Estruturas de Dados

Árvores B

Aula 15

Prof. Felipe A. Louza

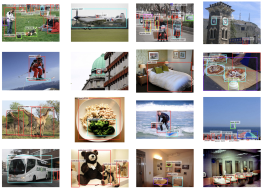
UFU

Roteiro

Roteiro

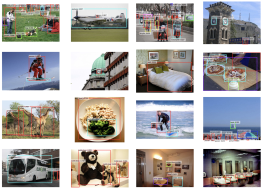
Introdução

Suponha que temos 1.000.000 de registros e cada um pode ser muito grande (uma foto $\approx$ 1MB, por exemplo)



Introdução

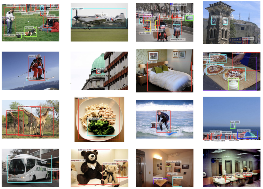
Suponha que temos 1.000.000 de registros e cada um pode ser muito grande (uma foto $\approx$ 1MB, por exemplo)



- Considere que **não temos** espaço suficiente para guardar todos os registros na memória ($\approx$ 1TB).

Introdução

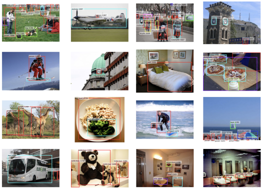
Suponha que temos 1.000.000 de registros e cada um pode ser muito grande (uma foto $\approx$ 1MB, por exemplo)



- Considere que **não temos** espaço suficiente para guardar todos os registros na memória ($\approx$ 1TB).
- Onde armazenar os dados?

Introdução

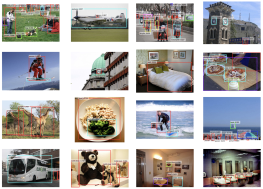
Suponha que temos 1.000.000 de registros e cada um pode ser muito grande (uma foto $\approx$ 1MB, por exemplo)



- Considere que **não temos** espaço suficiente para guardar todos os registros na memória ($\approx$ 1TB).
- Onde armazenar os dados?
- Qual estrutura de dados utilizar para realizar consultas?

Introdução

Suponha que temos 1.000.000 de registros e cada um pode ser muito grande (uma foto $\approx$ 1MB, por exemplo)



- Considere que **não temos** espaço suficiente para guardar todos os registros na memória ($\approx$ 1TB).
- Onde armazenar os dados?
- Qual estrutura de dados utilizar para realizar consultas?

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

Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1.000 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 1.000 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

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

Verificando nossa tentativa

Quanto tempo vai levar para realizar as 1.000 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

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

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:

- HDD (Hard Disk Drive) ou SSD (Solid-State Drive)
  - Memória **persistente**, 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 **persistente**, 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 **persistente**, 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 **persistente**, 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
HDD	até 200 MB/s	até 4TB	0,05
SSD	200 a 2500 MB/s	até 512 GB	0,3
RAM	2 a 20 GB/s	até 64 GB	7,5
Cache	32 a 64 GB/s[1]	até 25 MB	não é vendida

[1] em um processador 2GHz

Estruturas em Disco e Páginas

Registros na memória secundária:

- A memória secundária é **dividida** em páginas
  - usualmente de 2MB a 16MB

Estruturas em Disco e Páginas

Registros na memória secundária:

- A memória secundária é **dividida** em páginas
  - usualmente de 2MB a 16MB
- Se a página está na memória RAM, podemos acessá-la
- Se não está, precisamos **lê-la** na memória secundária

Estruturas em Disco e Páginas

Registros na memória secundária:

- A memória secundária é **dividida** em páginas
  - usualmente de 2MB a 16MB
- Se a página está na memória RAM, 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**
  - depois, acessar páginas que estão na memória é rápido

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
- 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
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo claro
  - 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
- É uma forma mais abstrata de falar de algoritmos
- Precisamos tomar o cuidado de:
  - Deixar o algoritmo claro
  - E que cada passo possa ser feito pelo computador

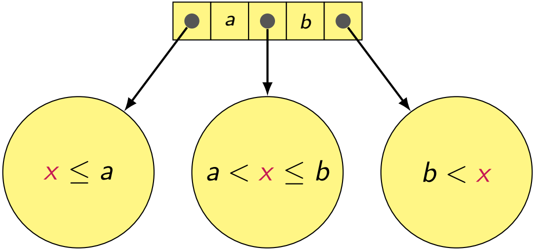
Se x é ponteiro para um objeto na memória secundária

