

# MC-202

## Árvores B

Lehilton Pedrosa

Universidade Estadual de Campinas

Segundo semestre de 2021

# Introdução

**Um problema:** Trabalhamos com 1.000.000 de registros e cada um pode ser muito grande (uma foto, por exemplo). Portanto, não podemos guardá-los todos na memória. Toda vez que executamos um programa, temos que executar cerca de 1000 consultas nesse banco de dados.

# Introdução

**Um problema:** Trabalhamos com 1.000.000 de registros e cada um pode ser muito grande (uma foto, por exemplo). Portanto, não podemos guardá-los todos na memória. Toda vez que executamos um programa, temos que executar cerca de 1000 consultas nesse banco de dados.

- Onde armazenar os dados?

# Introdução

**Um problema:** Trabalhamos com 1.000.000 de registros e cada um pode ser muito grande (uma foto, por exemplo). Portanto, não podemos guardá-los todos na memória. Toda vez que executamos um programa, temos que executar cerca de 1000 consultas nesse banco de dados.

- Onde armazenar os dados?

# Introdução

**Um problema:** Trabalhamos com 1.000.000 de registros e cada um pode ser muito grande (uma foto, por exemplo). Portanto, não podemos guardá-los todos na memória. Toda vez que executamos um programa, temos que executar cerca de 1000 consultas nesse banco de dados.

- Onde armazenar os dados?
- Qual estrutura de dados?

# Introdução

**Um problema:** Trabalhamos com 1.000.000 de registros e cada um pode ser muito grande (uma foto, por exemplo). Portanto, não podemos guardá-los todos na memória. Toda vez que executamos um programa, temos que executar cerca de 1000 consultas nesse banco de dados.

- Onde armazenar os dados?
- Qual estrutura de dados?

**Tentativa:** usar uma árvore binária de busca balanceada no disco

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**



## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  **$\log_2(1.000.000) \approx 20$**  nós

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  **$\log_2(1.000.000) \approx 20$**  nós

TEMPO

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  **$\log_2(1.000.000) \approx 20$**  nós

TEMPO = 1000 buscas

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  **$\log_2(1.000.000) \approx 20$**  nós

$$\text{TEMPO} = 1000 \text{ buscas} \times 20 \text{ nós/busca}$$

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  $\log_2(1.000.000) \approx 20$  nós

$$\text{TEMPO} = 1000 \text{ buscas} \times 20 \text{ nós/busca} \times 5 \text{ ms/nó}$$

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  $\log_2(1.000.000) \approx 20$  nós

$$\text{TEMPO} = 1000 \text{ buscas} \times 20 \text{ nós/busca} \times 5 \text{ ms/nó} = 100 \text{ s}$$

## Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1000 consultas?

- ler um nó no disco pode demorar **5 ms**
- a árvore tem **1.000.000** de nós
- a altura é de  $\log_2(1.000.000) \approx 20$  nós

$$\text{TEMPO} = 1000 \text{ buscas} \times 20 \text{ nós/busca} \times 5 \text{ ms/nó} = 100 \text{ s}$$

**Solução:** diminuir a altura da árvore para diminuir número de leituras no disco



# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução
    - e a memória alocada pelos mesmos

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução
    - e a memória alocada pelos mesmos
  - Memória volátil, é apagada se o computador é desligado



# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução
    - e a memória alocada pelos mesmos
  - Memória volátil, é apagada se o computador é desligado
- **Memória Cache**

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução
    - e a memória alocada pelos mesmos
  - Memória volátil, é apagada se o computador é desligado
- **Memória Cache**
  - Muito próxima do processador para ter acesso rápido

# Hierarquia de Memória

A memória do computador é dividida em uma hierarquia:

- **HDD** (*Hard Disk Drive*) ou **SSD** (*Solid-State Drive*)
  - Memória permanente, onde gravamos arquivos
  - Chamada de memória secundária
- **RAM** (*Random-Access Memory*)
  - Onde são armazenados os programas em execução
    - e a memória alocada pelos mesmos
  - Memória volátil, é apagada se o computador é desligado
- **Memória Cache**
  - Muito próxima do processador para ter acesso rápido
  - A informação é copiada da RAM para a Cache

# Comparação entre Memórias

Velocidade

Tamanho

US\$ por GB

---

<sup>1</sup>em um processador 2GHz

## Comparação entre Memórias

	Velocidade	Tamanho	US\$ por GB
<b>HDD</b>	até 200 MB/s	até 4TB	0,05

---

<sup>1</sup>em um processador 2GHz

## Comparação entre Memórias

	Velocidade	Tamanho	US\$ por GB
<b>HDD</b>	até 200 MB/s	até 4TB	0,05
<b>SSD</b>	200 a 2500 MB/s	até 512 GB	0,3

