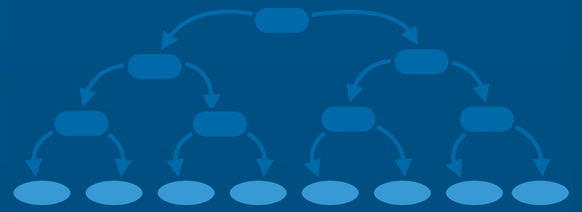
# Árvores B

## Estrutura de Dados e Algoritmos - IC/UFF

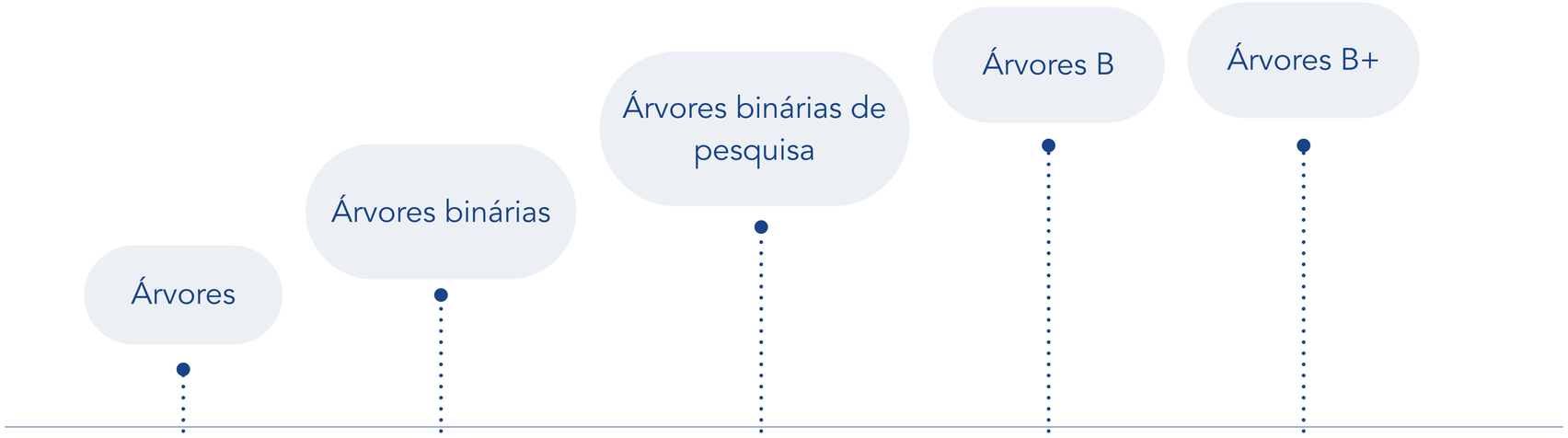
Lucas Roberto | Mariane Santos - Nov/23



# Agenda

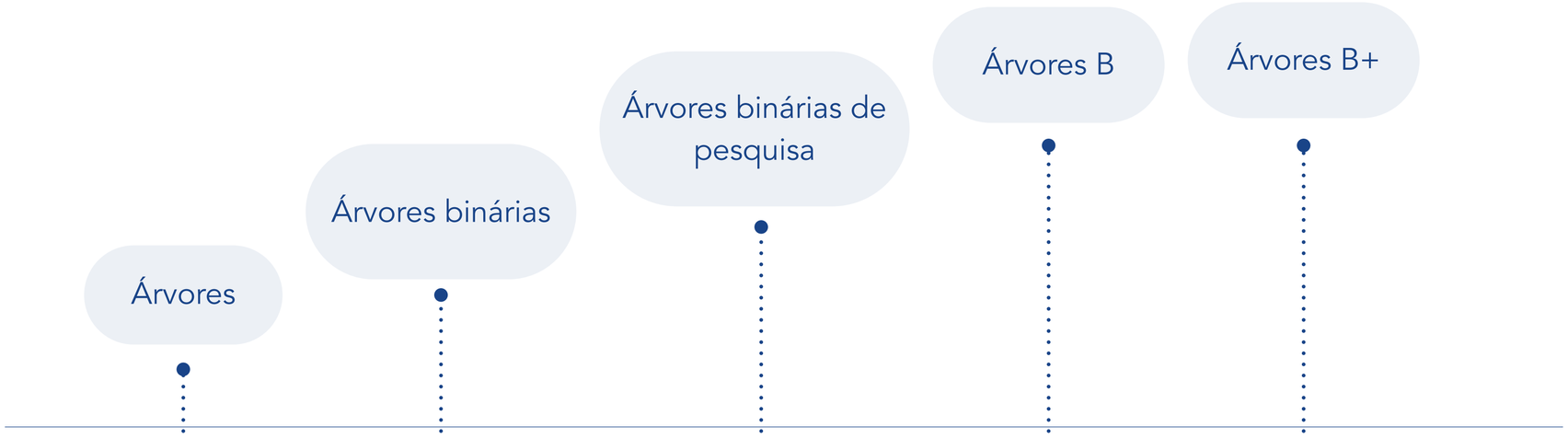
1. Contexto e Motivação
2. Árvores B
3. Operações e Propriedades das árvores B
4. Implementação
5. Análise de complexidade / Tempo de execução
6. Conclusão

# Estruturas de dados: Árvores



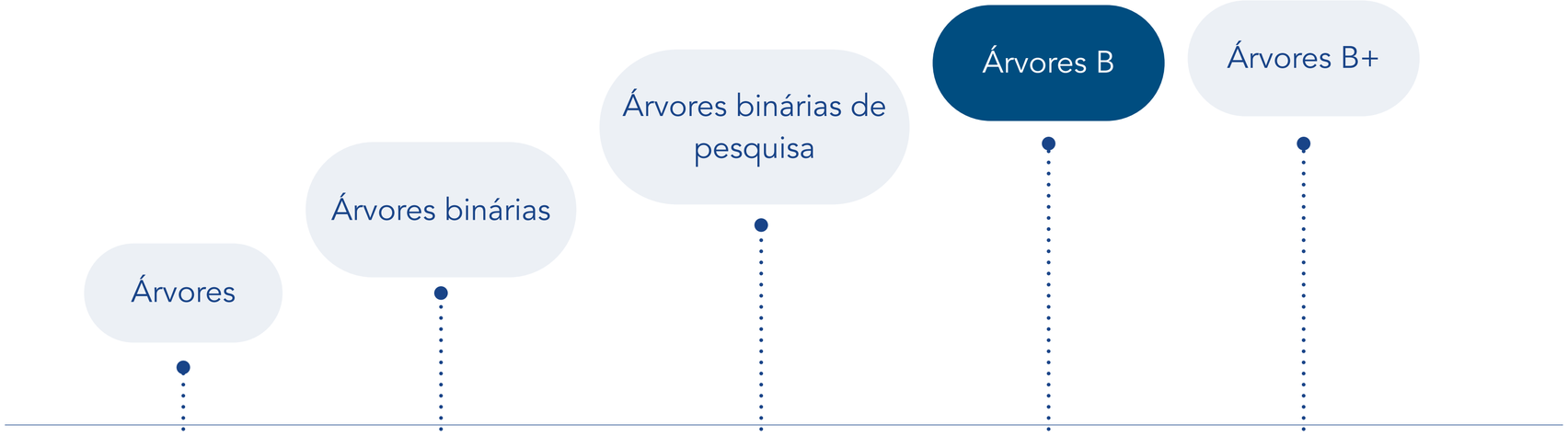
# Estruturas de dados: Árvores

**Objetivo:** Armazenar e organizar dados de maneira eficiente



# Estruturas de dados: Árvores

**Rudolf Bayer** e **Edward M. McCreight** (1972)
*Organization and Maintenance of Large Ordered Indexes*



# Árvores B

Características da árvore

Memória Secundária

Aplicações

# Árvores B

Características da árvore | Memória Secundária | Aplicações

**Aridade:** Árvores B são **M**-árias, baseadas no parâmetro **t,** que determina o **grau** da árvore

**Preenchimento:** Árvores B são **completas**

**Balanceamento:** Caracterizam-se por serem **perfeitamente balanceadas**, onde todas as **folhas** apresentam-se **em um mesmo nível**

**Nós:** Permite a existência de um número indefinido de filhos com **mais de uma chave/ponteiro por nó**

**Altura:** Possui uma altura, em média, menor que outras estruturas, permitindo **caminhos mais curtos**

# Árvores B

Características da árvore | **Memória Secundária** | Aplicações

**Memória x Velocidade:** Sabemos que a memória **principal** tem **alta velocidade**, mas **pouca capacidade**, enquanto a memória **secundária** tem **baixa velocidade**, mas **grande capacidade**.