- LeDoDisco(x): lê x da memória secundária
- EscreveNoDisco(x): grava x na memória secundária

Roteiro

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

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
  - $x.chave[1] < x.chave[2] < \cdots < x.chave[x.n]$
- $x.folha$ 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 $i = 1$
  - $k > x.chave[x.n]$ se $i = x.n + 1$
  - $x.chave[i\text{-}1] < k < x.chave[i]$ caso contrário

Árvores B

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
  - $x.chave[1] < x.chave[2] < \cdots < x.chave[x.n]$
- $x.folha$ 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 $i = 1$
  - $k > x.chave[x.n]$ se $i = x.n + 1$
  - $x.chave[i\text{-}1] < k < x.chave[i]$ 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

- 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

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

Árvores M-ária com no máximo $M = 2t$ filhos

Propriedades das Árvores B

Toda folha está à mesma distância h da raiz

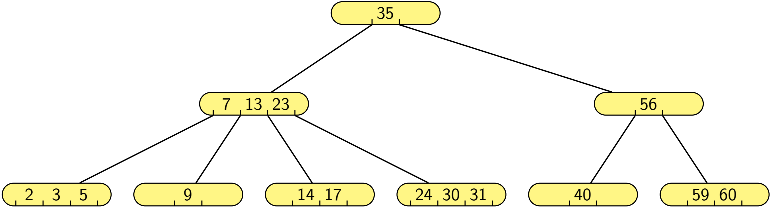
- h é a altura da árvore

Existe uma constante t que é o grau mínimo da árvore

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

Árvores M-ária com no máximo $M = 2t$ filhos

Exemplo



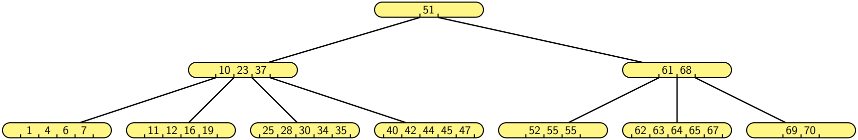
Para $t = 2$:

- cada nó não raiz tem no mínimo $t - 1 = 1$ registro
- cada nó tem no máximo $2t - 1 = 3$ registros

---

t é o grau mínimo da árvore

Outro exemplo



Para $t = 3$:

- cada nó não raiz tem no mínimo $t - 1 = 2$ registros
- cada nó tem no máximo $2t - 1 = 5$ registros

---

t é o grau mínimo da árvore

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 **até na altura** h, termos $2t^{h-1}$ nós

$h = O(log_t n)$

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 **até na altura** h, termos $2t^{h-1}$ nós

A árvore é muito larga e muito baixa!

$h = O(log_t n)$, se $n = 10^9$ e $t = 10^3$, $h = 2$

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 $t = 10^3$ e cada registro ocupa 1KB
- cada nó cabe em uma **página do disco** de 2MB

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 $t = 10^3$ e cada registro ocupa 1KB
- cada nó cabe em uma **página do disco** de 2MB
- para armazenamos até 10^9 registros ($\approx$ 950GB), teremos $h = 2$

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 $t = 10^3$ e cada registro ocupa 1KB
- cada nó cabe em uma **página do disco** de 2MB
- para armazenamos até 10^9 registros ($\approx$ 950GB), teremos $h = 2$
- i.e., fazemos <u>três</u> 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 $t = 10^3$ e cada registro ocupa 1KB
- cada nó cabe em uma **página do disco** de 2MB
- para armazenamos até 10^9 registros ($\approx$ 950GB), teremos $h = 2$
- i.e., fazemos <u>três</u> acessos ao disco

Consideramos que o registro está junto com a chave

- Ou então temos um ponteiro para o registro

Roteiro

Criando uma Árvore B

Criamos uma árvore vazia

- Basta alocar o nó e definir os campos

INICIA(T)

```
x = ALOCA()
x.folha = VERDADEIRO
x.n = 0
ESCREVENODISCO(x)
T.raiz = x
```

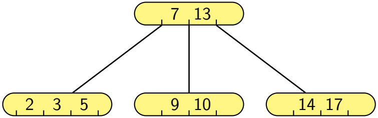
Busca na Árvore B

Para procurar a chave k no nó x

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

Busca(x, k)

```
i = 1
enquanto i ≤ x.n e x.chave[i] < k
    i = i + 1
se i ≤ x.n e k == x.chave[i]
    retorne (x, i)
senão se x.folha
    retorne NIL
senão
    LeDoDisco(x.c[i])
    retorne Busca(x.c[i], k)
```