---

<sup>1</sup>em um processador 2GHz

## Comparação entre Memórias

	Velocidade	Tamanho	US\$ por GB
<b>HDD</b>	até 200 MB/s	até 4TB	0,05
<b>SSD</b>	200 a 2500 MB/s	até 512 GB	0,3
<b>RAM</b>	2 a 20 GB/s	até 64 GB	7,5

---

<sup>1</sup>em um processador 2GHz

## Comparação entre Memórias

	Velocidade	Tamanho	US\$ por GB
<b>HDD</b>	até 200 MB/s	até 4TB	0,05
<b>SSD</b>	200 a 2500 MB/s	até 512 GB	0,3
<b>RAM</b>	2 a 20 GB/s	até 64 GB	7,5
<b>Cache</b>	32 a 64 GB/s <sup>1</sup>	até 25 MB	não é vendida

---

<sup>1</sup>em um processador 2GHz



# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB
- Se a página está na memória, podemos acessá-la

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB
- Se a página está na memória, podemos acessá-la
- Se não está, precisamos lê-la na memória secundária

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB
- Se a página está na memória, podemos acessá-la
- Se não está, precisamos lê-la na memória secundária
- O acesso a memória secundária é muito mais lento



# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB
- Se a página está na memória, podemos acessá-la
- Se não está, precisamos lê-la na memória secundária
- O acesso a memória secundária é muito mais lento
  - queremos ler o menor número de páginas possível

# Estruturas em Disco e Páginas

Queremos armazenar registros na memória secundária:

- A informação não cabe na memória principal
  - ou queremos que a informação seja permanente
- A memória secundária é dividida em **páginas**
  - usualmente de 2MB a 16MB
- Se a página está na memória, podemos acessá-la
- Se não está, precisamos lê-la na memória secundária
- O acesso a memória secundária é muito mais lento
  - queremos ler o menor número de páginas possível
  - acessar páginas que estão na memória é rápido

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:



# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo explícito

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo explícito
  - E que cada passo possa ser feito pelo computador

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo explícito
  - E que cada passo possa ser feito pelo computador

Se  $x$  é ponteiro para um objeto na memória secundária

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo explícito
  - E que cada passo possa ser feito pelo computador

Se  $x$  é ponteiro para um objeto na memória secundária

- **LEDoDISCO( $x$ )**: lê  $x$  da memória secundária

# Pseudocódigo e leitura/escrita de páginas

Usaremos **pseudocódigo** para apresentar a ED:

- Transmitem a ideia principal de um algoritmo
- Não há preocupação com detalhes de implementação
  - são agnósticos em relação a linguagem de programação
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo explícito
  - E que cada passo possa ser feito pelo computador

Se  $x$  é ponteiro para um objeto na memória secundária

- **LEDoDISCO( $x$ )**: lê  $x$  da memória secundária
- **ESCREVENoDISCO( $x$ )**: grava  $x$  na memória secundária

# Árvores $M$ -árias de Busca

Podemos generalizar árvores binárias de busca

# Árvores $M$ -árias de Busca

Podemos generalizar árvores binárias de busca

- Ex: árvores ternárias de busca

# Árvores $M$ -árias de Busca

Podemos generalizar árvores binárias de busca

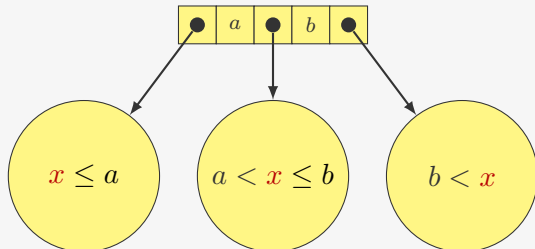
- Ex: árvores ternárias de busca
  - Nó pode ter 0, 1, 2 ou 3 filhos



# Árvores $M$ -árias de Busca

Podemos generalizar árvores binárias de busca

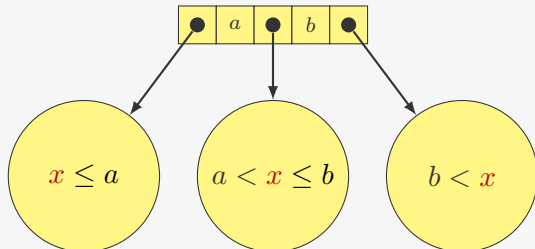
- Ex: árvores ternárias de busca
  - Nó pode ter 0, 1, 2 ou 3 filhos



# Árvores $M$ -árias de Busca

Podemos generalizar árvores binárias de busca

- Ex: árvores ternárias de busca
  - Nó pode ter 0, 1, 2 ou 3 filhos



Como fazer busca?