**Necessidade a ser resolvida:** Situações onde se trabalha com **grandes volumes de dados** que **não cabem** em sua totalidade na memória **principal**, possibilitando o carregamento de cada nó da árvore da memória **secundária** conforme é necessário

# Árvores B

Características da árvore

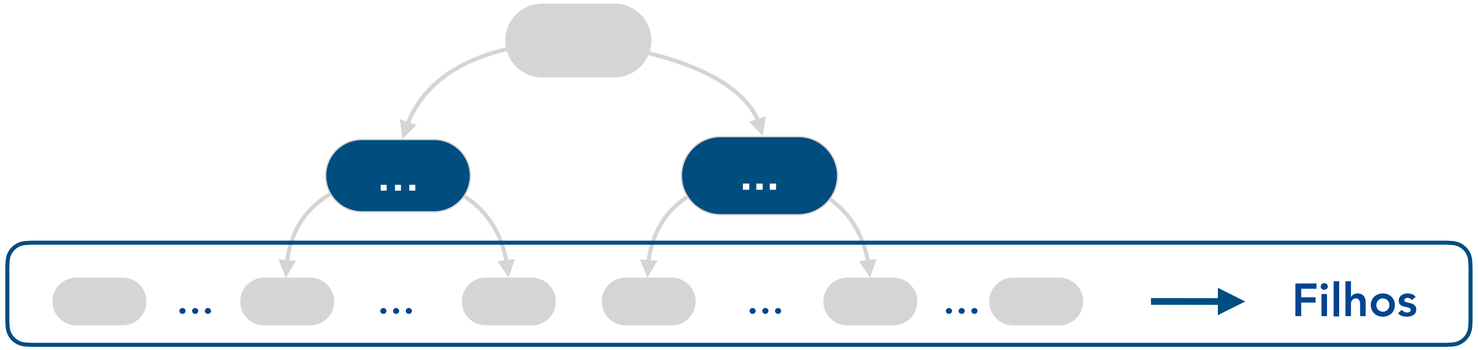
Memória Secundária

Aplicações

Utilizadas em aplicações variadas, tais como **banco de dados** e **sistemas de arquivos** em sistemas operacionais

# Propriedades de uma árvore B

## Grau da árvore: t

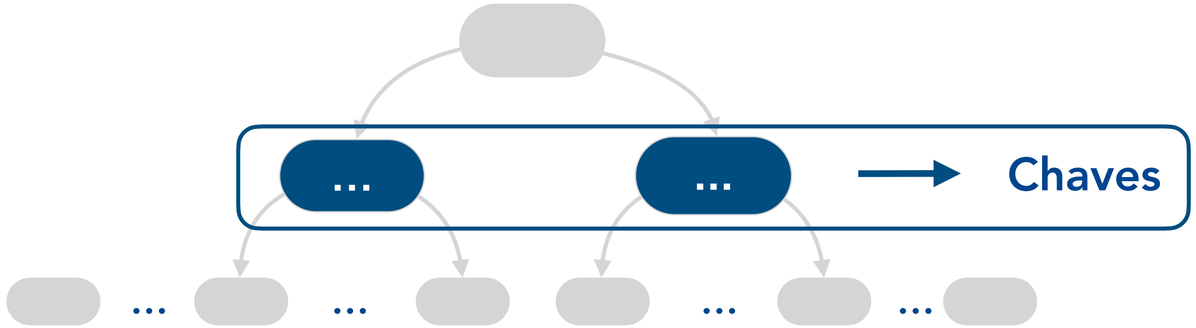


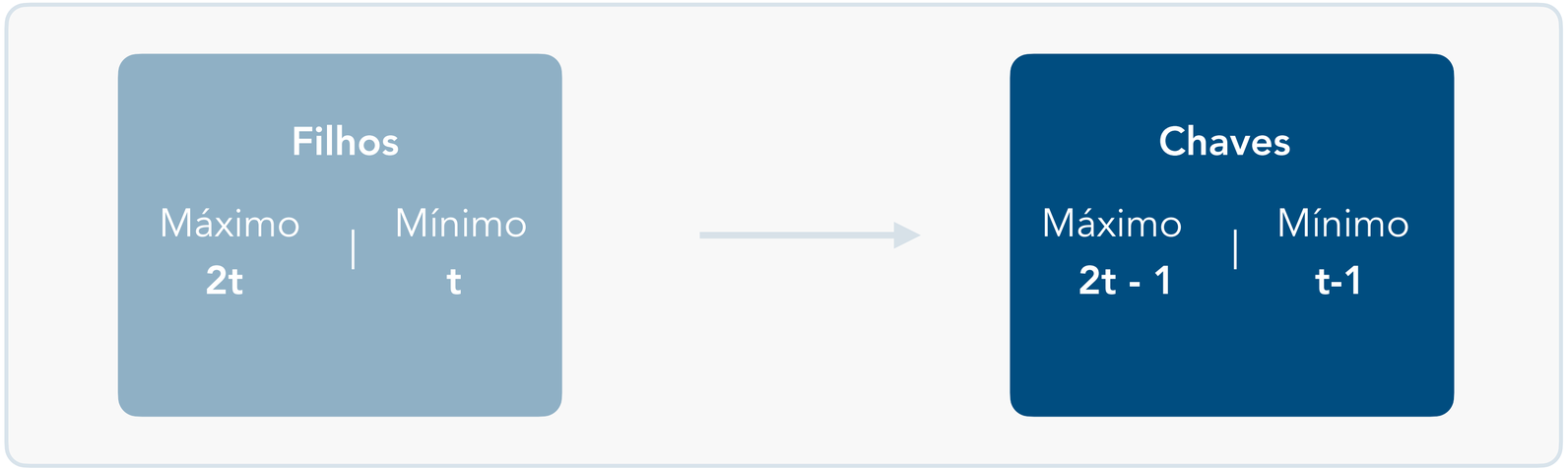

**Filhos**

| Máximo | Mínimo |
|---|---|
| **2t** | **t** |

# Propriedades de uma árvore B

## Grau da árvore: t

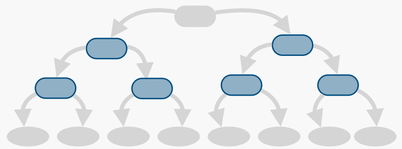


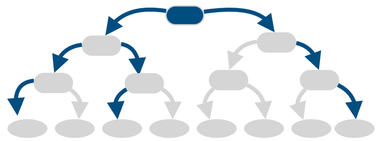
# Propriedades de uma árvore B

Grau da árvore

Cada nó, exceto pela raiz e pelas folhas, possui no **mínimo t** e no **máximo 2t filhos**



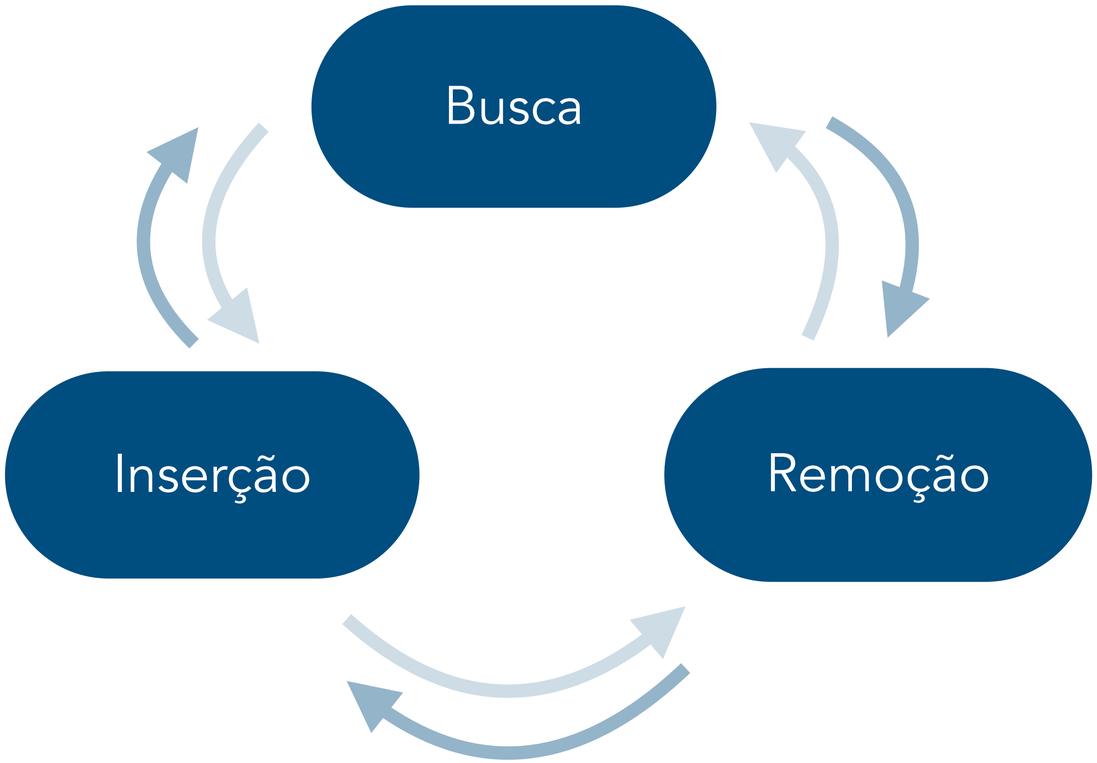
Cada nó tem **pelo menos t-1** e no máximo **2t-1 chaves**