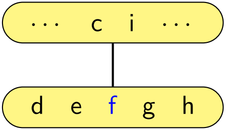


Inserção

A inserção ocorre sempre em um nó folha

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

Exemplo: $t = 3$

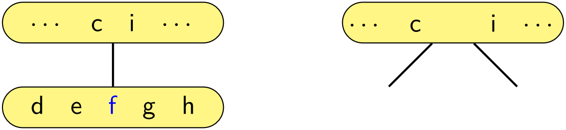


Inserção

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 ($x.chave[t]$)

Exemplo: $t = 3$

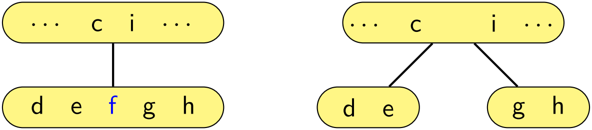


Inserção

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

Exemplo: $t = 3$

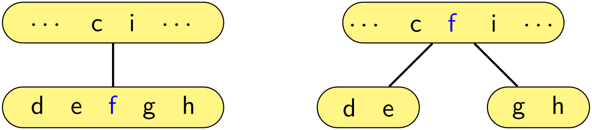


Inserção

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

Exemplo: $t = 3$

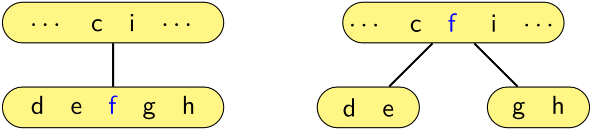


Inserção

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**...

Exemplo: $t = 3$

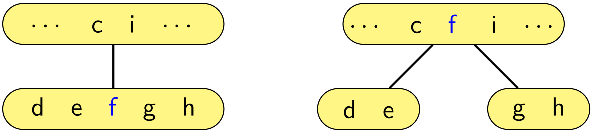


Inserção

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

Exemplo: $t = 3$

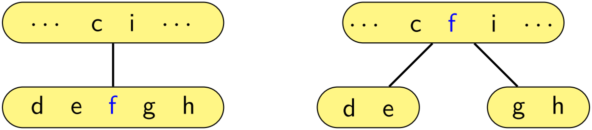


Inserção

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: $t = 3$

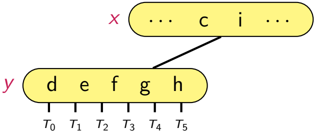


Dividindo um nó

```
DivideFilho(x, i)
z = Aloca()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
EscreveNoDisco(y)
EscreveNoDisco(z)
EscreveNoDisco(x)
```

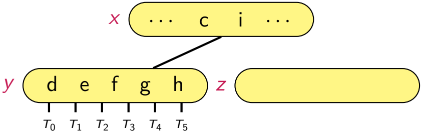
Dividindo um nó

```
DIVIDEFILHO(x, i)
z = ALOCA()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
ESCREVENODISCO(y)
ESCREVENODISCO(z)
ESCREVENODISCO(x)
```



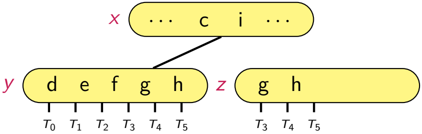
Dividindo um nó

```
DivideFilho(x, i)
z = Aloca()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
EscreveNoDisco(y)
EscreveNoDisco(z)
EscreveNoDisco(x)
```



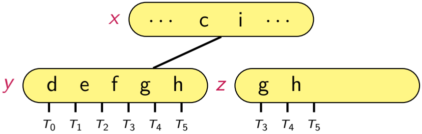
Dividindo um nó

```
DivideFilho(x, i)
z = Aloca()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
EscreveNoDisco(y)
EscreveNoDisco(z)
EscreveNoDisco(x)
```



Dividindo um nó

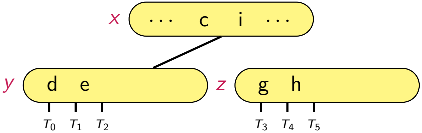
```
DivideFilho(x, i)
z = Aloca()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
EscreveNoDisco(y)
EscreveNoDisco(z)
EscreveNoDisco(x)
```