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não



# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho
- se a chave  $k$  está na subárvore  $x.c[i]$ , então

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho
- se a chave  $k$  está na subárvore  $x.c[i]$ , então
  - $k < x.chave[1]$  se  $i = 1$

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho
- se a chave  $k$  está na subárvore  $x.c[i]$ , então
  - $k < x.chave[1]$  se  $i = 1$
  - $x.chave[x.n] < k$  se  $i = x.n + 1$

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho
- se a chave  $k$  está na subárvore  $x.c[i]$ , então
  - $k < x.chave[1]$  se  $i = 1$
  - $x.chave[x.n] < k$  se  $i = x.n + 1$
  - $x.chave[i-1] < k < x.chave[i]$  caso contrário

# Árvores B

São árvores  $M$ -árias de busca com propriedades adicionais

Cada nó  $x$  tem os seguintes campos:

- $x.n$  é o número de chaves armazenadas em  $x$
- $x.chave[i]$  é  $i$ -ésima chave armazenada
  - $x.chave[1] < x.chave[2] < \dots < x.chave[x.n]$
- $x.folha$  indica se  $x$  é uma folha ou não

Cada nó interno  $x$  contém  $x.n + 1$  ponteiros

- $x.c[i]$  é o ponteiro para o  $i$ -ésimo filho
- se a chave  $k$  está na subárvore  $x.c[i]$ , então
  - $k < x.chave[1]$  se  $i = 1$
  - $x.chave[x.n] < k$  se  $i = x.n + 1$
  - $x.chave[i-1] < k < x.chave[i]$  caso contrário

O  $T.raiz$  indica o nó que é a raiz da árvore

# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz



# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

Existe uma constante  $t$  que é o grau mínimo da árvore

# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

Existe uma constante  $t$  que é o grau mínimo da árvore

- Todo nó exceto a raiz precisa ter pelo menos  $t - 1$  chaves

# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

Existe uma constante  $t$  que é o grau mínimo da árvore

- Todo nó exceto a raiz precisa ter pelo menos  $t - 1$  chaves
  - ou seja, cada nó interno tem pelo menos  $t$  filhos

# Propriedades das Árvores B

Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

Existe uma constante  $t$  que é o grau mínimo da árvore

- Todo nó exceto a raiz precisa ter pelo menos  $t - 1$  chaves
  - ou seja, cada nó interno tem pelo menos  $t$  filhos
- Todo nó tem no máximo  $2t - 1$  chaves

# Propriedades das Árvores B

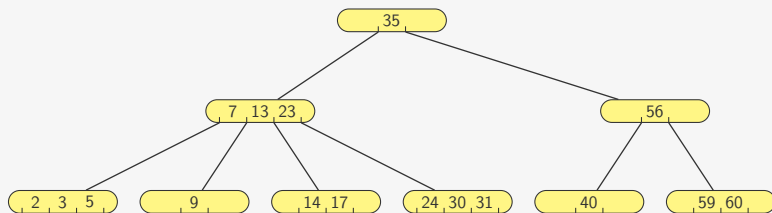
Toda folha está à mesma distância  $h$  da raiz

- $h$  é a altura da árvore

Existe uma constante  $t$  que é o grau mínimo da árvore

- Todo nó exceto a raiz precisa ter pelo menos  $t - 1$  chaves
  - ou seja, cada nó interno tem pelo menos  $t$  filhos
- Todo nó tem no máximo  $2t - 1$  chaves
  - ou seja, cada nó interno tem no máximo  $2t$  filhos

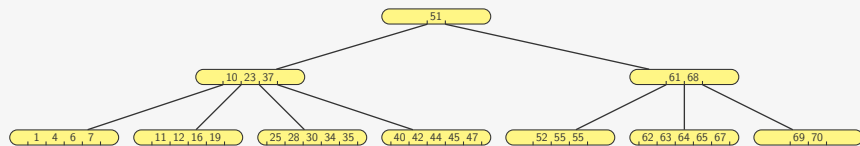
## Exemplo



Para  $t = 2$ :