A raiz é **uma folha** ou tem no **mínimo 2 filhos**

Cada **caminho** da raiz até qualquer folha tem o **mesmo tamanho**

# Operações de uma árvore B



# Operações de uma árvore B

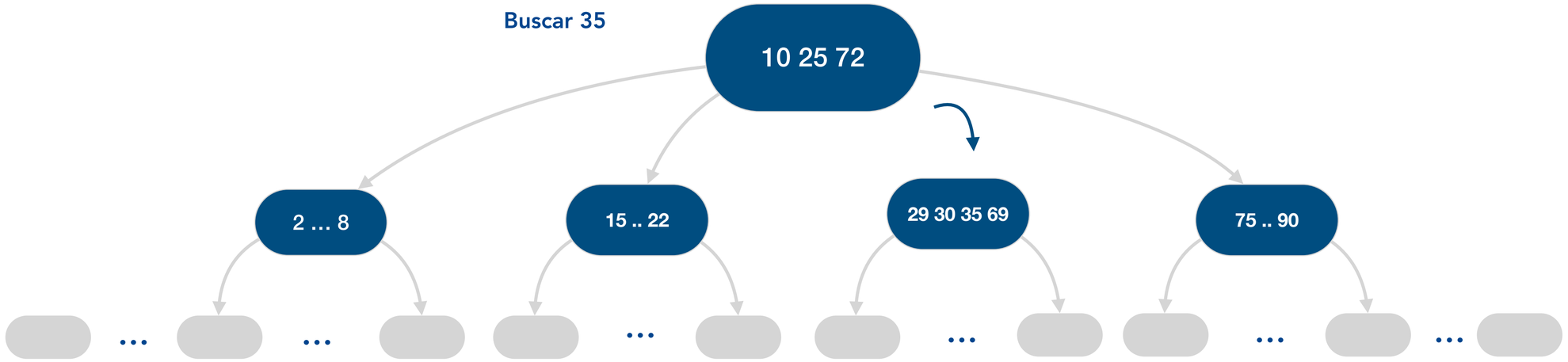
Busca

Operação mais simples: Busca/Procura por um elemento armazenado na árvore B

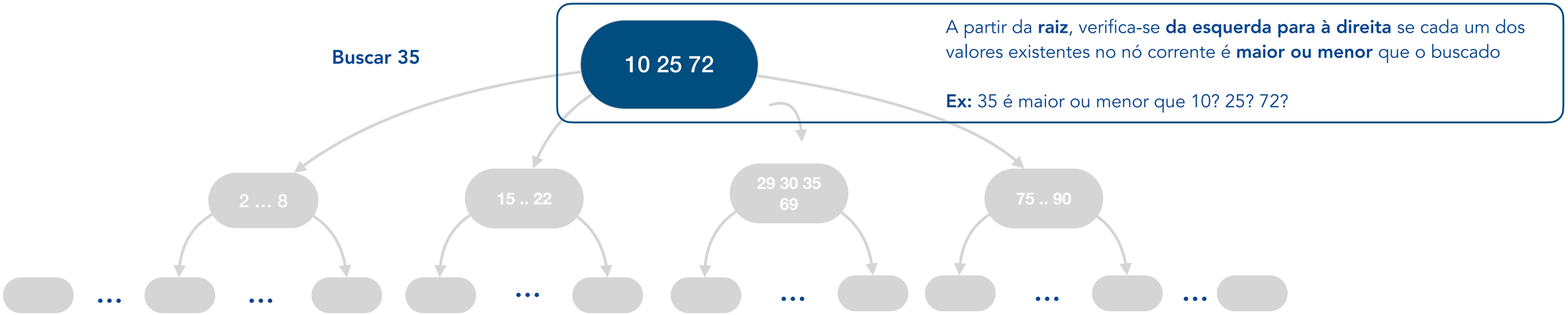
Inserção

Remoção

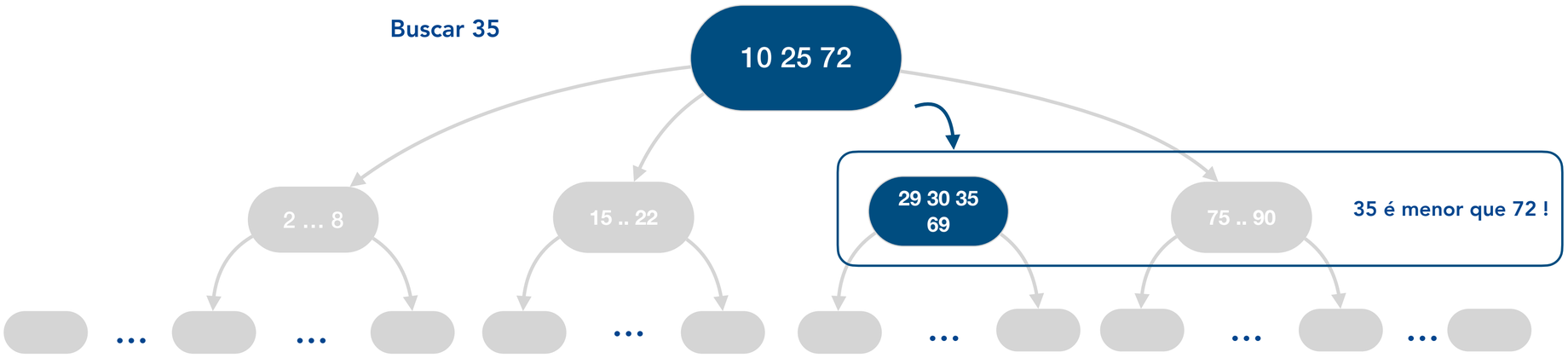
# Operações de uma árvore B: **Busca**



# Operações de uma árvore B: **Busca**



# Operações de uma árvore B: **Busca**



# Operações de uma árvore B

Busca

Operação mais simples: Busca/Procura por um elemento armazenado na árvore B

Inserção

Busca-se manter as propriedades de funcionamento da árvore B, em especial que todas as folhas tenham a **mesma profundidade** e os nós tenham um **número máximo de chaves**

Processo de **split:** Único meio de uma árvore B crescer

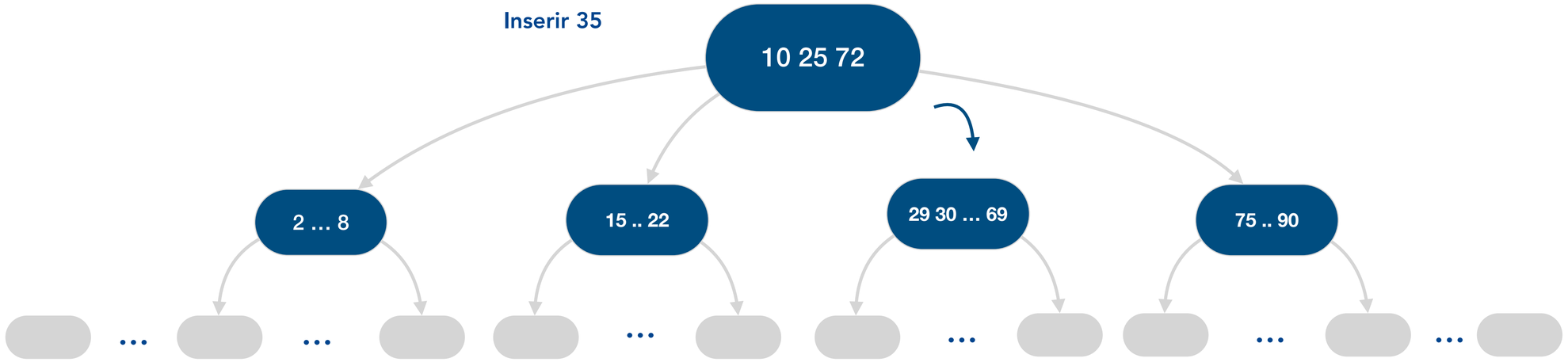
**Altura:** A altura de uma árvore B aumenta em cima, em vez de embaixo