Dividindo um nó

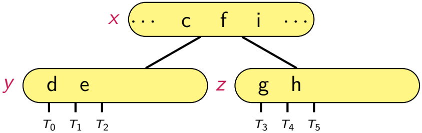
DIVIDEFILHO(x, i)

```
z = ALOCA()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
ESCREVENODISCO(y)
ESCREVENODISCO(z)
ESCREVENODISCO(x)
```



Dividindo um nó

```
DivideFilho(x, i)
z = Aloca()
y = x.c[i]
z.folha = y.folha
z.n = t − 1
para j = 1 até t − 1
    z.chave[j] = y.chave[j + t]
se não y.folha
    para j = 1 até t
        z.c[j] = y.c[j + t]
y.n = t − 1
para j = x.n + 1 decrescendo até i + 1
    x.c[j + 1] = x.c[j]
x.c[i + 1] = z
para j = x.n decrescendo até i
    x.chave[j + 1] = x.chave[j]
x.chave[i] = y.chave[t]
x.n = x.n + 1
EscreveNoDisco(y)
EscreveNoDisco(z)
EscreveNoDisco(x)
```

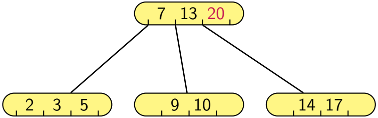


Inserindo

Vamos inserir a chave k na árvore T

- verificamos se não é necessário dividir a raiz

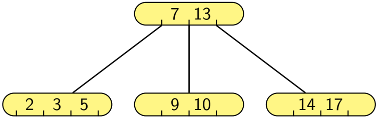
```
Insere(T, k)
r = T.raiz
se r.n == 2t − 1
    s = Aloca()
    T.raiz = s
    s.folha = Falso
    s.n = 0
    s.c[1] = r
    DivideFilho(s, 1)
    InsereNãoCheio(s, k)
senão
    InsereNãoCheio(r, k)
```



Vamos supor que a raiz **está cheia** (depois de inserções e remoções).

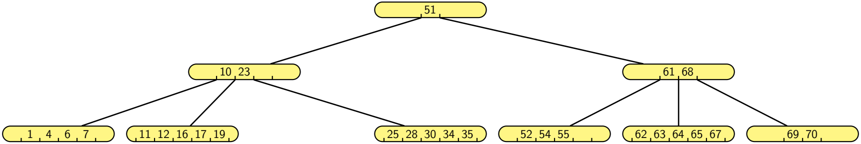
Inserindo chave k em um nó não-cheio x

```
INSERENÃOCHEIO(x, k)
i = x.n
se x.folha
    enquanto i ≥ 1 e k < x.chave[i]
        x.chave[i + 1] = x.chave[i]
        i = i − 1
    x.chave[i + 1] = k
    x.n = x.n + 1
    ESCREVENODISCO(x)
senão
    enquanto i ≥ 1 e k < x.chave[i]
        i = i − 1
    i = i + 1
    LEDODISCO(x.c[i])
    se x.c[i].n == 2t − 1
        DIVIDEFILHO(x, i)
        se k > x.chave[i]
            i = i + 1
    INSERENÃOCHEIO(x.c[i], k)
```



