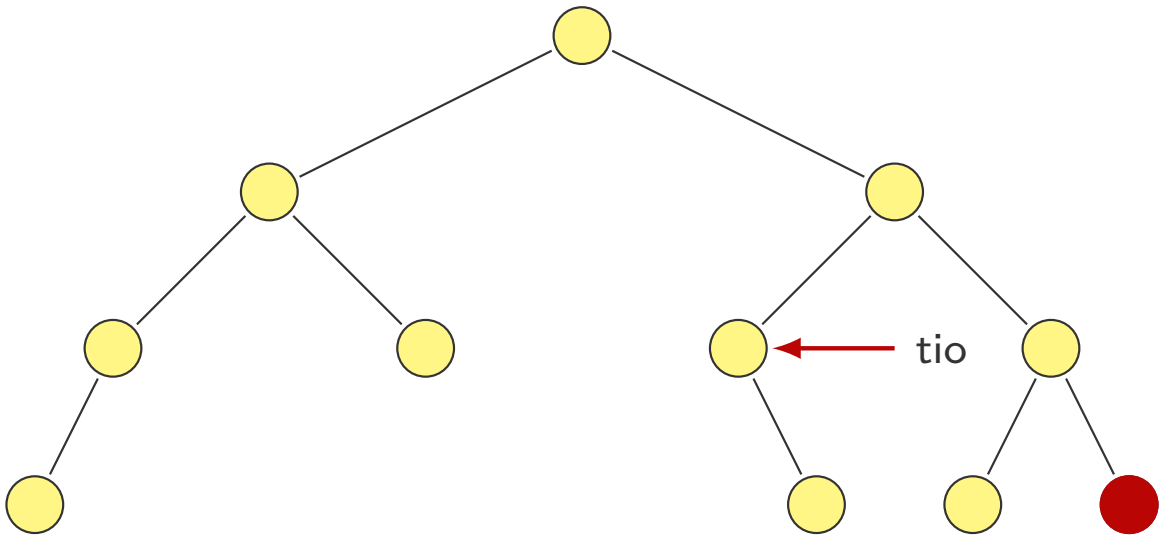
Árvores Binárias

Exemplo de árvore binária:



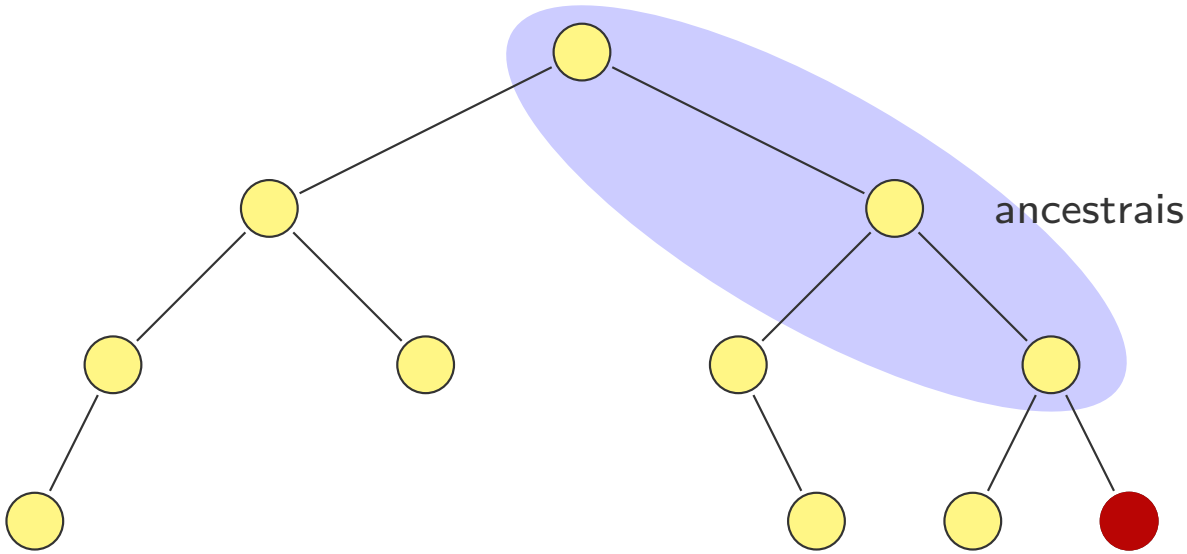
Árvores Binárias

Exemplo de árvore binária:



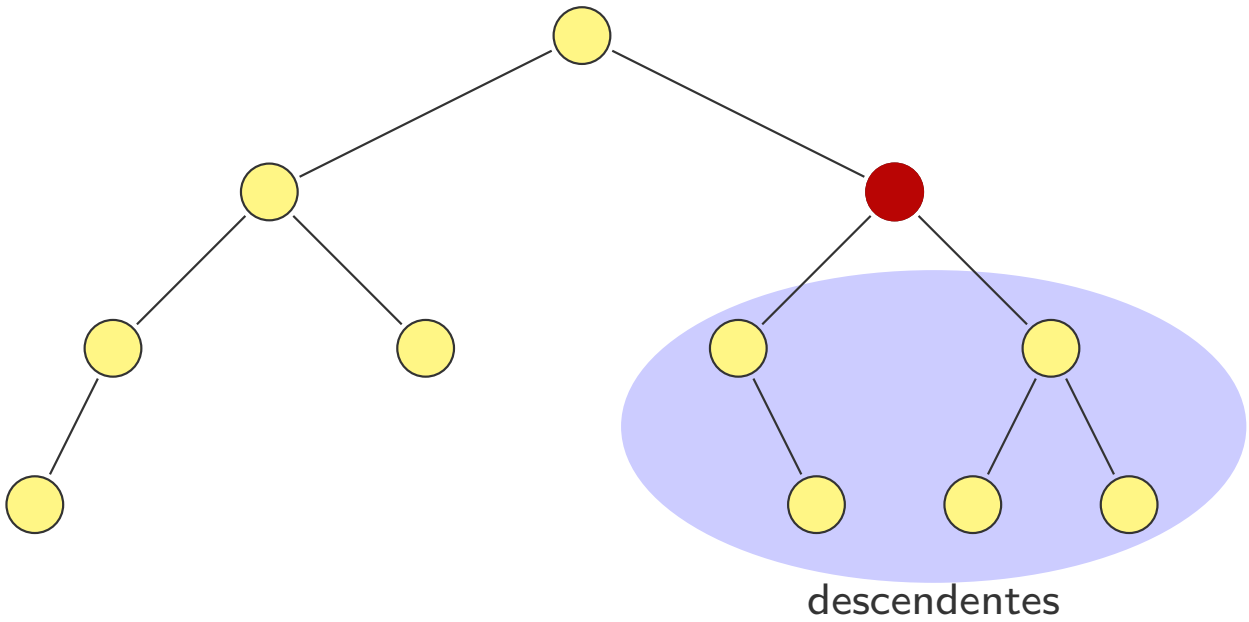
Árvores Binárias

Exemplo de árvore binária:



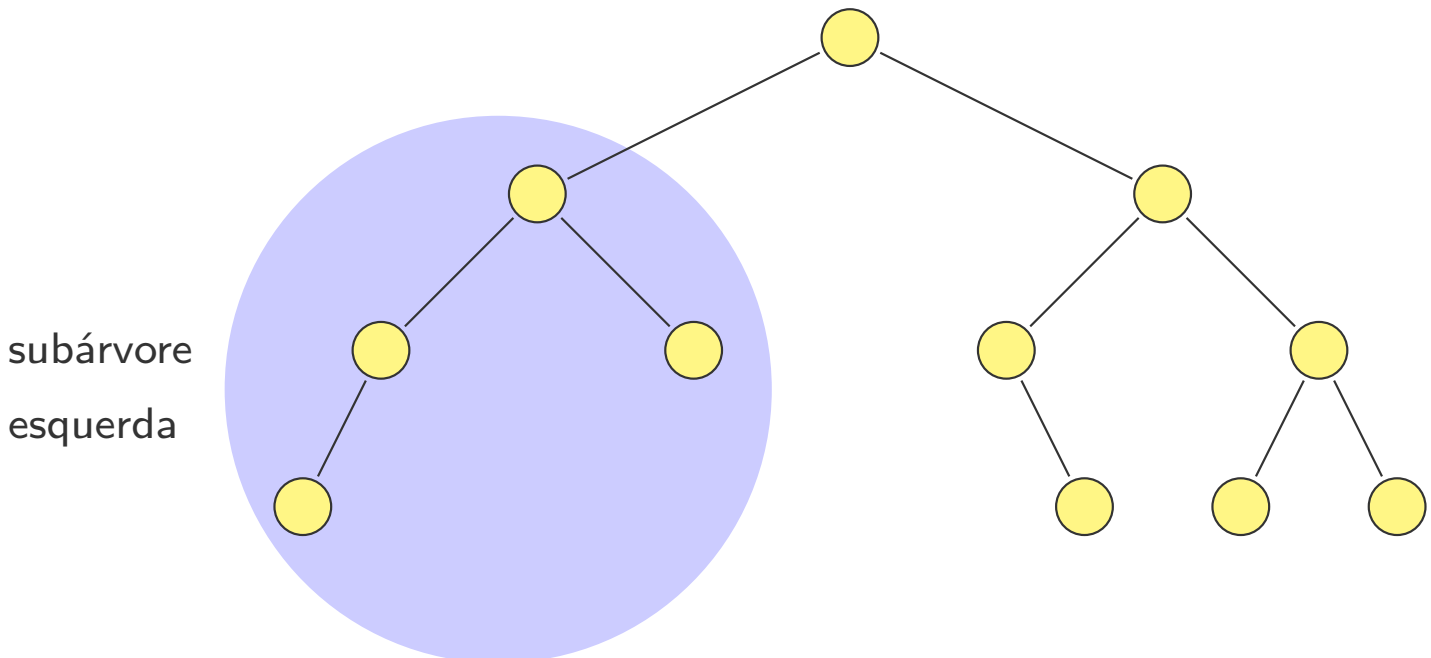
Árvores Binárias

Exemplo de árvore binária:



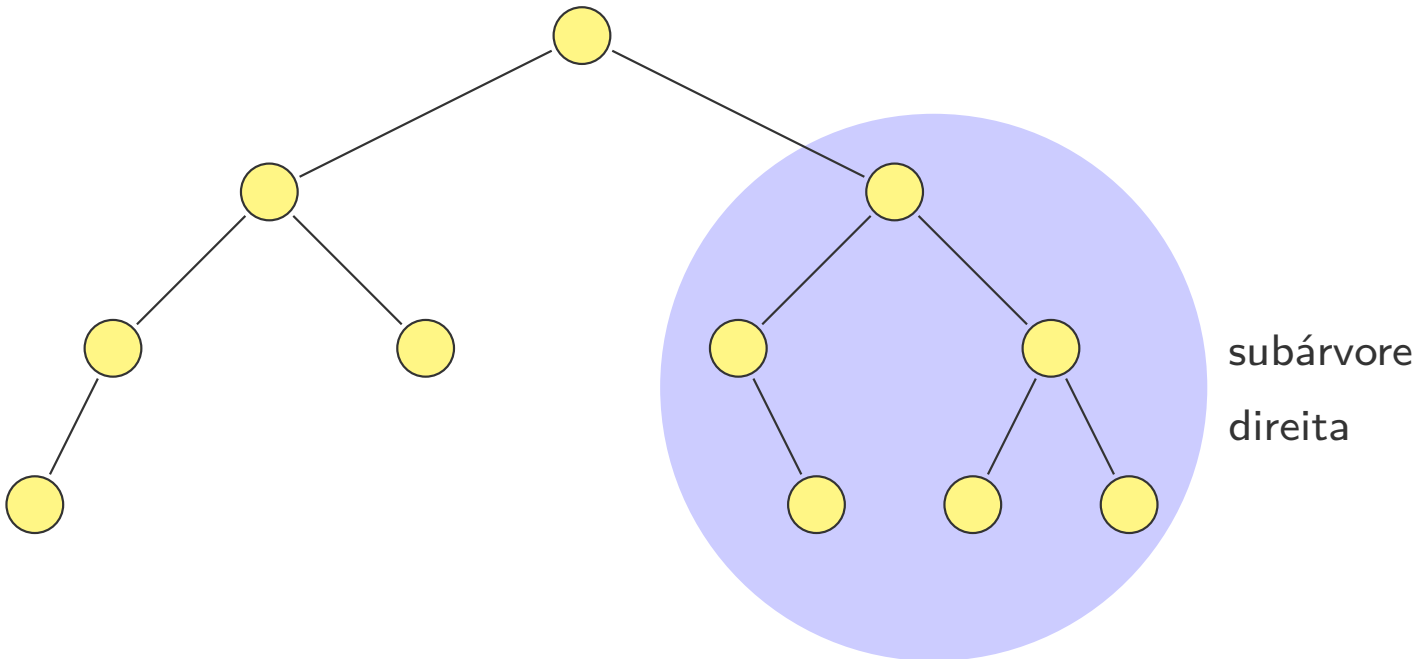
Árvores Binárias

Exemplo de árvore binária:



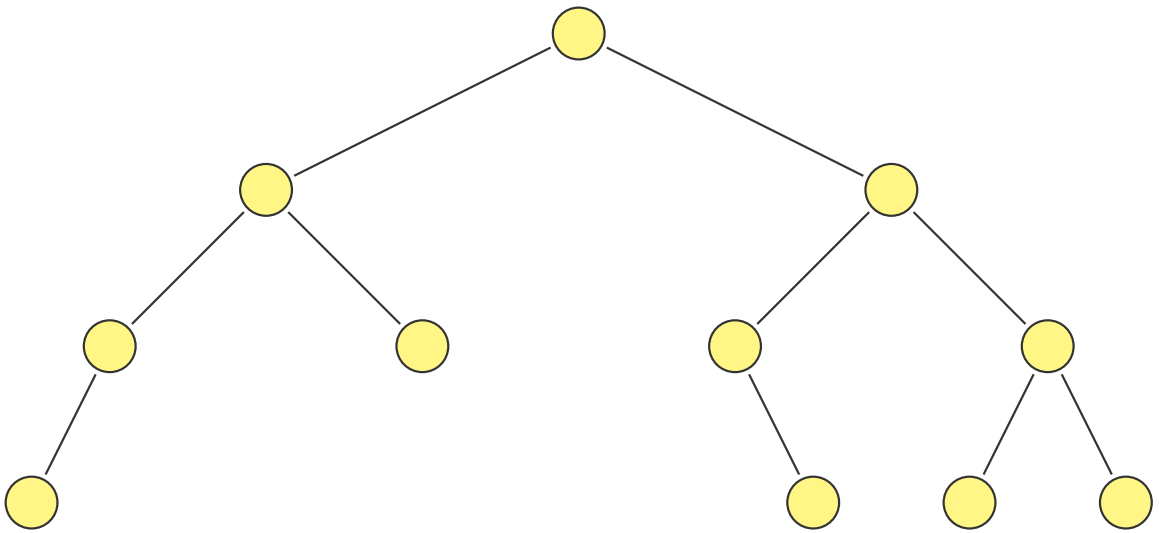
Árvores Binárias

Exemplo de árvore binária:



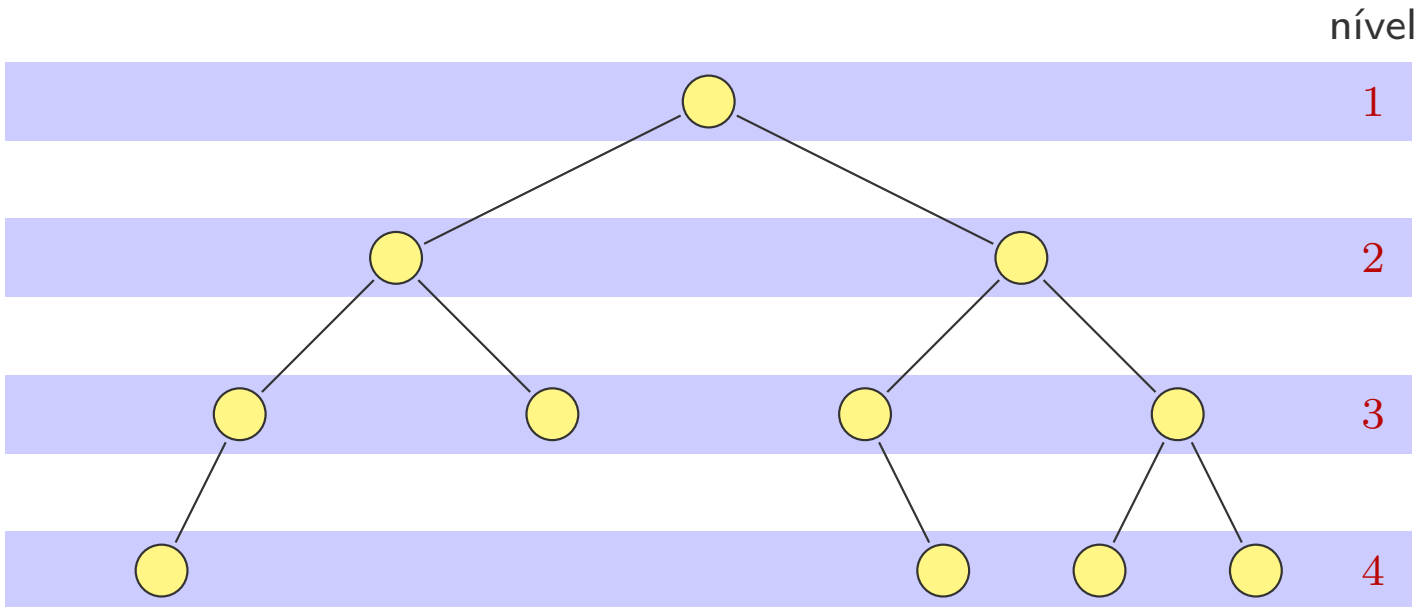
Árvores Binárias — Nível e Altura

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Árvores Binárias — Nível e Altura