Remoção

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

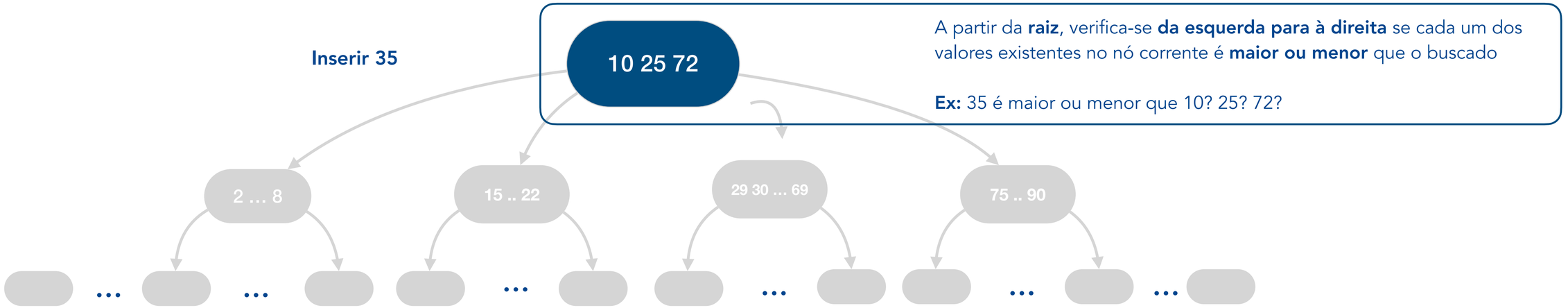
# Operações de uma árvore B: **Inserção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

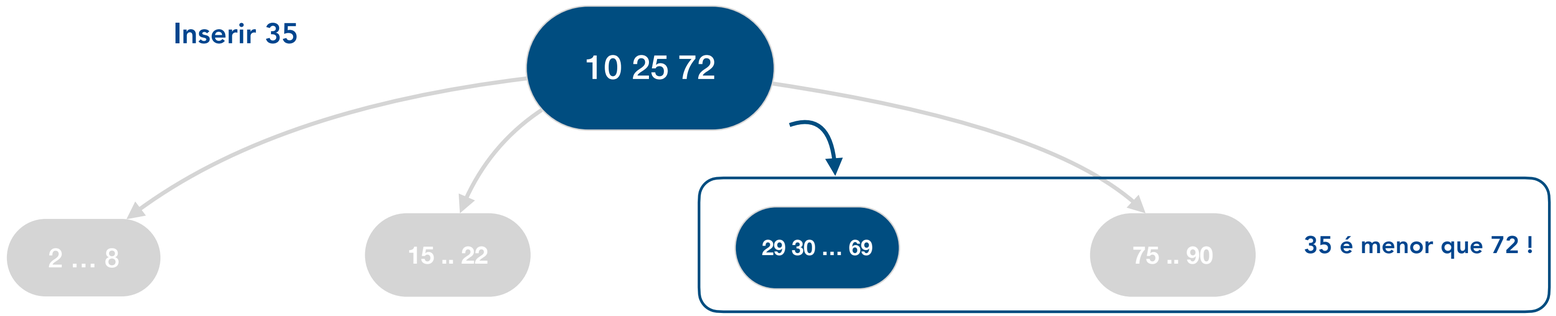
# Operações de uma árvore B: **Inserção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

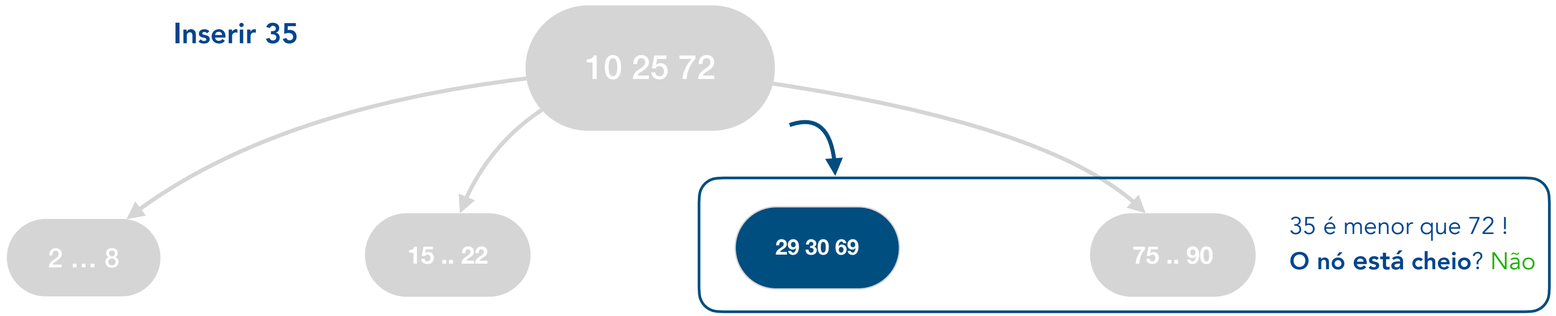
# Operações de uma árvore B: **Inserção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

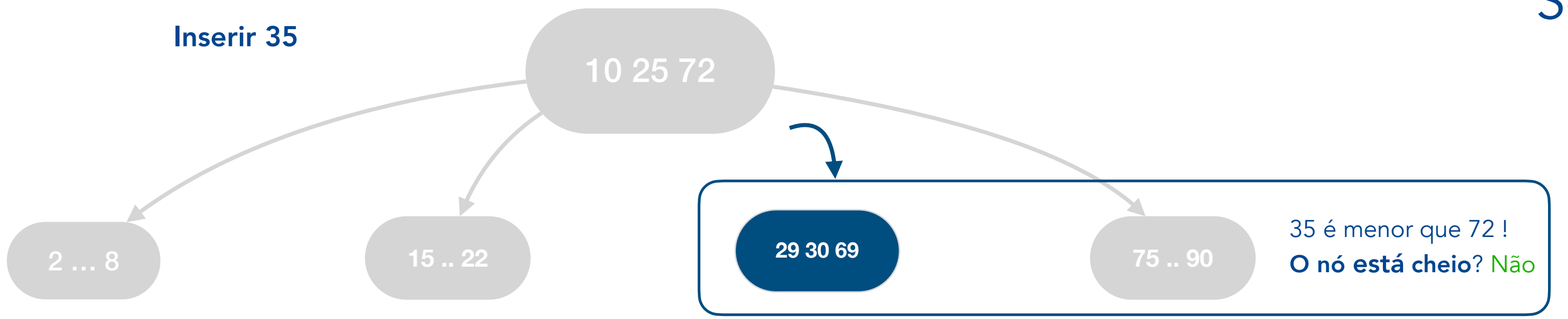
# Operações de uma árvore B: **Inserção simples**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção simples**

