CTC-12

Projeto e Análise de Algoritmos

Carlos Alberto Alonso Sanches

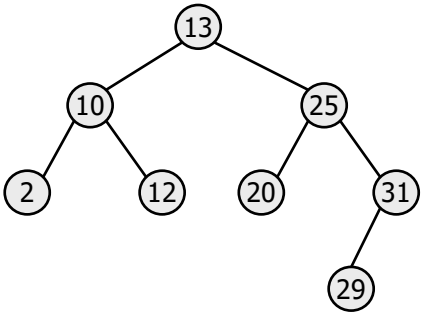
CTC-12

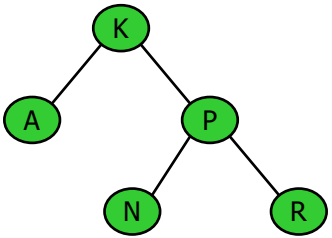
3) Árvores balanceadas

AVL, Rubro-Negras, *B-Trees*

Árvores binárias de busca

- Uma árvore binária é de busca se *em cada um de seus nós* :
  - todas as chaves armazenadas na sua sub-árvore esquerda são menores que a chave do próprio nó;
  - todas as chaves armazenadas na sua sub-árvore direita são maiores que a chave do próprio nó.
- Exemplos:





- É fácil constatar que esta estrutura de dados permite a realização de operações de busca, inserção e eliminação em tempo de pior caso proporcional à altura da árvore.

Operações em árvores binárias de busca

- Operações de inserção e de eliminação de dados a partir das chaves:
  - Inserção "ingênua":
    - São realizadas comparações a partir da raiz, descendo-se à esquerda ou à direita, até encontrar uma posição vaga.
  - Eliminação "ingênua":
    - Inicialmente, é preciso encontrar o nó a ser eliminado.
    - Se for uma folha, basta retirá-la.
    - Se tiver um único filho, este filho ocupará o seu lugar.
    - Se tiver dois filhos, será necessário trocá-lo com o descendente mais à esquerda da sua sub-árvore direita (ou com o descendente mais à direita da sua sub-árvore esquerda).
- A realização aleatória dessas operações pode gerar árvores binárias de busca com formatos muito variados, comprometendo a eficiência das futuras operações...