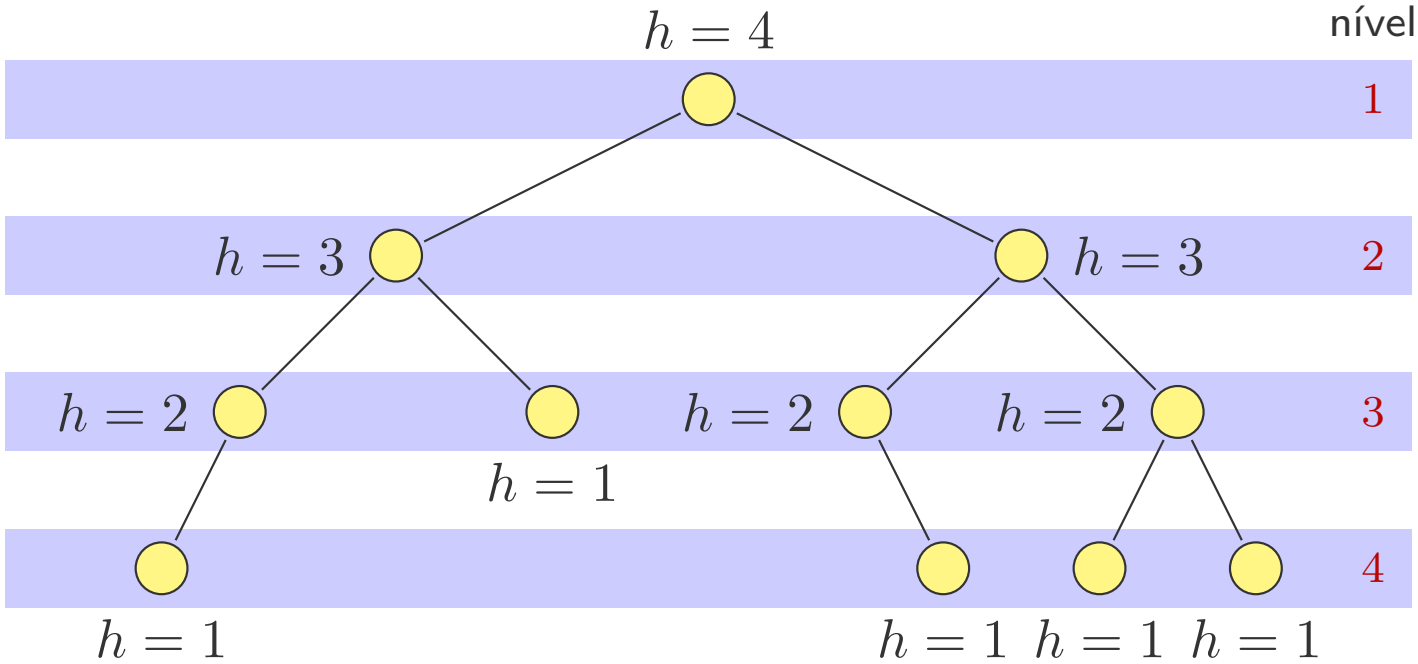
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

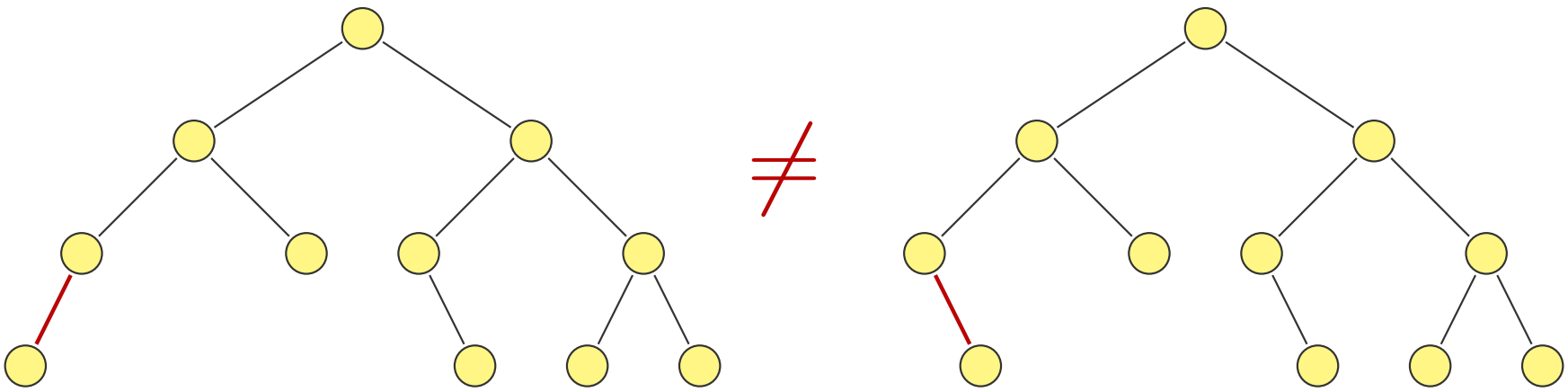
Universidade Federal do Ceará
Campus Quixadá



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



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.

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.