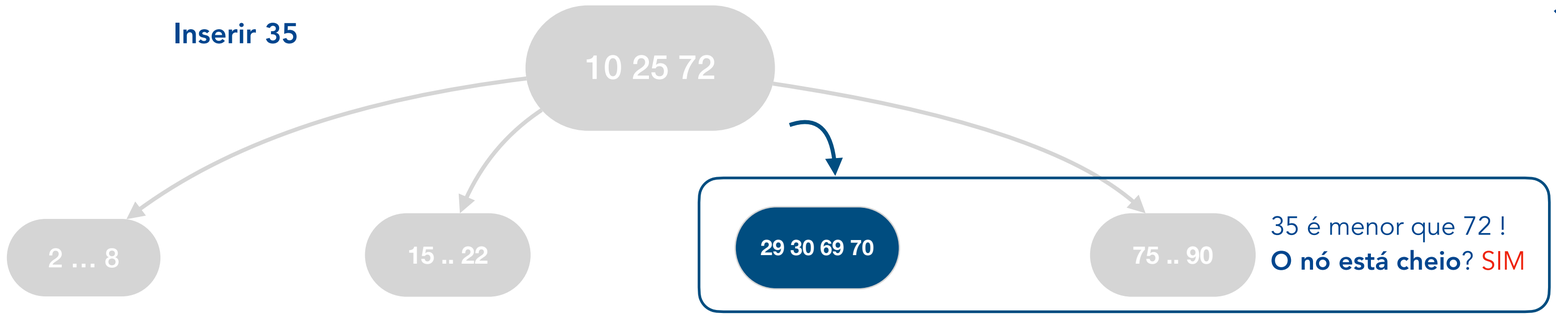



**Caso1:**
Se **não** então **insere**

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção + Split**

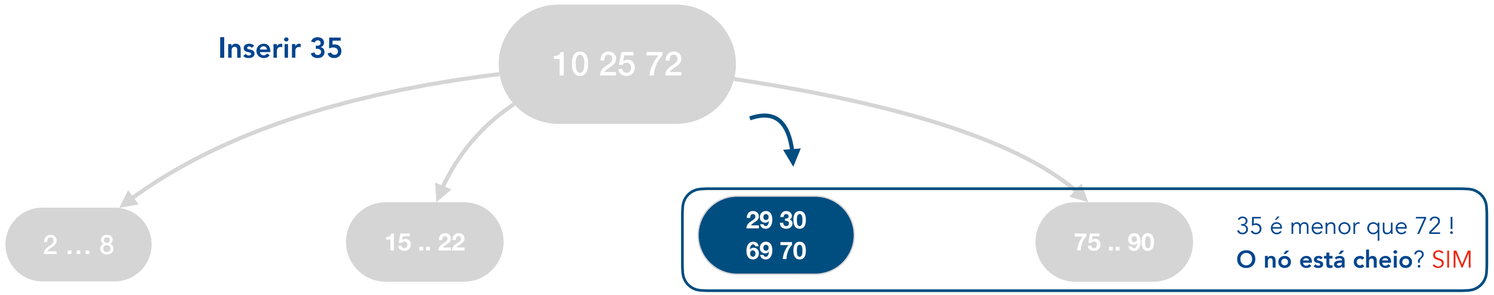



**Caso2:**
Se **sim** então **split**

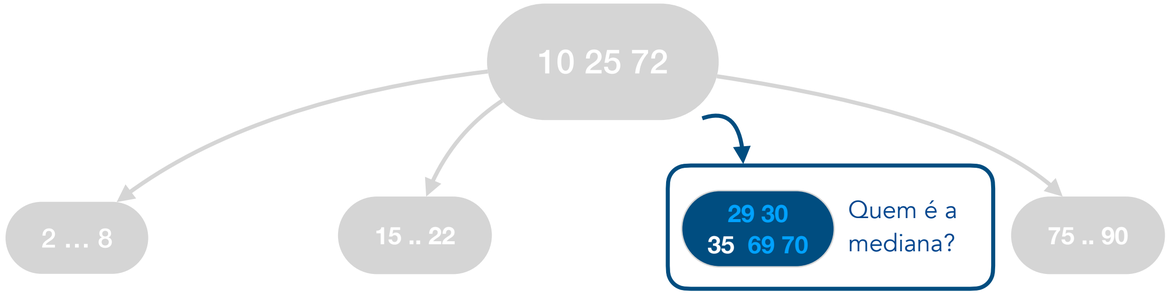
**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção + Split**