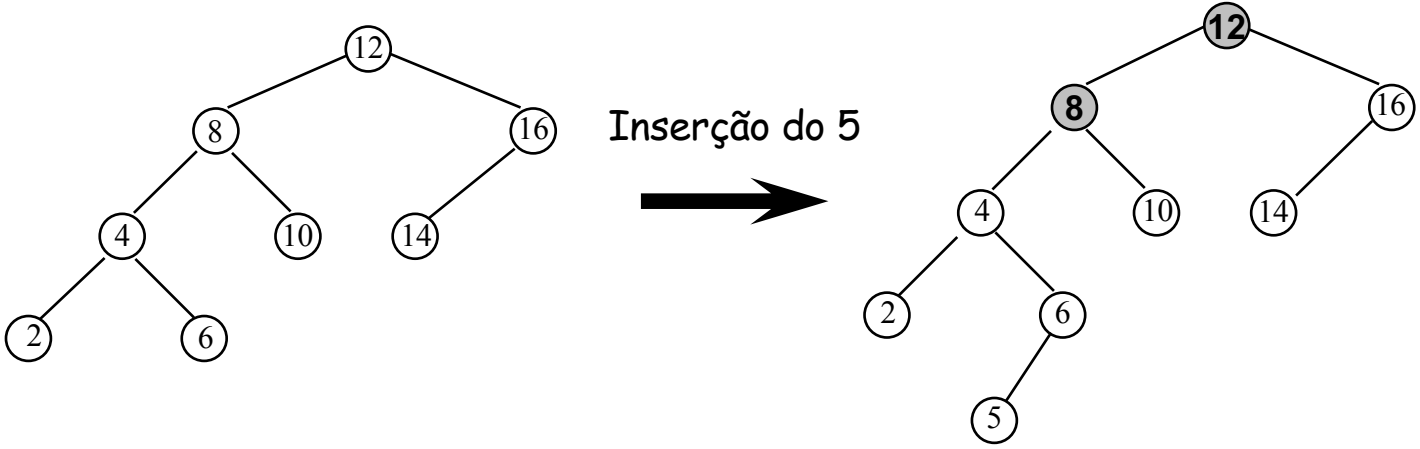
Necessidade de balanceamento

- Uma árvore binária de busca com inserções e eliminações aleatórias não garante acesso em tempo logarítmico:
  - Essas operações podem desbalanceá-la.
  - A árvore pode degenerar numa lista ligada, onde a busca passa a gastar tempo linear no pior caso.
- Balanceamento das árvores binárias de busca:
  - Evitam esses casos degenerados.
  - Garantem tempo logarítmico em todas as operações.
  - Requerem algoritmos mais elaborados para inserção e eliminação.
  - De modo geral, os nós das árvores balanceadas armazenam mais informações.
- Dois conhecidos modelos: árvores AVL e rubro-negras.

Árvores AVL

- Autores: Adelson-Velskii e Landis (1962)
- Foi o primeiro modelo de balanceamento proposto para árvores binárias de busca.
- Exigências para as sub-árvores de cada nó:
  - Diferença de alturas não pode exceder 1
  - Pode ser mantida sem onerar o tempo das operações
  - Garante altura logarítmica para a árvore
- Definição: uma árvore AVL é uma árvore binária de busca em cujos nós as alturas das sub-árvores diferem no máximo de uma unidade.

Exemplo de árvore AVL



12 e 8 desbalanceados

Após cada inserção ou eliminação, é necessário verificar o balanceamento de todos os nós da árvore

Inserção em árvores AVL

- Após uma inserção, somente podem ficar desbalanceados os nós do caminho da raiz até esse novo nó.
- Nesse caminho, é preciso encontrar o nó mais profundo (de maior nível) no qual ocorreu desbalanceamento.
  - Veremos que basta rebalancear esse nó!
- Supondo que X seja esse nó, possíveis casos a serem analisados:

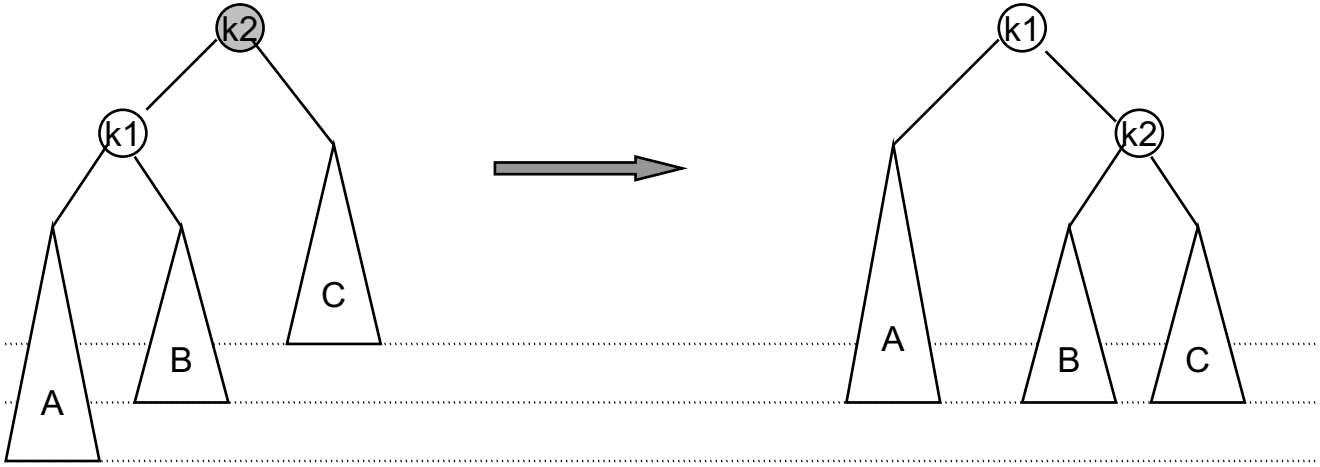
  a) árvore esquerda do filho esquerdo de X