- 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

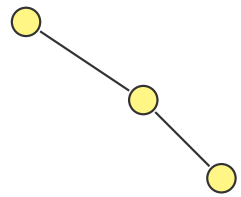
Se a altura é h, então a árvore binária:

Relação entre altura e número de nós da Árvore Binária

Universidade Federal do Ceará
Campus Quixadá

Se a altura é h, então a árvore binária:

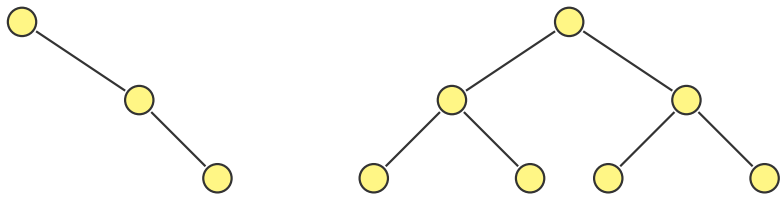
- tem no mínimo h nós



Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

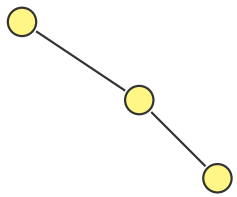
- tem no mínimo h nós
- tem no máximo $2^h - 1$ nós

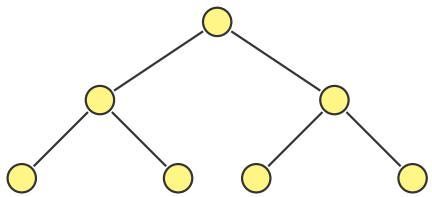


Relação entre altura e número de nós da Árvore Binária

Se a altura é h, então a árvore binária:

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



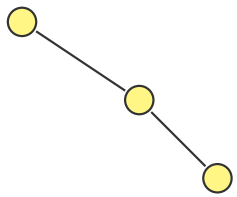


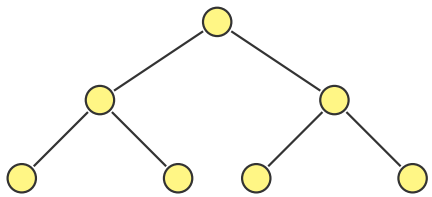
Se a árvore binária tem $n \geq 1$ nós, então:

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





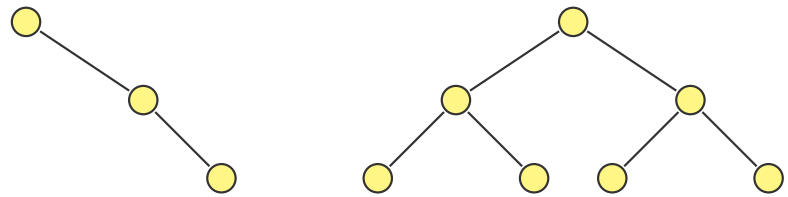
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



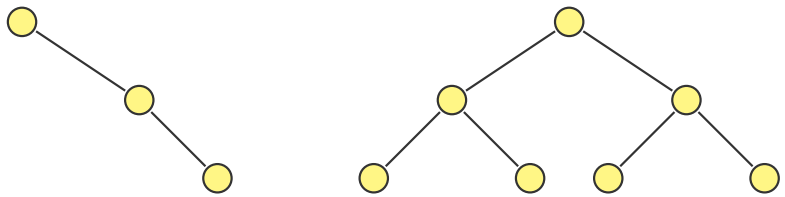
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



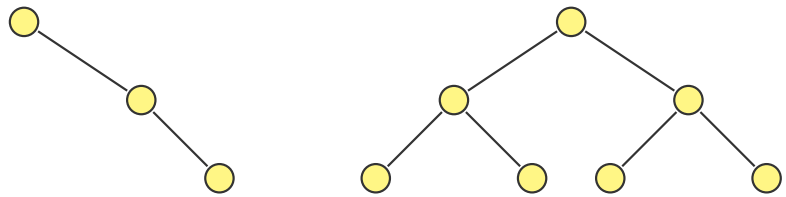
Se a árvore binária tem $n \geq 1$ nós, então:

- 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



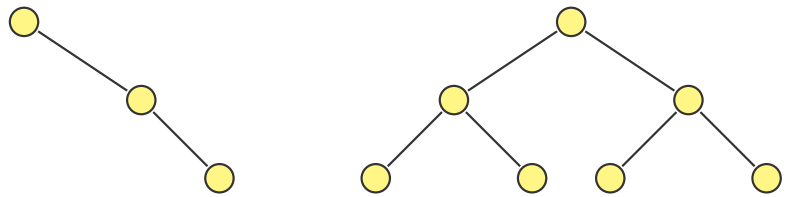
Se a árvore binária tem $n \geq 1$ nós, então:

- 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:

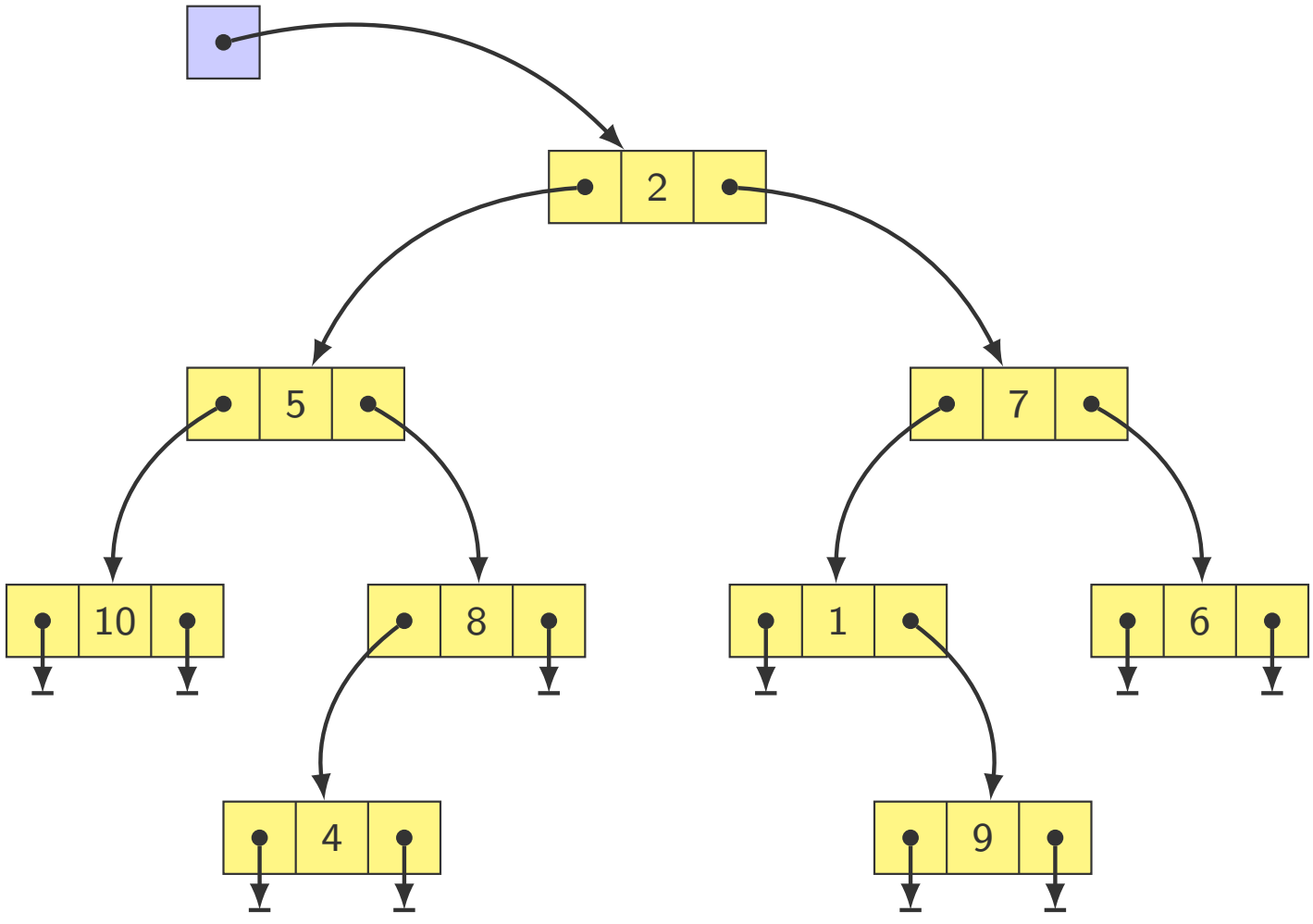
- a altura é no mínimo $\lceil \log_2(n+1) \rceil$
  - $n \leq 2^h - 1 \Rightarrow n + 1 \leq 2^h \Rightarrow \log_2(n+1) \leq \log_2 2^h \Rightarrow h \geq \log_2(n+1)$
  - quando a árvore é completa
- a altura é no máximo n
  - quando cada nó interno tem apenas um filho (a árvore é um caminho)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