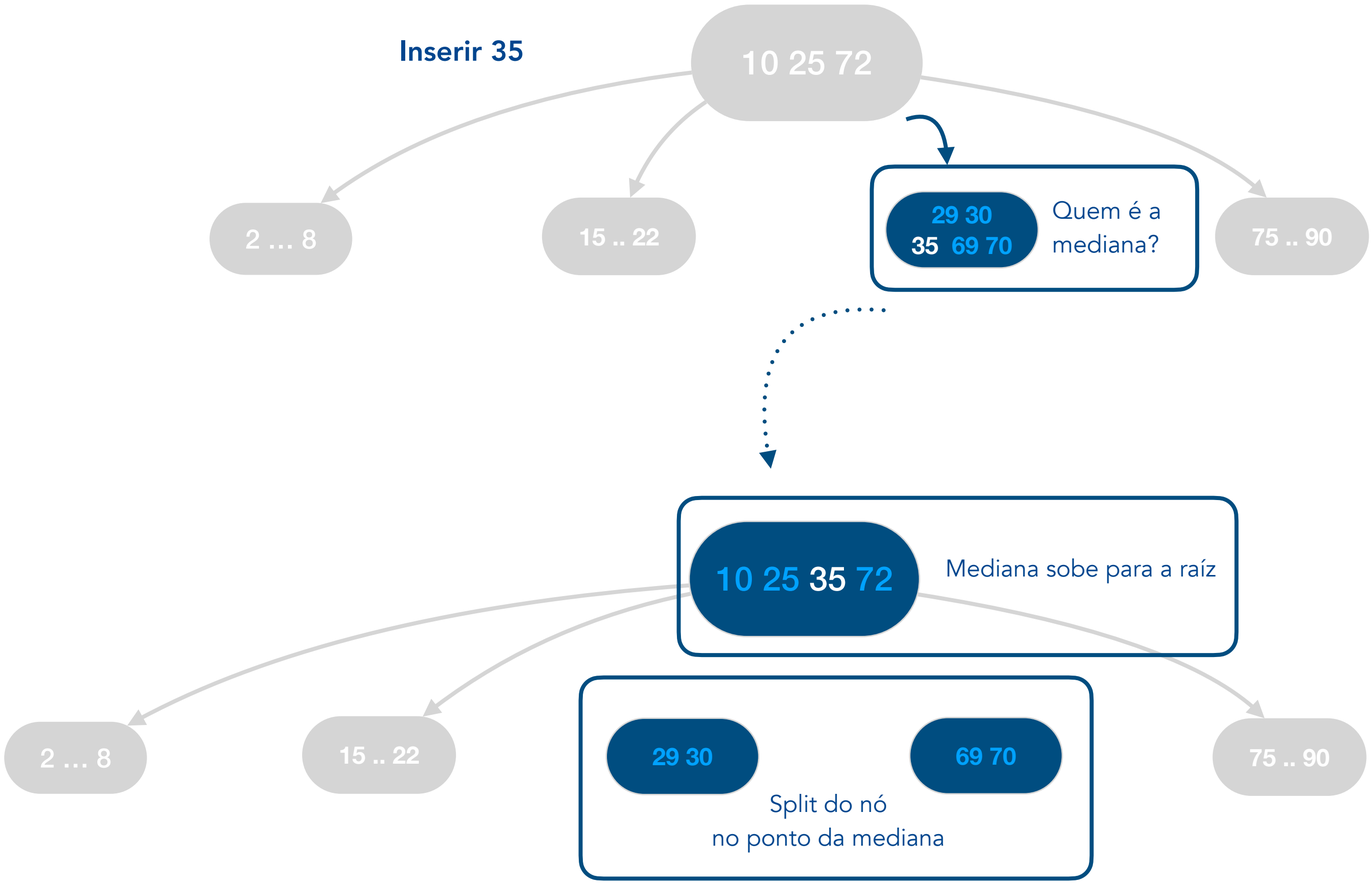

**Caso2:**
Se **sim** então **split**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção + Split**

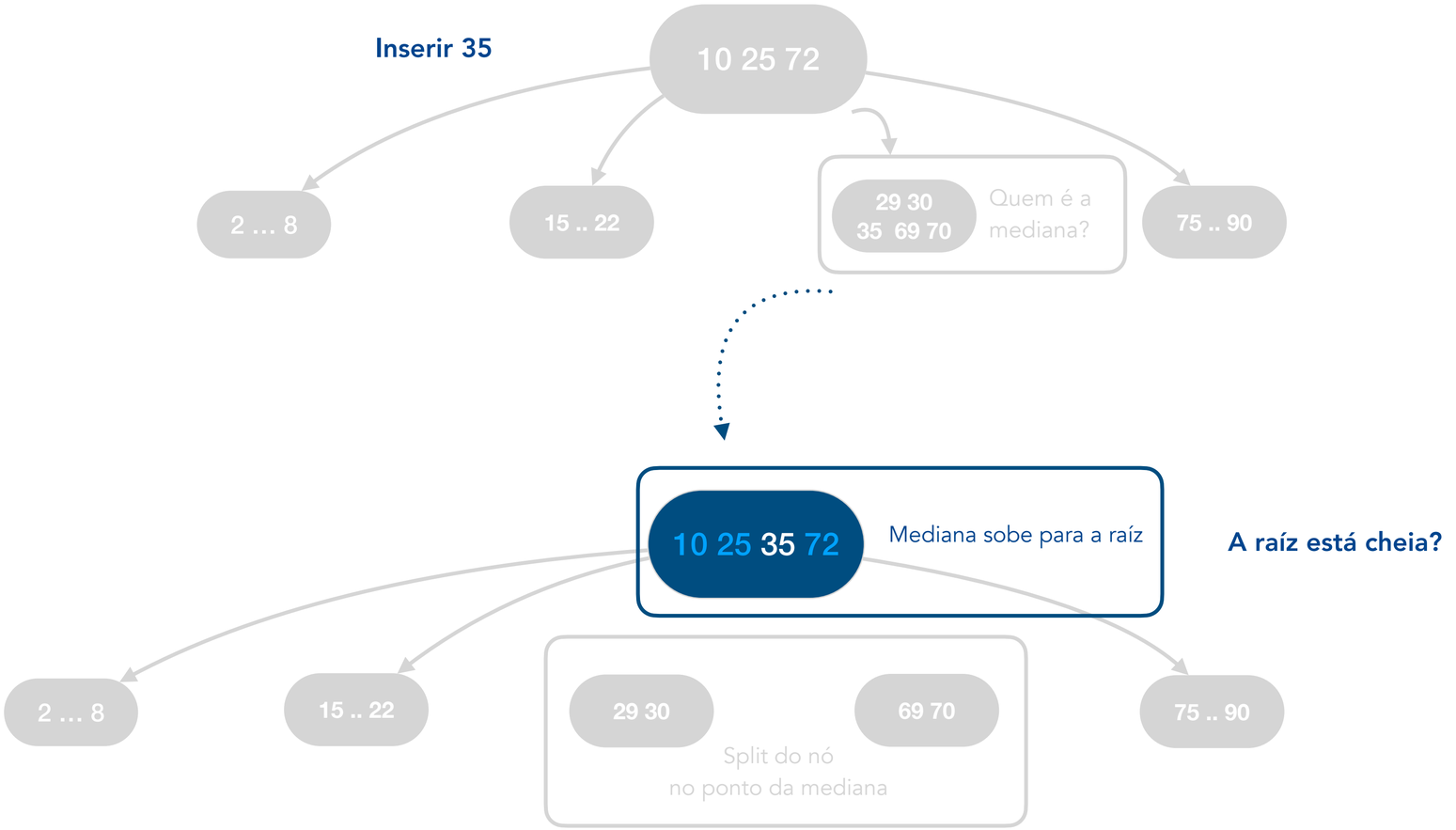



**Caso2:**
Se **sim** então **split**

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção + Split**



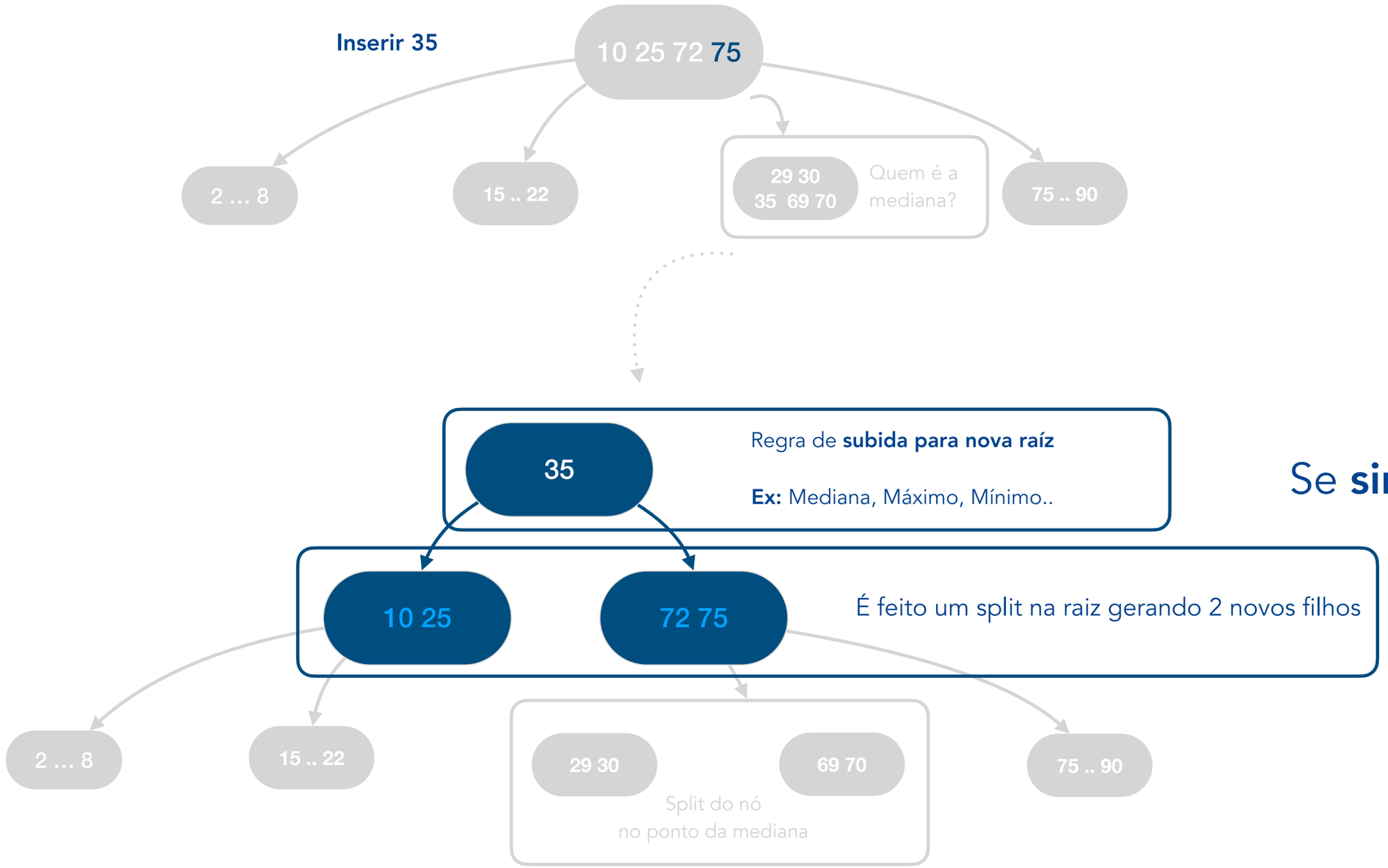

**Caso1 (raíz):**

Se **não** então **Insere**

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Inserção + Split da raíz**




**Caso2 (raíz):**
Se **sim** então **split da raíz**

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B

**Busca**

Operação mais simples: Busca/Procura por um elemento armazenado na árvore B

**Inserção**

Busca-se manter as propriedades de funcionamento da árvore B, em especial que todas as folhas tenham a **mesma profundidade** e os nós tenham um **número máximo de chaves**

Processo de **split:** Único meio de uma árvore B crescer

**Altura:** A altura de uma árvore B aumenta em cima, em vez de embaixo

**Remoção**

É possível **eliminar** uma chave **de qualquer nó**, afetando também a construção de nós intermediários.

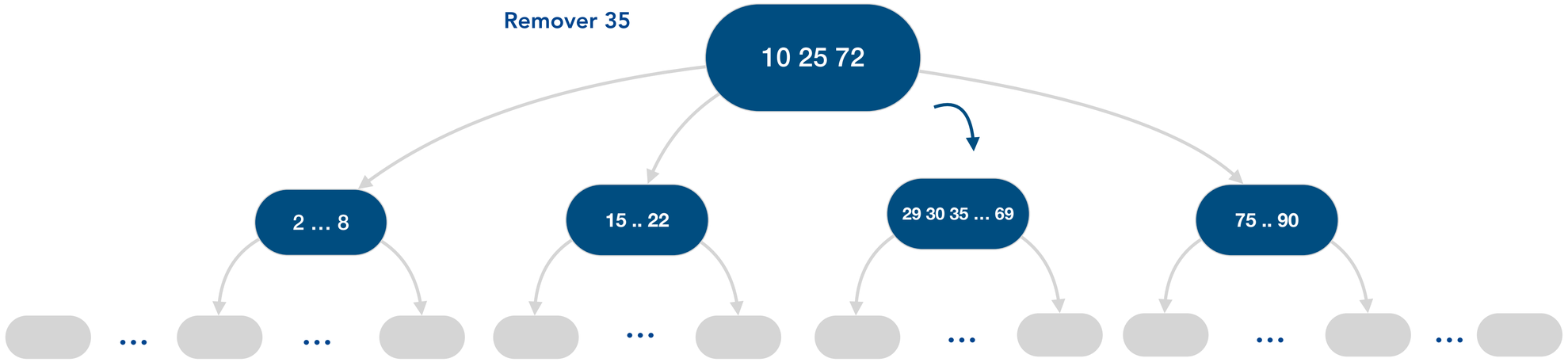
É preciso garantir que o processo não gere uma árvore cuja estrutura viole as propriedades de funcionamento, mantendo o **número mínimo de chaves** em um nó

São utilizados procedimentos de **redistribuição** de chaves entre nós, **fusões** ou movimentos de predecessores e sucessores

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

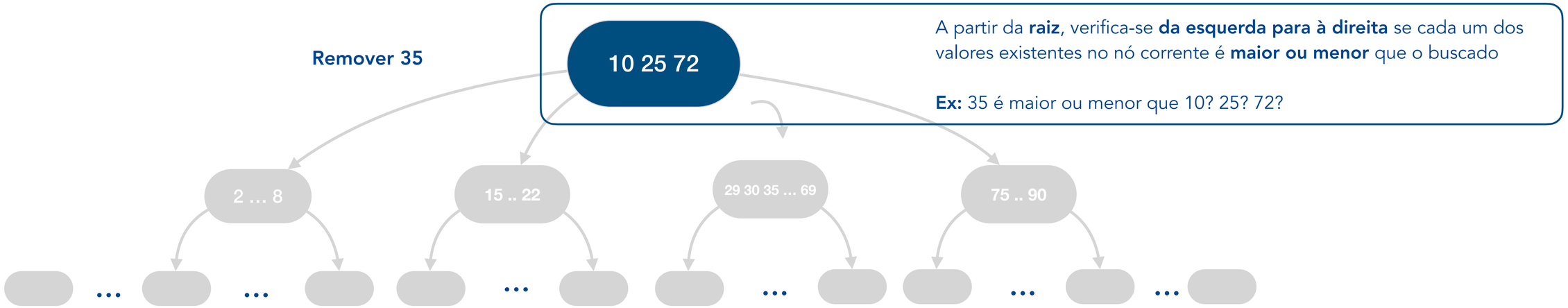
# Operações de uma árvore B: **Remoção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