  b) árvore direita do filho esquerdo de X

  c) árvore esquerda do filho direito de X

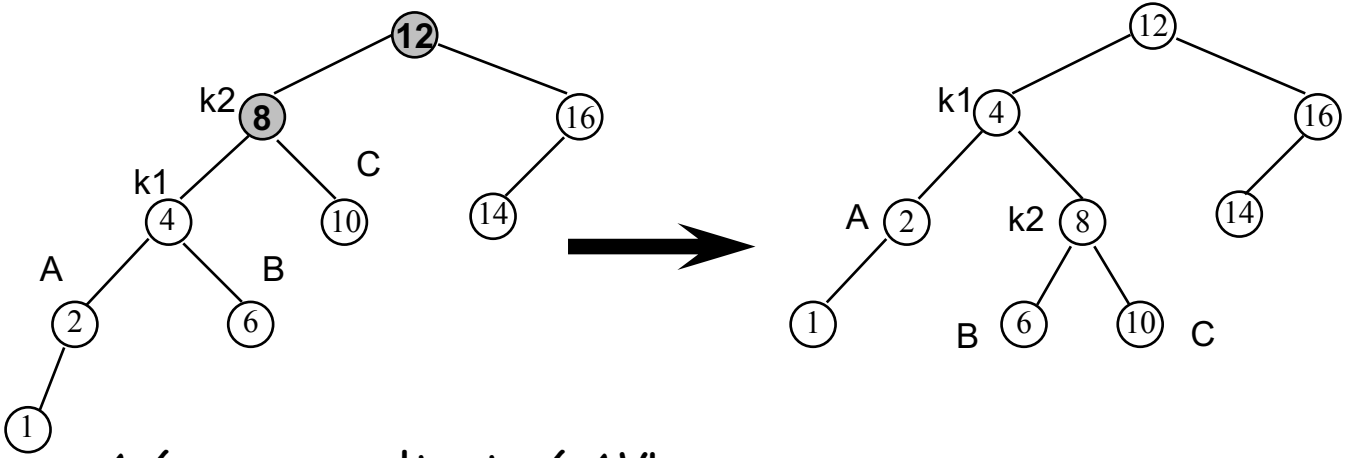
  d) árvore direita do filho direito de X
- Casos simétricos: *a* e *d* (caso 1); *b* e *c* (caso 2).

Caso 1: rotação simples



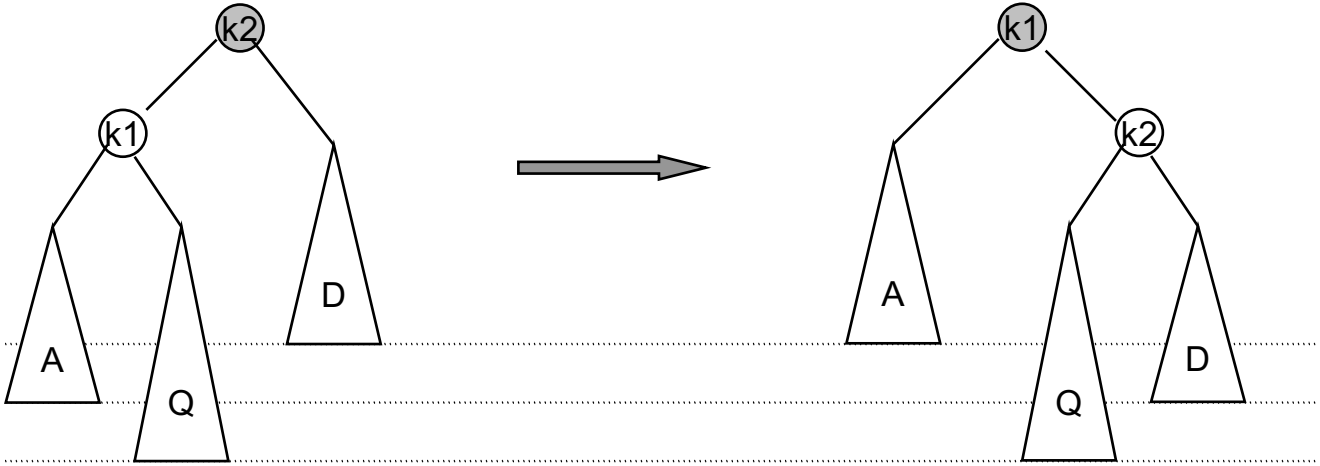
- k2 é nó mais profundo que sofreu desbalanceamento
  - Sua sub-árvore esquerda ficou 2 níveis abaixo da direita
    - B não está no mesmo nível de A (pois k2 estaria desbalanceado antes da inserção)
    - B não está no mesmo nível que C (pois k1 seria o nó mais profundo)