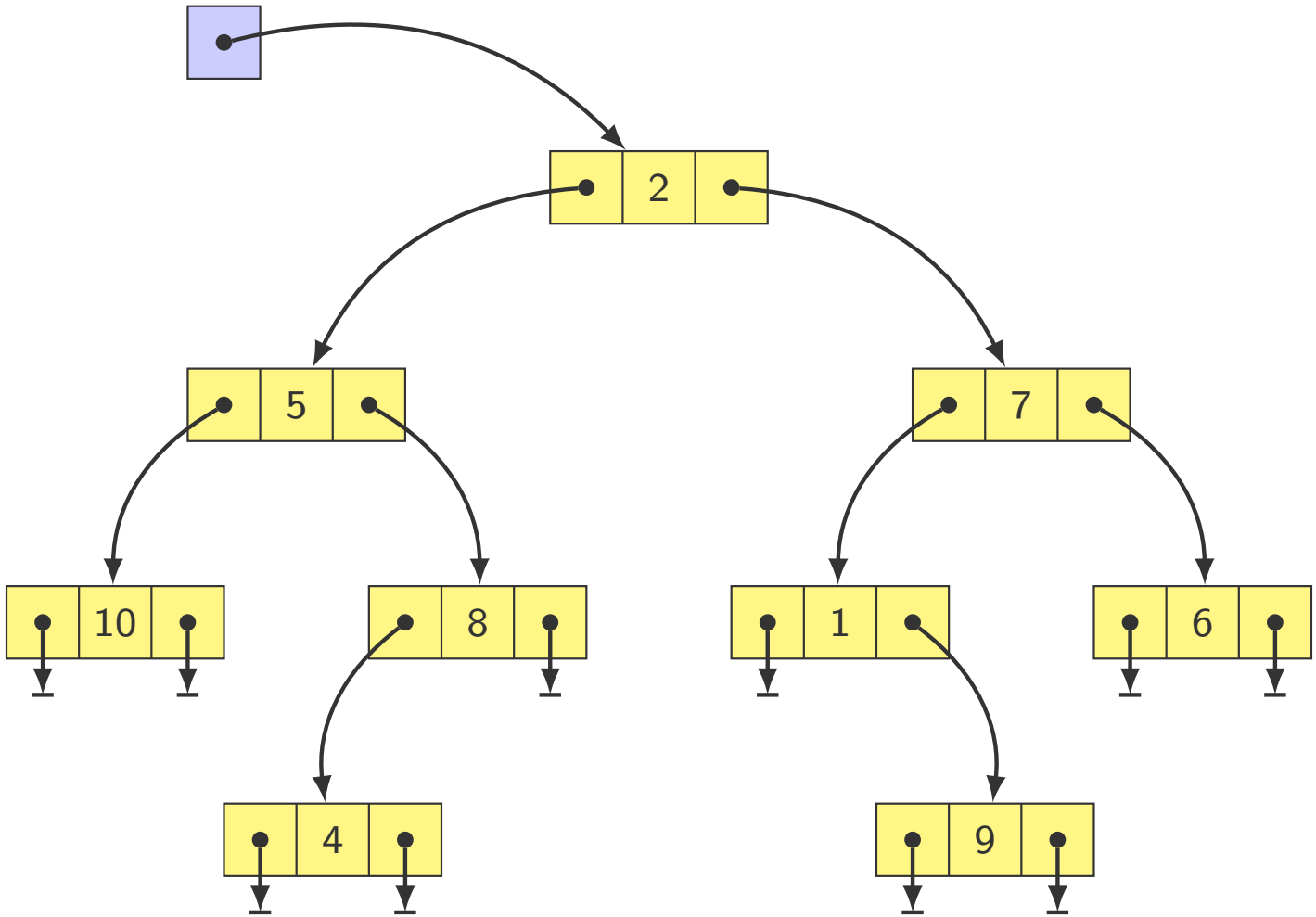
Implementação

Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



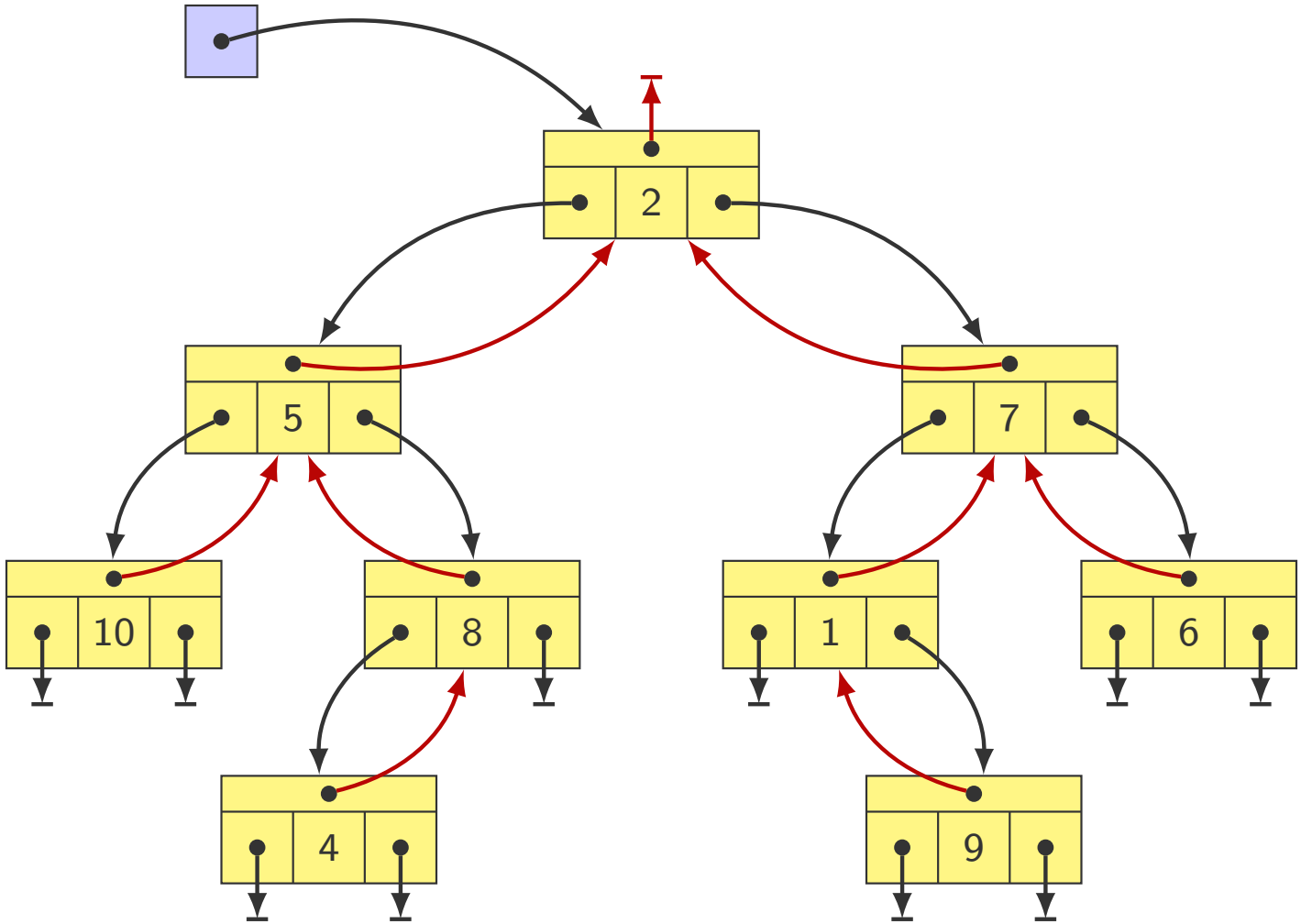
Implementação



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 — Decisões de projeto

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

Implementação — Decisões de projeto

- Os nós da árvore implementada 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 implementada 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 implementada 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 implementada 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 implementada 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

- 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

- Os nós da árvore implementada 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

- 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.:** Vamos supor que todas as chaves a serem armazenadas na árvore são distintas.

Struct Node – Arquivo Tree.h


```
1  struct Node {
2      int key;          // valor a ser guardado
3      Node *left;       // ponteiro para filho esquerdo
4      Node *right;      // ponteiro para filho direito
5
6      Node(int k, Node *l, Node *r) { // Construtor
7          this->key = k;
8          this->left = l;
9          this->right = r;
10     }
11
12     ~Node() { // Destrutor
13         std::cout << this->key << " removed" << std::endl;
14     }
15 };
```

Struct Node – Arquivo `Tree.h`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  struct Node {