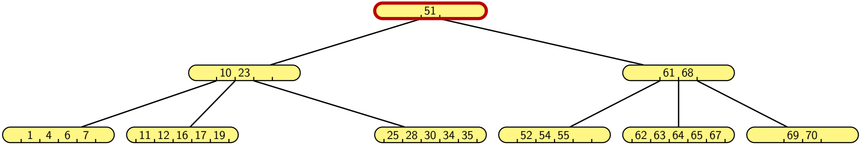
Exemplo: inserindo em nó não cheio

Inserindo 53



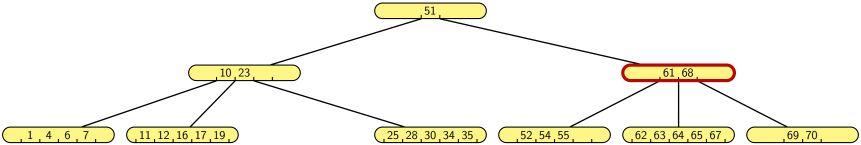
Exemplo: inserindo em nó não cheio

Inserindo 53



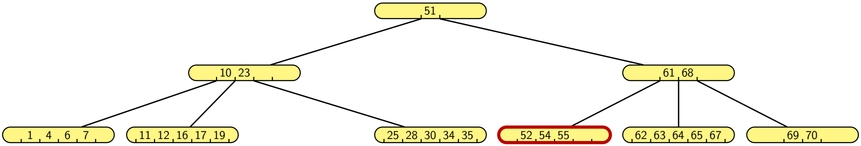
Exemplo: inserindo em nó não cheio

Inserindo 53



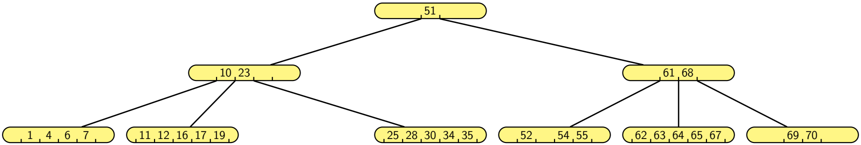
Exemplo: inserindo em nó não cheio

Inserindo 53



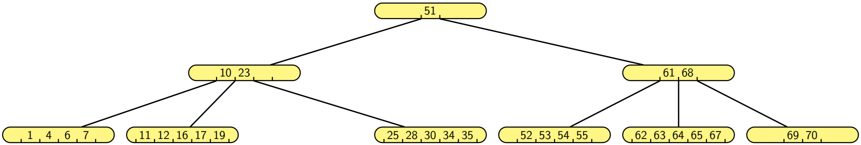
Exemplo: inserindo em nó não cheio

Inserindo 53



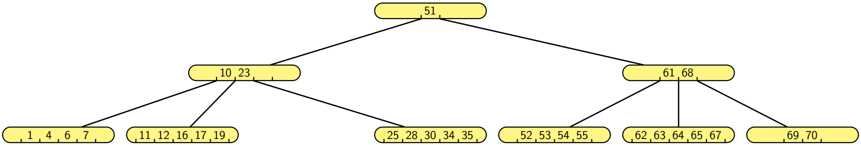
Exemplo: inserindo em nó não cheio

Inserindo 53



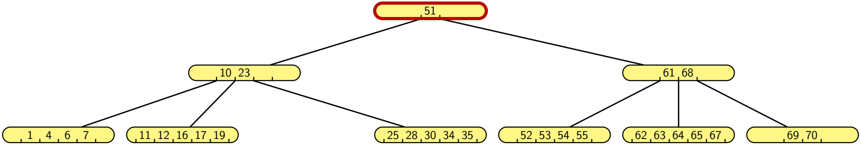
Exemplo: inserindo em nó cheio

Inserindo 18



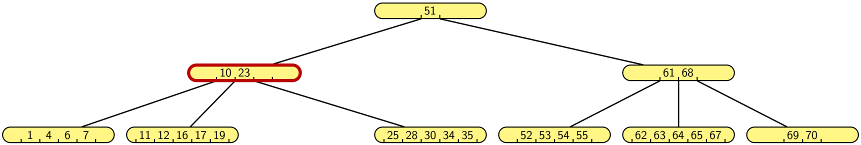
Exemplo: inserindo em nó cheio

Inserindo 18



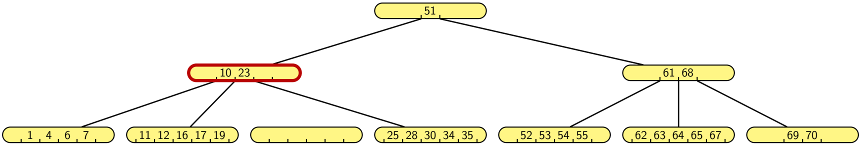
Exemplo: inserindo em nó cheio

Inserindo 18



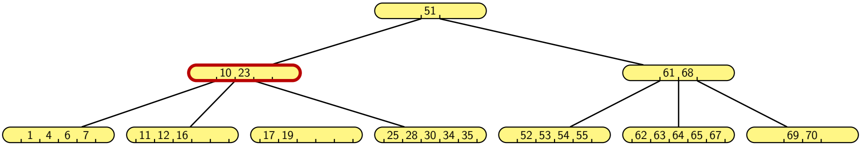
Exemplo: inserindo em nó cheio

Inserindo 18



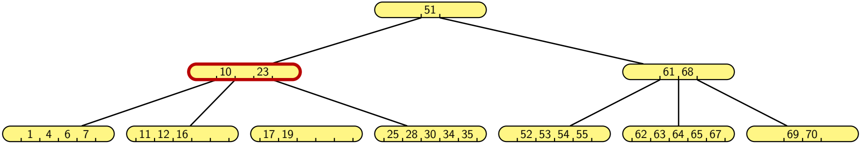
Exemplo: inserindo em nó cheio

Inserindo 18



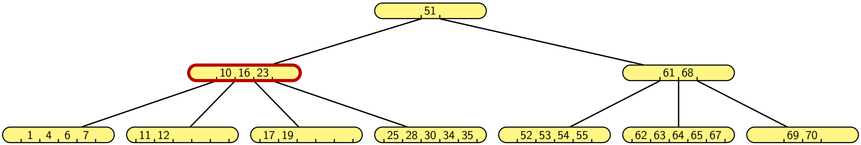
Exemplo: inserindo em nó cheio

Inserindo 18



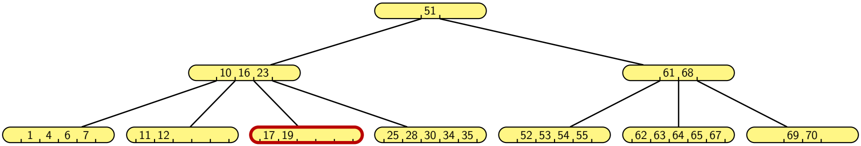
Exemplo: inserindo em nó cheio

Inserindo 18



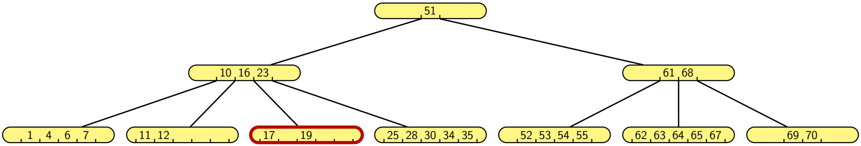
Exemplo: inserindo em nó cheio

Inserindo 18