Exemplo



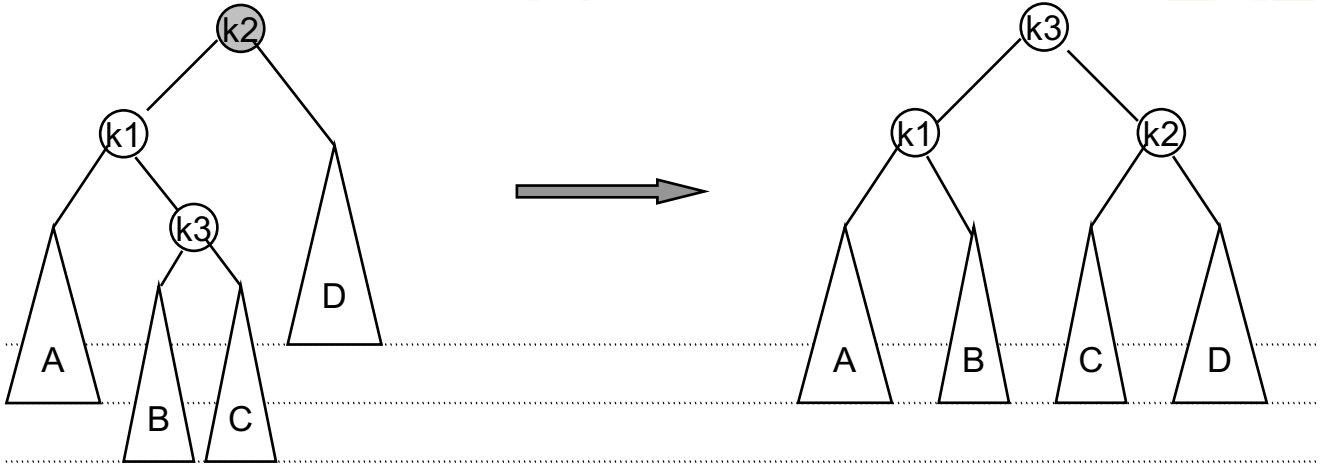
- A árvore resultante é AVL
  - k1 e k2 ficam balanceados
  - A altura da árvore resultante é igual à da árvore anterior à inserção
  - A troca de ponteiros pode ser feita em tempo constante
  - O nó k2 pode ser encontrado em tempo proporcional à altura da árvore

Caso 2



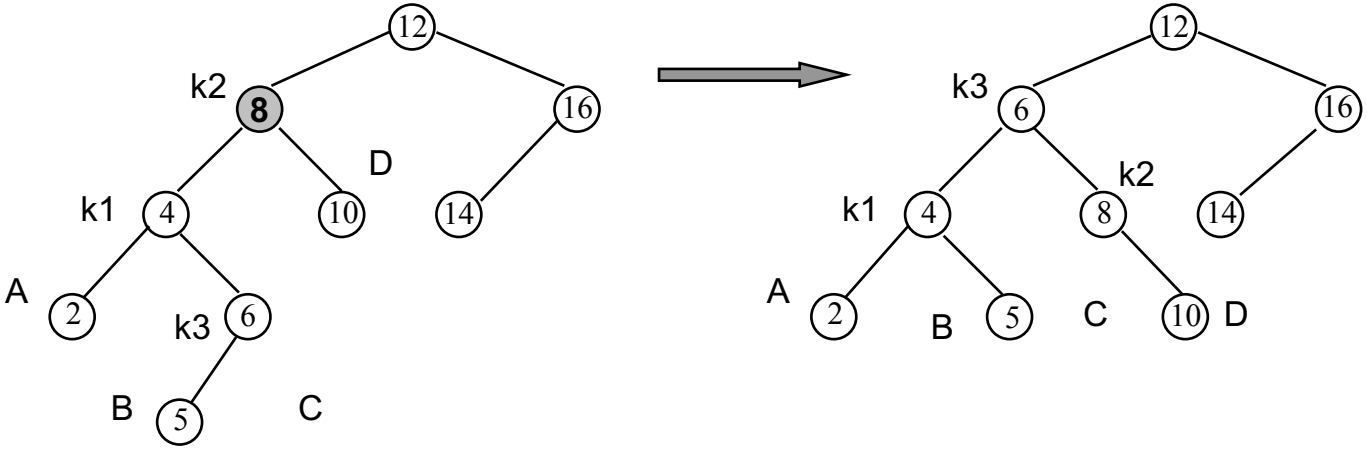
- Uma rotação simples não resolveria o desbalanceamento
  - A sub-árvore Q, que está a 2 níveis de diferença de D, passaria a estar a 2 níveis de diferença de A