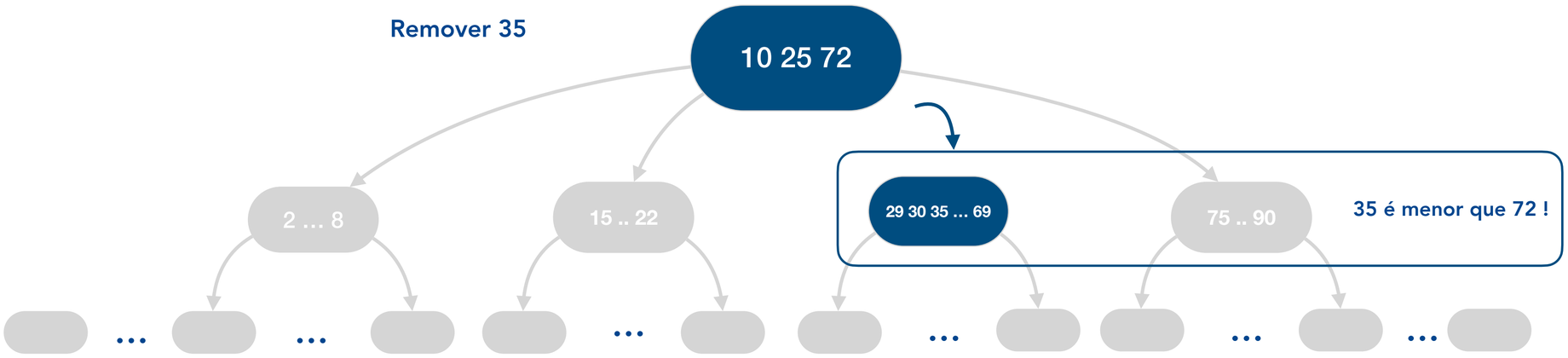
# Operações de uma árvore B: **Remoção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

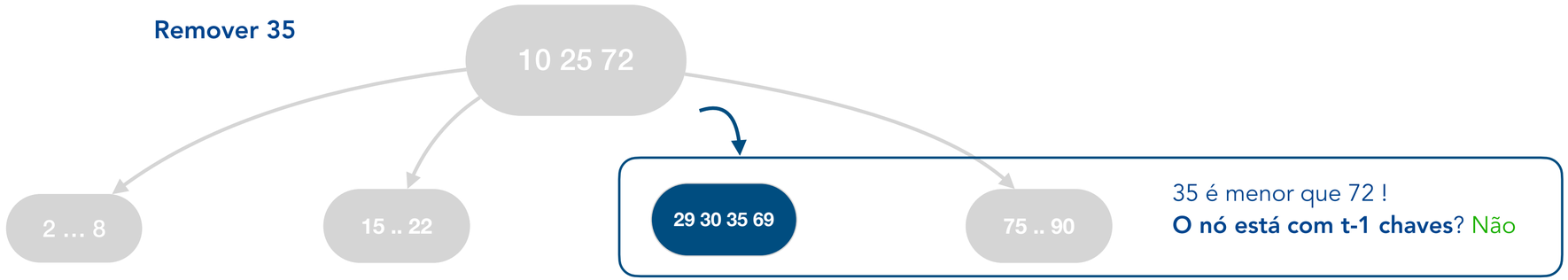
# Operações de uma árvore B: **Remoção**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Remoção simples (nó folha)**

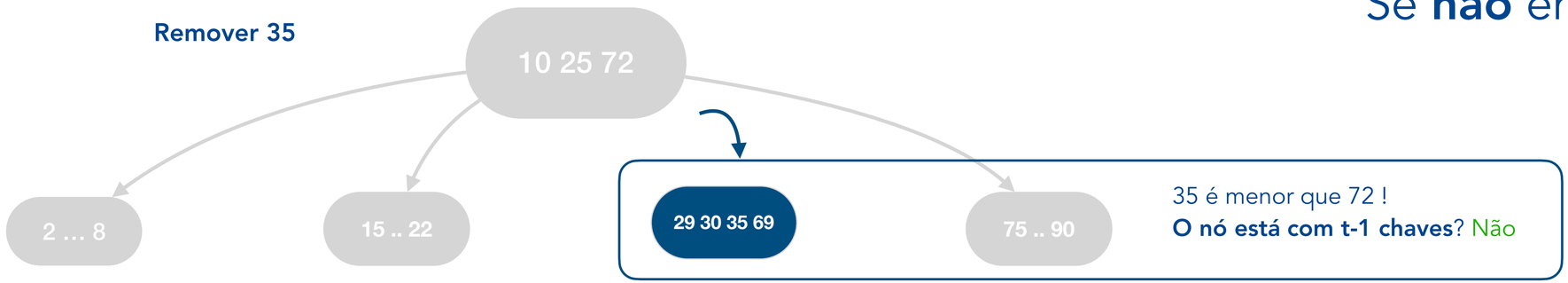



**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Remoção simples**

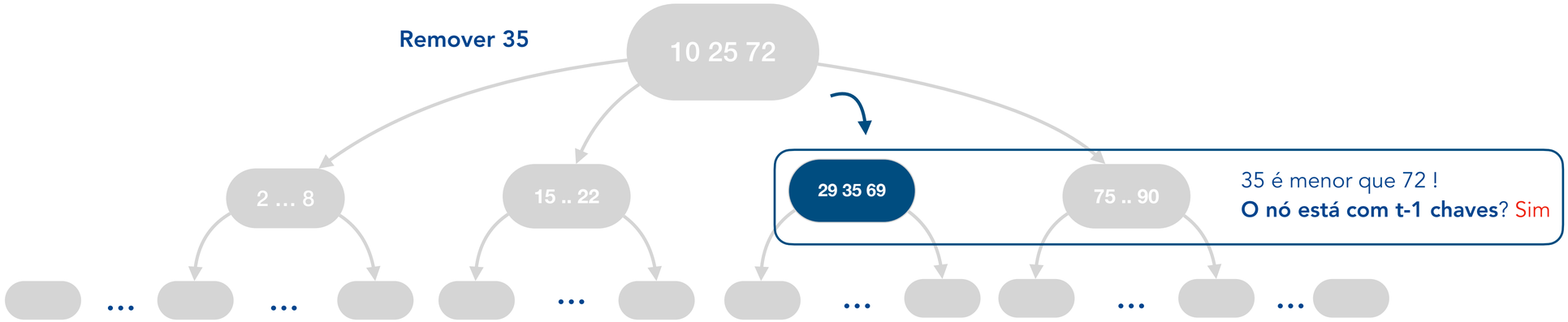
**Caso1:**
Se **não** então **remove**




**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Operações de uma árvore B: **Remoção em outros casos**




## Caso de remoção em nó interno

a) Trocas pelo predecessor ou sucessor

## Caso de remoção em folha com mínimo de ocupação

a) Redistribuição das chaves (Irmãos podem ceder uma chave)
b) Fusão de folhas (Irmãos com ocupação mínima)

**Chaves**

| Máximo | Mínimo |
|---|---|
| **2t - 1** | **t-1** |

# Implementação

**Operações:** Montagem da árvore **+** Operações de Busca, Inserção e Remoção

**Plataforma:** binder

**Características:** Utilização de um notebook **JupyterLab versão 3.6.6** com **4GB** de memória **RAM**.
Para a utilização do **kernel C++17**, foi importada a estrutura **conda-forge** e a biblioteca **xeus-cling 0.15.1**
Para contabilizar o tempo de execução utilizou-se a biblioteca **chrono** (cplusplus.com)

**Avaliação dos resultados:** Tempo médio e desvio padrão entre **100 execuções**
Variando-se tanto a **ordem** da árvore B, quanto o **número de chaves**.