- cada nó não raiz tem pelo menos 1 registro
- cada nó tem no máximo 3 registros

## Outro exemplo



Para  $t = 3$ :

- cada nó não raiz tem pelo menos 2 registros
- cada nó tem no máximo 5 registros



## Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

## Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

- a raiz tem pelo menos 2 filhos

## Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

- a raiz tem pelo menos  $2$  filhos
- esses filhos têm pelo menos  $2t$  filhos

## Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

- a raiz tem pelo menos  $2$  filhos
- esses filhos têm pelo menos  $2t$  filhos
- que têm pelo menos  $2t^2$  filhos

# Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

- a raiz tem pelo menos  $2$  filhos
- esses filhos têm pelo menos  $2t$  filhos
- que têm pelo menos  $2t^2$  filhos
- e assim por diante

# Altura de uma Árvore B

Uma árvore  $B$  com  $n$  chaves tem altura  $h \leq \log_t \frac{n+1}{2}$

- a raiz tem pelo menos  $2$  filhos
- esses filhos têm pelo menos  $2t$  filhos
- que têm pelo menos  $2t^2$  filhos
- e assim por diante

A árvore é muito **larga** e muito **baixa**!

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco



## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Quando  $t = 2$ , temos as *Árvores 2 – 3 – 4*

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Quando  $t = 2$ , temos as **Árvores 2 – 3 – 4**

- Equivalentes às árvores **rubro-negras**

## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

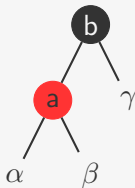
- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Quando  $t = 2$ , temos as **Árvores 2 – 3 – 4**

- Equivalentes às árvores **rubro-negras**





## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

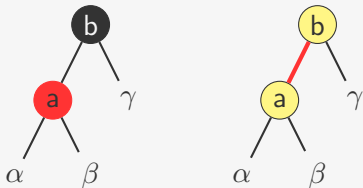
- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Quando  $t = 2$ , temos as **Árvores 2 – 3 – 4**

- Equivalentes às árvores **rubro-negras**



## Escolhendo $t$

Queremos que um nó caiba em uma página do disco

- mas não queremos utilizar mal a página do disco

Escolha  $t$  máximo tal que  $2t - 1$  chaves caibam na página

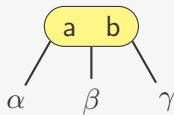
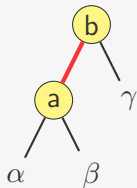
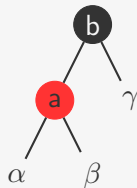
- Se  $t = 1001$  e  $h = 2$ , armazenamos até  $10^9$  chaves
- i.e., fazemos dois acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Quando  $t = 2$ , temos as **Árvores 2 – 3 – 4**

- Equivalentes às árvores **rubro-negras**



## Busca na Árvore B

Para procurar a chave  $k$  no nó  $x$

## Busca na Árvore B

Para procurar a chave  $k$  no nó  $x$

- Basta verificar se a chave está em  $x$

## Busca na Árvore B

Para procurar a chave  $k$  no nó  $x$

- Basta verificar se a chave está em  $x$
- Se não estiver, basta buscar no filho correto

## Busca na Árvore B

Para procurar a chave  $k$  no nó  $x$

- Basta verificar se a chave está em  $x$
- Se não estiver, basta buscar no filho correto

# Busca na Árvore B

Para procurar a chave  $k$  no nó  $x$

- Basta verificar se a chave está em  $x$
- Se não estiver, basta buscar no filho correto

**BUSCA**( $x, k$ )

```
1   $i = 1$ 
2  enquanto  $i \leq x.n$  e  $k > x.chave[i]$ 
3     $i = i + 1$ 
4  se  $i \leq x.n$  e  $k == x.chave[i]$ 
5    retorne  $(x, i)$ 
6  senão se  $x.folha$ 
7    retorne NIL
8  senão
9    LEDODISCO( $x.c[i]$ )
10  retorne BUSCA( $x.c[i], k$ )
```

# Criando uma Árvore B

Criamos uma árvore vazia



# Criando uma Árvore B

Criamos uma árvore vazia

- Basta alocar o nó e definir os campos

# Criando uma Árvore B

Criamos uma árvore vazia

- Basta alocar o nó e definir os campos

# Criando uma Árvore B

Criamos uma árvore vazia

- Basta alocar o nó e definir os campos

**INICIA**(*T*)

```
1 x = ALOCA()
2 x.folha = VERDADEIRO
3 x.n = 0
4 ESCREVENODISCO(x)
5 T.raiz = x
```

# Inserção

A inserção ocorre sempre em um nó folha

## Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )

## Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )

## Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves

# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra



# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra
  - mas o pai poderia estar cheio...

# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra
  - mas o pai poderia estar cheio...
- dividimos todo nó cheio no caminho a inserção

# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra
  - mas o pai poderia estar cheio...
- dividimos todo nó cheio no caminho a inserção
  - assim, o pai nunca estará cheio

# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra
  - mas o pai poderia estar cheio...
- dividimos todo nó cheio no caminho a inserção
  - assim, o pai nunca estará cheio

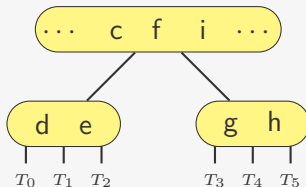
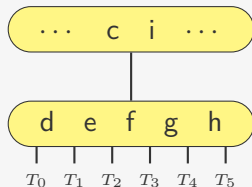
Exemplo:  $t = 3$

# Inserção

A inserção ocorre sempre em um nó folha

- porém, o nó folha pode estar cheio ( $x.n == 2t - 1$ )
- dividimos o nó na chave mediana ( $x.chave[t]$ )
  - em dois nós com  $t - 1$  chaves
  - inserimos  $x.chave[t]$  no pai para representar a quebra
  - mas o pai poderia estar cheio...
- dividimos todo nó cheio no caminho a inserção
  - assim, o pai nunca estará cheio

Exemplo:  $t = 3$



# Dividindo um nó

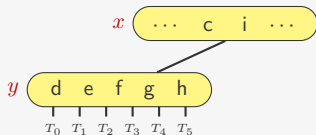
**DIVIDEFILHO**( $x, i$ )

```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9       $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13  $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16  $x.chave[i] = y.chave[t]$ 
17  $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```

# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

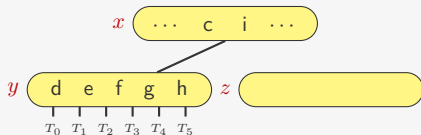
```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9     $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17    $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```



# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9     $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17    $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```

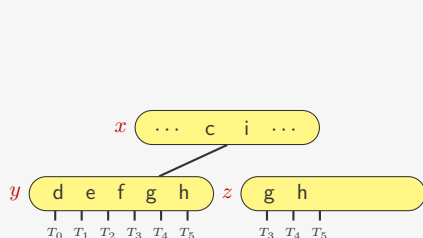




# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

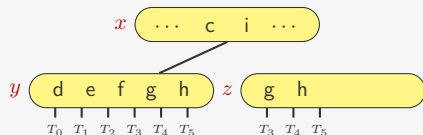
```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9     $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17    $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```



# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

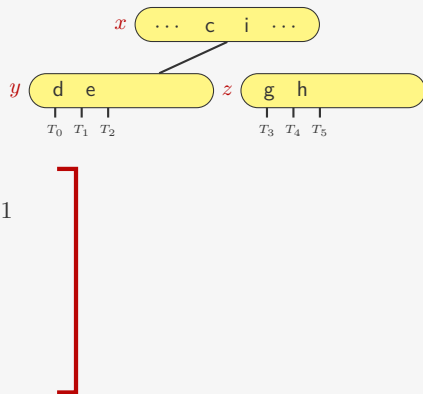
```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9       $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17    $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```



# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

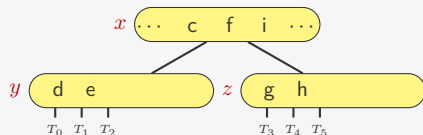
```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9       $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17  $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```



# Dividindo um nó

**DIVIDEFILHO**( $x, i$ )

```
1   $z = \text{ALOCA}()$ 
2   $y = x.c[i]$ 
3   $z.folha = y.folha$ 
4   $z.n = t - 1$ 
5  para  $j = 1$  até  $t - 1$ 
6     $z.chave[j] = y.chave[j + t]$ 
7  se não  $y.folha$ 
8    para  $j = 1$  até  $t$ 
9       $z.c[j] = y.c[j + t]$ 
10  $y.n = t - 1$ 
11 para  $j = x.n + 1$  decrecendo até  $i + 1$ 
12    $x.c[j + 1] = x.c[j]$ 
13    $x.c[i + 1] = z$ 
14 para  $j = x.n$  decrecendo até  $i$ 
15    $x.chave[j + 1] = x.chave[j]$ 
16    $x.chave[i] = y.chave[t]$ 
17  $x.n = x.n + 1$ 
18 ESCREVENODISCO( $y$ )
19 ESCREVENODISCO( $z$ )
20 ESCREVENODISCO( $x$ )
```



# Inserindo

Vamos inserir a chave  $k$  na árvore  $T$

- verificamos se não é necessário dividir a raiz

**INSERE**( $T, k$ )