Caso 2: rotação dupla



- Uma (e somente uma) das sub-árvores B ou C está 2 níveis abaixo de D
  - k3 ficaria na raiz
  - As novas posições de k1, k2 e das sub-árvores respeitam a ordenação
  - A altura da árvore resultante é igual à da árvore anterior à inserção

Exemplo



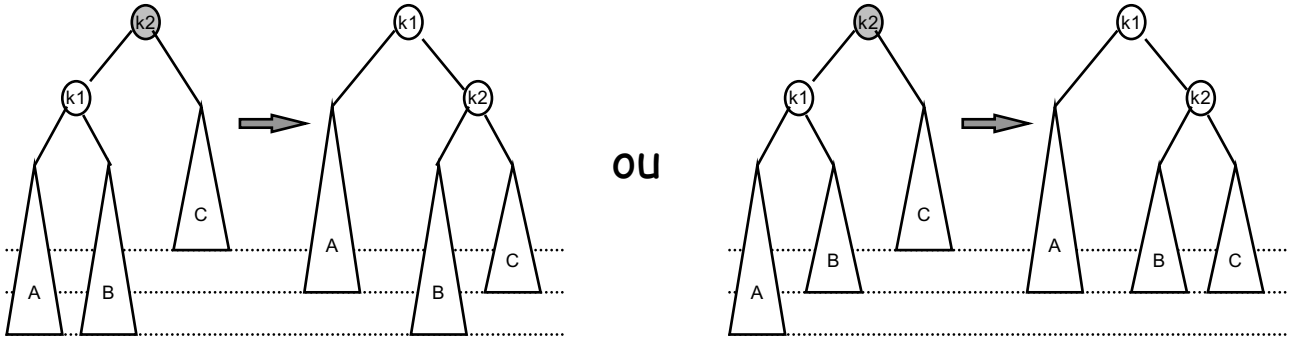
- Essa rotação dupla corresponde a 2 rotações simples
  - Entre k1 e k3
  - Entre k2 e k3
- Também pode ser feita em tempo constante

Eliminação em árvores AVL

- A eliminação de um nó numa árvore AVL é, inicialmente, análoga à que ocorre numa árvore binária de busca.
- Como foi comentado, essa técnica pode provocar desbalanceamentos na árvore.
- O rebalanceamento começará no nó mais profundo que, após a eliminação, perdeu a propriedade AVL.
- Do modo semelhante às inserções, será preciso verificar três pares de casos.

Casos de rotação simples

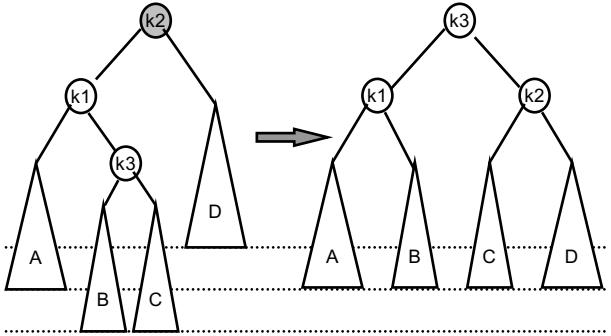
- Nos dois esquemas abaixo, como a eliminação ocorreu na sub-árvore C, bastará realizar uma rotação simples:



- No segundo esquema, a sub-árvore resultante diminuiu de altura (uma unidade).
  - Por isso, também será preciso realizar um eventual rebalanceamento no pai de k1. Isso pode continuar até a raiz...
- Há também os correspondentes casos simétricos a esses esquemas (C é inicialmente a sub-árvore esquerda de k2).