2      int key;          // valor a ser guardado
3      Node *left;       // ponteiro para filho esquerdo
4      Node *right;      // ponteiro para filho direito
5
6      Node(int k, Node *l, Node *r) { // Construtor
7          this->key = k;
8          this->left = l;
9          this->right = r;
10     }
11
12     ~Node() { // Destrutor
13         std::cout << this->key << " removed" << std::endl;
14     }
15 };
```

- **Obs.:** O nó também pode ser implementado como uma classe.

Classe Tree – Arquivo Tree.h


```
1  enum class Position { LEFT, RIGHT };
2
3  class Tree {
4  public:
5      Tree();
6      Tree(int rootKey);
7      void add(int key, int parent, Position p);
8      bool contains(int key);
9      bool empty();
10     void printKeys();
11     void clear();
12     ~Tree();
13 private:
14     Node *root;
15     void add(Node *root, int key, int parent, Position p);
16     bool contains(Node *root, int key);
17     void printKeys(Node *root);
18     Node *clear(Node *root);
19 };
```

Tree.cpp — Implementação

Construtores:

```
1  #include <iostream>
2  #include "Tree.h"
3
4  // Construtor default: cria uma árvore binária vazia
5  Tree::Tree() {
6      root = nullptr;
7  }
8
9  // Construtor: cria uma árvore com um nó raiz
10 // contendo a chave passada como argumento
11 Tree::Tree(int rootKey) {
12     root = new Node(rootKey, nullptr, nullptr);
13 }
```

`Tree.cpp` — Implementação

Saber se a árvore é vazia:

Tree.cpp — Implementação

Saber se a árvore é vazia:

```
1 // retorna true se a árvore está vazia
2 bool Tree::empty() {
3     return root == nullptr;
4 }
```

Tree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 // retorna true se a árvore está vazia
2 bool Tree::empty() {
3     return root == nullptr;
4 }
```

Percorrendo e imprimindo a árvore:

Tree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 // retorna true se a árvore está vazia
2 bool Tree::empty() {
3     return root == nullptr;
4 }
```

Percorrendo e imprimindo a árvore:

```
1 // Função recursiva
2 void Tree::printKeys(Node *node) {
3     if(node != nullptr) {
4         printKeys(node->left);
5         std::cout << node->key << std::endl;
6         printKeys(node->right);
7     }
8 }
```

Tree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Saber se a árvore é vazia:

```
1 // retorna true se a árvore está vazia
2 bool Tree::empty() {
3     return root == nullptr;
4 }
```

Percorrendo e imprimindo a árvore:

```
1 // Função recursiva
2 void Tree::printKeys(Node *node) {
3     if(node != nullptr) {
4         printKeys(node->left);
5         std::cout << node->key << std::endl;
6         printKeys(node->right);
7     }
8 }
```

```
1 // Função pública
2 // Imprime na tela todas as chaves da árvore
3 void Tree::printKeys() {
4     printKeys(root);
5 }
```

`Tree.cpp` — Implementação

Buscando uma chave na árvore:

Tree.cpp — Implementação

Buscando uma chave na árvore:

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

Tree.cpp — Implementação

Buscando uma chave na árvore:

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

Observações:

Tree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

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

Tree.cpp — Implementação

Universidade Federal do Ceará
Campus Quixadá

Buscando uma chave na árvore:

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

Buscando uma chave na árvore (função pública):

```
1 // função pública: retorna true se e somente se
2 // a árvore contém a chave passada como argumento
3 bool Tree::contains(int key) {
4     contains(root, key);
5 }
```

Tree.cpp — Implementação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Liberando memória alocada para a árvore:

```
1  // Função recursiva que libera todos os nós da árvore.
2  // Ao final, retorna a raiz da árvore resultante = nullptr
3  Node *Tree::clear(Node *node) {
4      if(node != nullptr) {
5          node->left = clear(node->left);
6          node->right = clear(node->right);
7          delete node;
8          return nullptr;
9      }
10 }
```

Tree.cpp — Implementação

Liberando memória alocada para a árvore:

```
1  // Função recursiva que libera todos os nós da árvore.
2  // Ao final, retorna a raiz da árvore resultante = nullptr
3  Node *Tree::clear(Node *node) {
4      if(node != nullptr) {
5          node->left = clear(node->left);
6          node->right = clear(node->right);
7          delete node;
8          return nullptr;
9      }
10 }
```

```
1  // Função pública que libera todos os nós
2  // da árvore, deixando ela vazia
3  void Tree::clear() {
4      root = clear(root);
5  }
```

Tree.cpp — Implementação

Destrutor:

```
1 // Destrutor: libera todos os nós
2 // alocados dinamicamente
3 Tree::~Tree() {
4     clear();
5 }
```

Tree.cpp — Implementação

Adicionar uma nova chave:

```
1  // função pública
2  // adiciona na árvore a chave passada como argumento
3  // o pai do novo nó será o nó com chave igual a 'parent'
4  // e o novo nó será inserido à esquerda ao à direita
5  // de acordo com o valor do parâmetro pos.
6  // Supõe que todas as chaves na árvore são distintas
7  void Tree::add(int key, int parent, Position pos){
8      if(root == nullptr) {
9          root = new Node(key, nullptr, nullptr);
10     }
11     else add(root, key, parent, pos);
12 }
```

Tree.cpp — Implementação

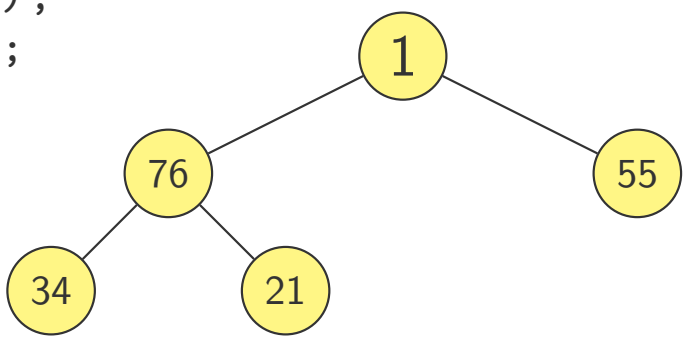
Adicionar uma nova chave:

```
1  void Tree::add(Node *node, int key, int parent, Position p) {
2      if(node->key == key) return;
3      if(node->key == parent && p == Position::LEFT) {
4          if(node->left != nullptr) return;
5          Node *novo = new Node(key, nullptr, nullptr);
6          node->left = novo;
7          return;
8      }
9      if(node->key == parent && p == Position::RIGHT) {
10         if(node->right != nullptr)  return;
11         Node *novo = new Node(key, nullptr, nullptr);
12         node->right = novo;
13         return;
14     }
15     if(node->left != nullptr)
16         add(node->left, key, parent, p);
17     if(node->right != nullptr)
18         add(node->right, key, parent, p);
19 }
```

main.cpp — Exemplo de programa cliente

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  #include <iostream>
2  #include "Tree.h"
3
4  // Cria árvore com 5 nós, imprime chaves e finaliza
5  // liberando a memória que foi alocada para a árvore
6  int main()
7  {
8      Tree t(1); // Cria árvore com nó raiz de chave 1
9      t.add(76, 1, Position::LEFT);
10     t.add(55, 1, Position::RIGHT);
11     t.add(21, 76, Position::RIGHT);
12     t.add(34, 76, Position::LEFT);
13
14     t.printKeys();
15     return 0;
16 }
```



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