```
1   $r = T.raiz$ 
2  se  $r.n == 2t - 1$ 
3     $s = ALOCA()$ 
4     $T.raiz = s$ 
5     $s.folha = FALSO$ 
6     $s.n = 0$ 
7     $s.c[1] = r$ 
8    DIVIDEFILHO( $s, 1$ )
9    INSERENÃOCHEIO( $s, k$ )
10 senão
11   INSERENÃOCHEIO( $r, k$ )
```

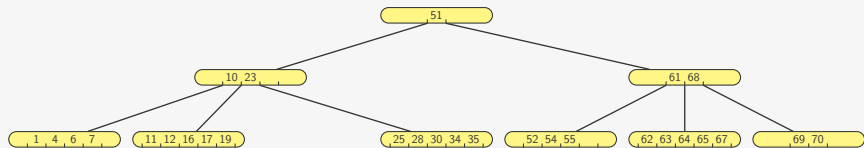
## Inserindo chave $k$ em um nó não-cheio $x$

**INSERENÃOCHIEIO**( $x, k$ )

```
1   $i = x.n$ 
2  se  $x.folha$ 
3      enquanto  $i \geq 1$  e  $k < x.chave[i]$ 
4           $x.chave[i + 1] = x.chave[i]$ 
5           $i = i - 1$ 
6       $x.chave[i + 1] = k$ 
7       $x.n = x.n + 1$ 
8      ESCREVENODISCO( $x$ )
9  senão
10     enquanto  $i \geq 1$  e  $k < x.chave[i]$ 
11          $i = i - 1$ 
12      $i = i + 1$ 
13     LEDODISCO( $x.c[i]$ )
14     se  $x.c[i].n == 2t - 1$ 
15         DIVIDEFILHO( $x, i$ )
16         se  $k > x.chave[i]$ 
17              $i = i + 1$ 
18     INSERENÃOCHIEIO( $x.c[i], k$ )
```

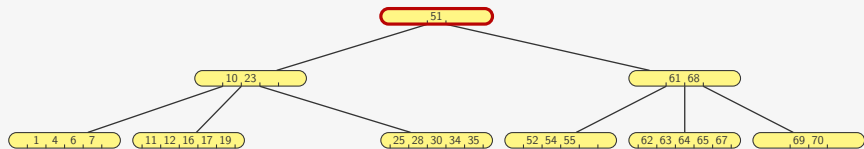
## Exemplo: inserindo em nó não cheio

Inserindo 53



# Exemplo: inserindo em nó não cheio

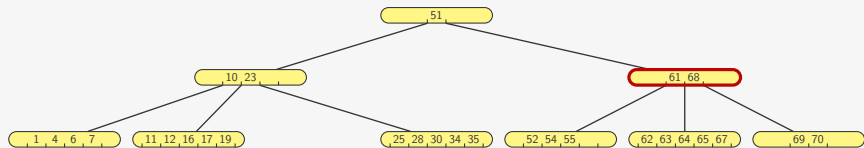
Inserindo 53





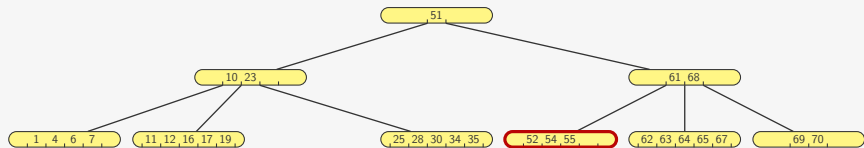
## Exemplo: inserindo em nó não cheio

Inserindo 53



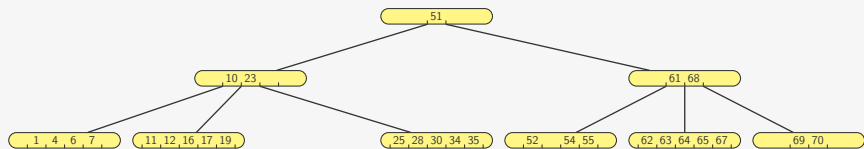
## Exemplo: inserindo em nó não cheio

Inserindo 53



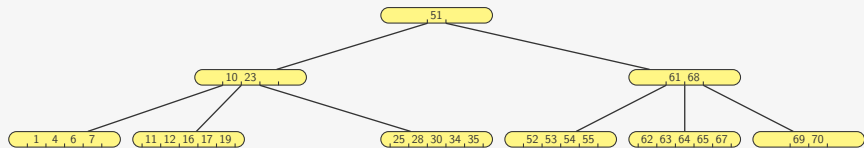
## Exemplo: inserindo em nó não cheio

Inserindo 53



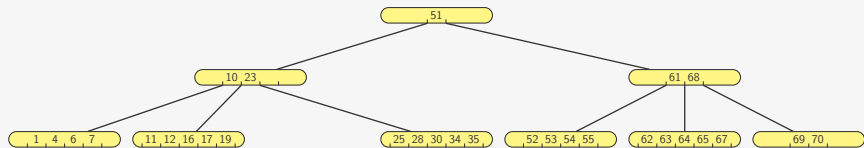
## Exemplo: inserindo em nó não cheio

Inserindo 53



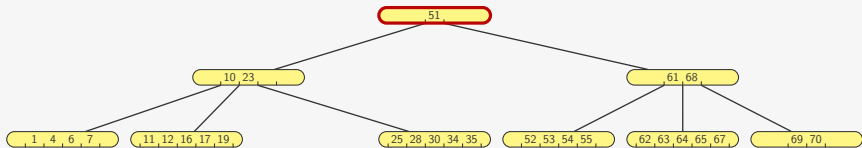
## Exemplo: inserindo em nó cheio

Inserindo 18