Casos de rotação dupla

- No esquema abaixo, B ou C podem ter altura menor (uma unidade). Como a eliminação ocorreu na sub-árvore D, será preciso realizar uma rotação dupla:



- A sub-árvore resultante diminuiu de altura (uma unidade).
  - Por isso, também será preciso realizar um eventual rebalanceamento no pai de k3. Isso pode continuar até a raiz...
- Há também o caso simétrico, onde D é inicialmente a sub-árvore esquerda de k2 e k3 é filho esquerdo de k1.

Altura de árvores AVL

- Seja n(h) o número mínimo de nós de uma árvore AVL com altura h.
- Sabemos que n(0)=1 e n(1)=2.
- Para h>1, essa árvore AVL mínima será formada pela raiz, por uma sub-árvore de altura h-1 e por outra sub-árvore de altura h-2.
- Portanto, $n(h) = 1 + n(h-1) + n(h-2)$, para h>1.
- Como $n(h-1) > n(h-2)$, sabemos que $n(h) > 2.n(h-2)$.
- Repetindo:
  - $n(h) > 2.n(h-2) > 2.(2.n(h-2-2)) = 4.n(h-4)$
  - $n(h) > 4.n(h-4) > 4.(2.n(h-4-2)) = 8.n(h-6)$
  - Generalizando: $n(h) > 2^i.n(h-2i)$, para i>0.

Altura de árvores AVL

- $n(h) > 2^i.n(h-2i)$, para $i>0$.
- Consideremos o caso $h-2i = 1$, ou seja, $i = (h-1)/2$:
  - $n(h) > 2^{(h-1)/2}.n(1)$
  - $n(h) > 2.2^{(h-1)/2}$
  - $n(h) > 2^{(h+1)/2}$
  - $\lg n(h) > (h+1)/2$
  - $h < 2.\lg n(h) - 1$
- Lembrando: $n(h)$ é o número mínimo de nós de uma árvore AVL com altura h.
- Portanto, $h = O(\log n)$: a altura de uma árvore AVL é de ordem logarítmica em relação ao seu número de nós.
- Desse modo, os algoritmos de inserção, de eliminação e de busca na árvore AVL são logarítmicos!

Altura de árvores AVL

- Por outro lado, é fácil verificar que n(h) = F(h+3)-1, onde F(h) é o h-ésimo número de Fibonacci.
- Mais precisamente, sabe-se que h < 1,44.lg (n+2), onde h é a altura e n é o número de nós de uma árvore AVL, ou seja, o pior caso tem um fator multiplicativo pequeno.
- Em 1998, Knuth mostrou que, para n grande, a altura média de uma árvore AVL é lg n + 0,25.