# Complexidade / Tempo de execução

**Busca**

**Melhor e Pior Caso:** $O(h) = O(t \,.\, h) = O(t \,.\, log_t n) \propto O(log_t n)$ $\theta(log_t n)$

**Inserção**

**Melhor Caso:** Precisa localizar o nó para inserção da chave em uma única passada $O(log_t n)$

**Pior Caso:** Além de localizar o nó, precisa fazer split até a raiz $O(2. \log_t n) \propto O(log_t n)$

**Remoção**

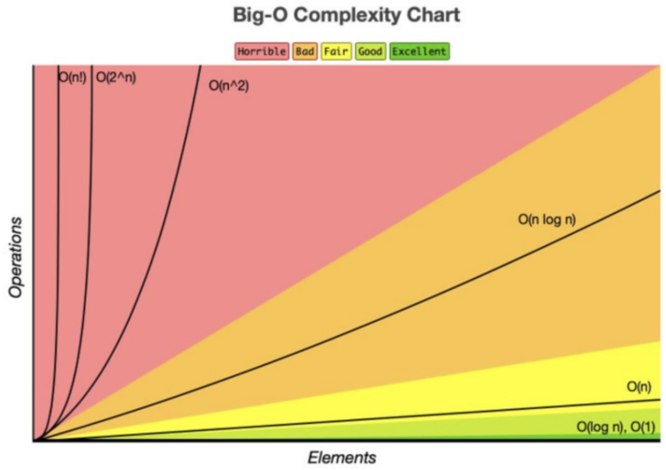
**Melhor Caso:** Precisa localizar o nó para remoção da chave em uma única passada $O(log_t n)$

**Pior Caso:** Na prática, a remoção ocorrerá majoritariamente nas folhas $O(k \,.\, \log_t n) \propto O(log_t n)$

**Montagem**

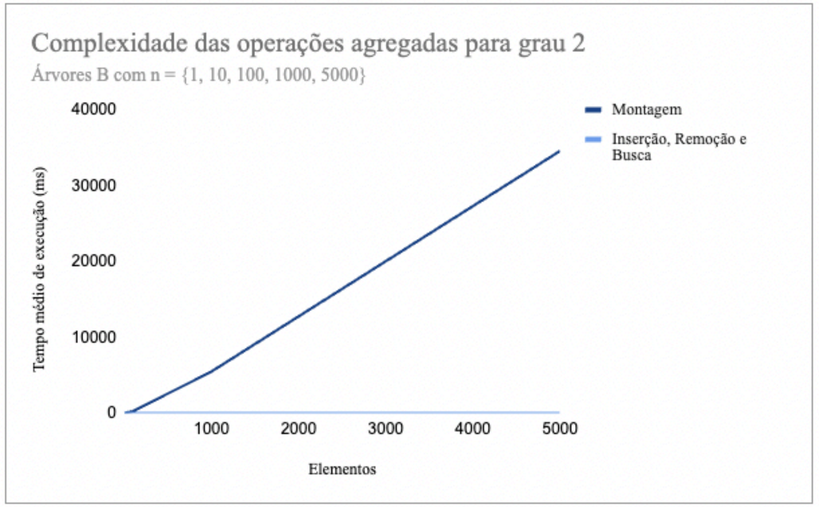
Realizamos **N vezes** o processo de **inserção** $O(n \,.\, \log_t n)$

# Complexidade / Tempo de execução: **Tipo de operação**



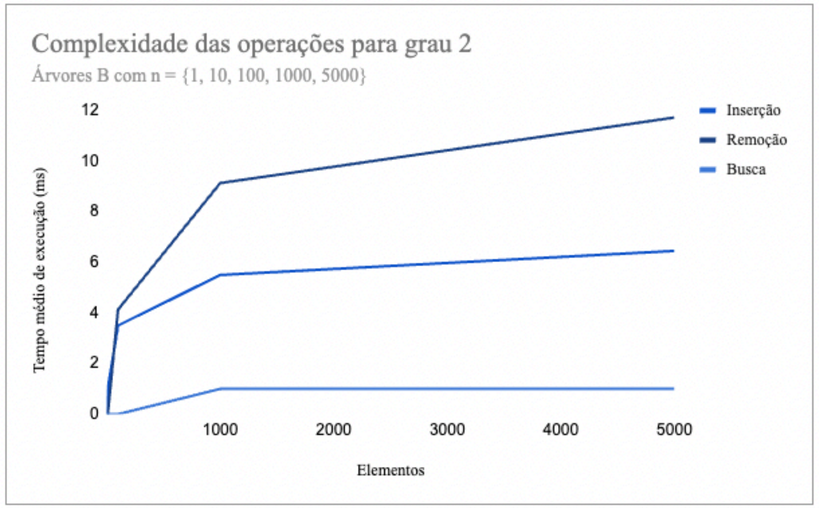

Montagem

**Mais Linear**




Operações comuns

**Log**

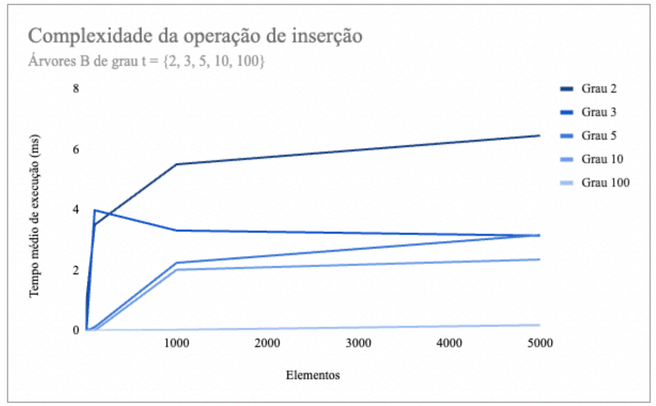


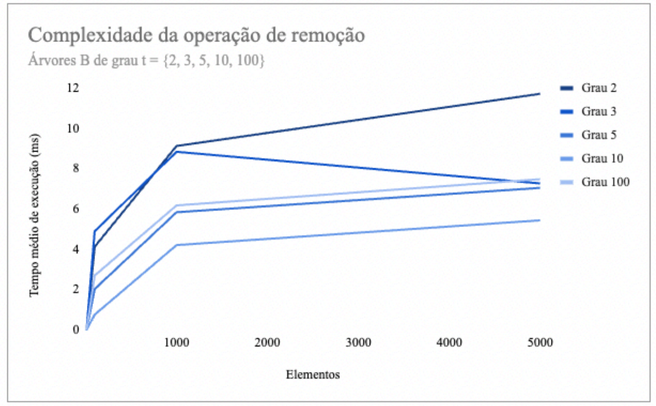
# Complexidade / Tempo de execução: **Grau da árvore**

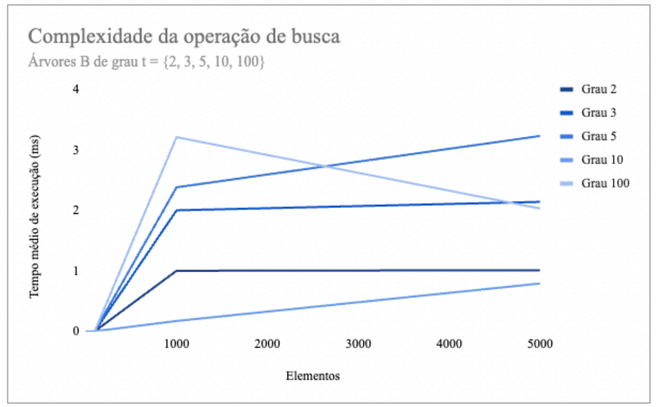
A escolha do **grau da árvore** pode **influenciar no tempo de execução** ao passo que aumentamos o número de chaves

**Personalização:** Possibilidade de adequar o grau a **necessidade da sua aplicação**

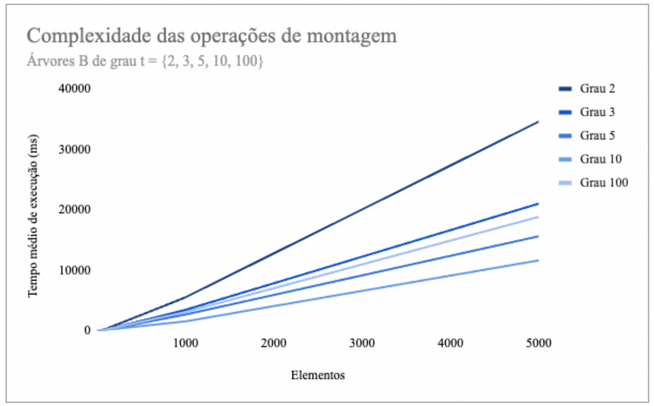
Operações comuns
**Log**




Operações comuns
**Log**




Montagem
**Mais Linear**

# Conclusão

**Tema**

Apresentamos e discutimos em detalhes as **características da estrutura de árvores B**

**Objetivo**

Ser uma **alternativa computacionalmente eficiente** de armazenamento em memória secundária

**Resultados**

Sugerem que os tempos de execução das operações de busca, inserção e remoção no geral possuem comportamentos muito similares **(Complexidade proporcional a $O(log_t n)$ )**

Deve-se atentar ao **definir o grau da árvore** que irá se trabalhar por conta da **influência** no desempenho

**Trabalhos futuros**

Possível alternativa de tema seria a evolução para as **árvores B+**

# Agradecimentos

Este trabalho contou com a colaboração do material disponibilizado pelo professor **Marcos Didonet Del Fabro** da **UFPR** no github

Agradecemos também o professor **Igor Machado Coelho** da **UFF** pelo apoio ao longo da construção do trabalho

Qualquer sugestão ou crítica, por favor entre em contato através dos canais abaixo

E-mails: lucasroberto@id.uff.br | mary.binha.mb@gmail.com

github: @lucasroberto | @marianesantos