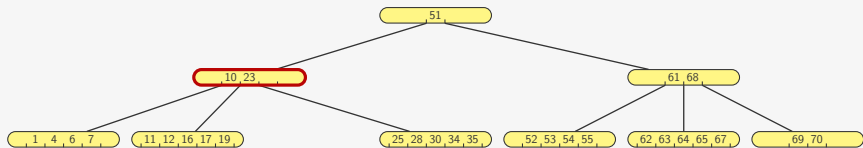
# Exemplo: inserindo em nó cheio

Inserindo 18



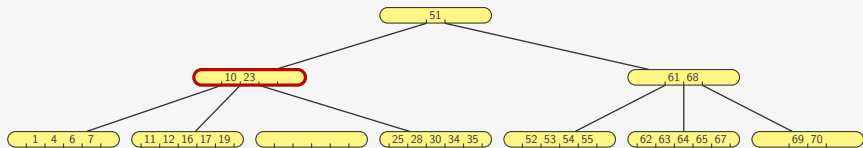
# Exemplo: inserindo em nó cheio

Inserindo 18



# Exemplo: inserindo em nó cheio

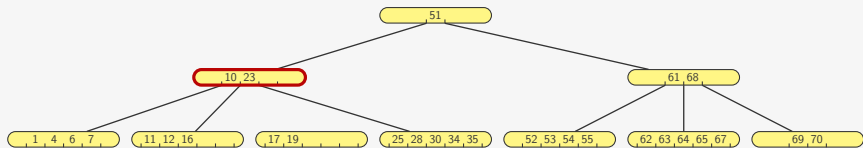
Inserindo 18





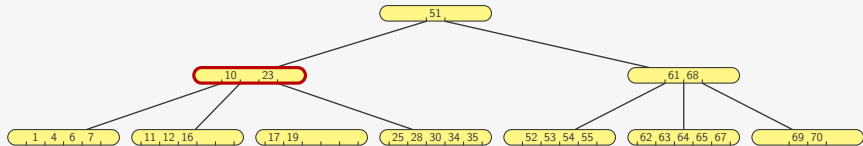
# Exemplo: inserindo em nó cheio

Inserindo 18



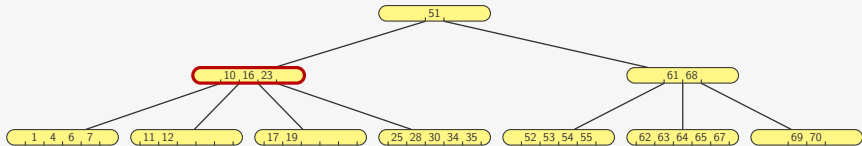
# Exemplo: inserindo em nó cheio

Inserindo 18



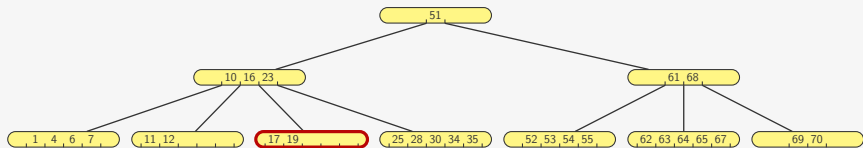
# Exemplo: inserindo em nó cheio

Inserindo 18



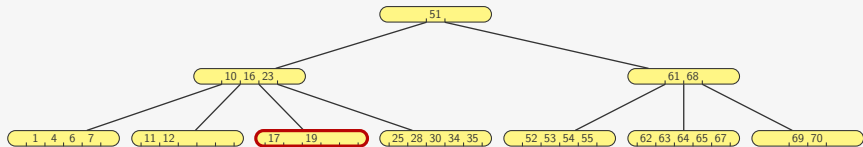
# Exemplo: inserindo em nó cheio

Inserindo 18



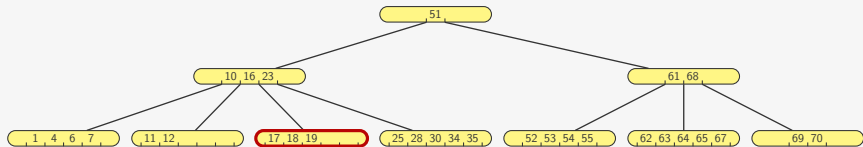
# Exemplo: inserindo em nó cheio

Inserindo 18



# Exemplo: inserindo em nó cheio

Inserindo 18



# Remoção

A remoção é mais complicada que a inserção

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore



# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

- nesse caso não há problema em remover

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

- nesse caso não há problema em remover
- se não houver, tentamos mover uma chave de um vizinho

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

- nesse caso não há problema em remover
- se não houver, tentamos mover uma chave de um vizinho
- nem sempre conseguimos

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

- nesse caso não há problema em remover
- se não houver, tentamos mover uma chave de um vizinho
- nem sempre conseguimos
  - quando cada um dos dois vizinhos tiver apenas  $t - 1$  chaves

# Remoção

A remoção é mais complicada que a inserção