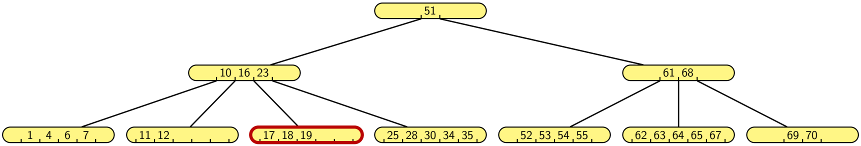
Exemplo: inserindo em nó cheio

Inserindo 18



Exemplo: inserindo em nó cheio

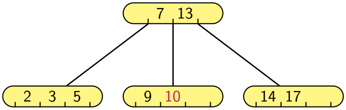
Inserindo 18



Remoção

A remoção é **mais complicada** que a inserção

- 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



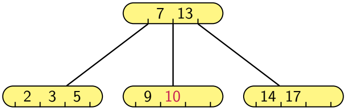
Remoção

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

- nesse caso não há problema em remover



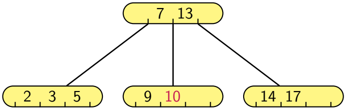
Remoção

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



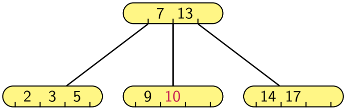
Remoção

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
  - quando os dois vizinhos tiver apenas $t - 1$ chaves
  - juntamos os nós formando um nó com $2t - 1$ chaves



Roteiro

Variantes

Árvores B^*:

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

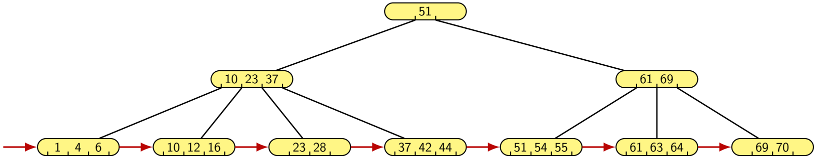
Variantes

Árvores B^*:

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

Árvores B^+:

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



Fim

Dúvidas?

Roteiro

Referências

1. Materiais adaptados dos slides do Prof. Rafael C. S. Schouery, da Universidade Estadual de Campinas.
2. Thomas H. Cormen *et al.*, "*Introduction to Algorithms - Second Edition*" (Capítulo 18).