- Ela pode ocorrer em qualquer lugar da árvore
- Cada nó precisa continuar com pelo menos  $t - 1$  chaves
  - exceto a raiz que tem que ter pelo menos  $1$  chave

Para resolver esse problema, garantimos que os nós no caminho da remoção têm pelo menos  $t$  chaves

- nesse caso não há problema em remover
- se não houver, tentamos mover uma chave de um vizinho
- nem sempre conseguimos
  - quando cada um dos dois vizinhos tiver apenas  $t - 1$  chaves
  - juntamos os nós formando um nó com  $2t - 1$  chaves



# Variantes

Árvores  $B^*$ :

# Variantes

Árvores  $B^*$ :

- Nós não raiz precisam ficar pelo menos  $2/3$  cheios

# Variantes

Árvores  $B^*$ :

- Nós não raiz precisam ficar pelo menos  $2/3$  cheios

Árvores  $B^+$ :

# Variantes

## Árvores $B^*$ :

- Nós não raiz precisam ficar pelo menos  $2/3$  cheios

## Árvores $B^+$ :

- Mantêm cópias das chaves nos nós internos, mas as chaves e os registros são armazenados nas folhas

# Variantes

## Árvores $B^*$ :

- Nós não raiz precisam ficar pelo menos  $2/3$  cheios

## Árvores $B^+$ :

- Mantêm cópias das chaves nos nós internos, mas as chaves e os registros são armazenados nas folhas
- Permite acesso sequencial dos dados

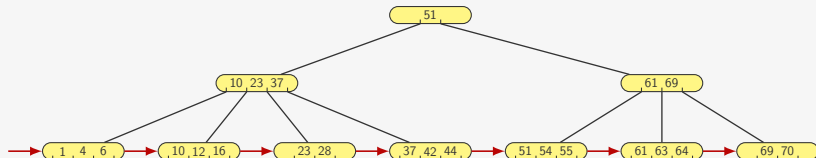
# Variantes

## Árvores $B^*$ :

- Nós não raiz precisam ficar pelo menos  $2/3$  cheios

## Árvores $B^+$ :

- Mantêm cópias das chaves nos nós internos, mas as chaves e os registros são armazenados nas folhas
- Permite acesso sequencial dos dados



## Exercício

Qual a árvore obtida após inserirmos sequencialmente os números 13 e 33 na árvore seguinte?