Implementação

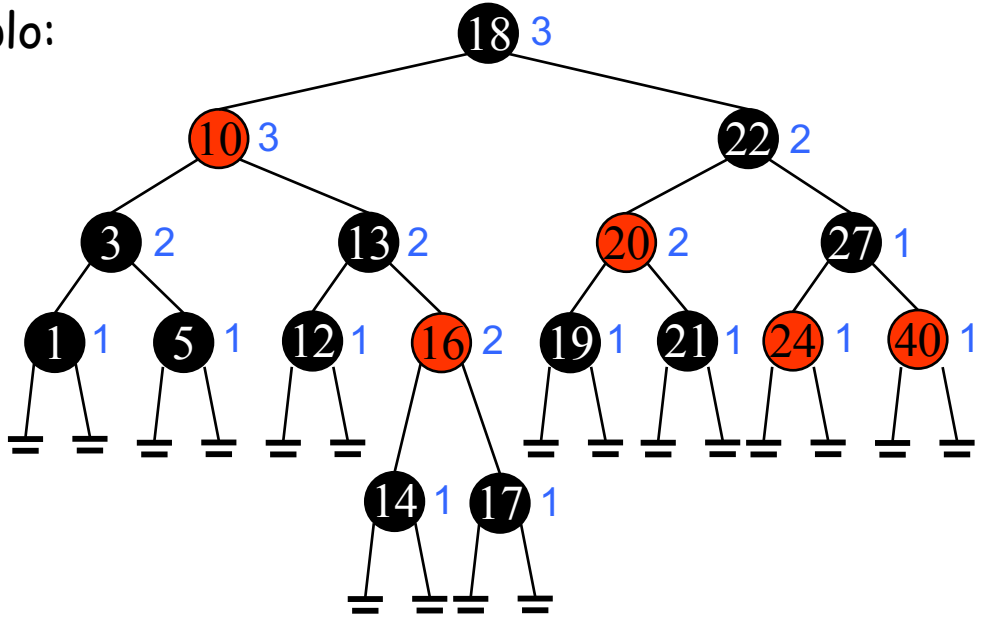
- Vimos que, nas árvores AVL, inserções e eliminações gastam, no pior caso, tempo proporcional à altura da árvore, que por sua vez é limitada pelo logaritmo do número de nós.
- Na sua implementação, o que seria preciso acrescentar à estrutura de dados?
- Em cada nó, basta acrescentar uma *flag* de três valores (-1, 0 ou 1) que indica a diferença entre as alturas das suas sub-árvores.

Árvores rubro-negras

- As árvores rubro-negras (Guibas e Sedgewick, 1978) são árvores binárias de busca balanceadas segundo um critério diferente do usado nas árvores AVL.
- Todos os nós (inclusive os nulos!) têm cor vermelha ou preta:
  1) A raiz e os nós nulos são pretos.
  2) Se um nó é vermelho, então seus filhos são pretos.
  3) Todos os caminhos de um nó até qualquer nó nulo percorrem um número idêntico de nós pretos (é a *altura negra* desse nó).
- Em cada sub-árvore rubro-negra, considere que a folha mais próxima e a folha mais distante de sua raiz tenham distâncias p e d, respectivamente, onde $p \leq d$.
- As condições acima asseguram que $d \leq 2p$.

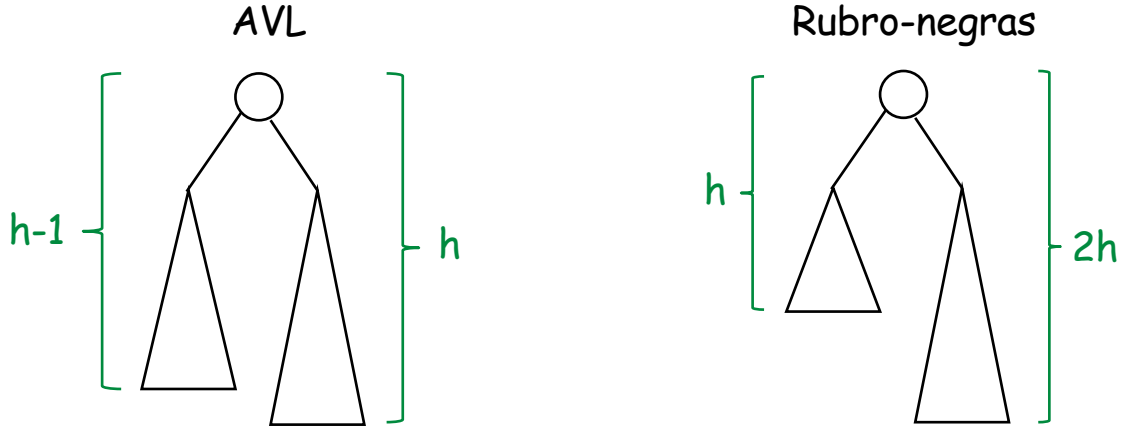
Árvores rubro-negras

- Definição: a *altura negra* de um nó x, designada por an(x), é o número de nós pretos abaixo dele em qualquer caminho até um nó nulo.
- Exemplo:



AVL x rubro-negras

- Ambas estruturas "toleram" um certo desbalanceamento em cada nó da árvore:



- Em ambos os casos, essa tolerância é de simples manutenção, e garante relação logarítmica entre altura e número total de nós.

Árvores rubro-negras

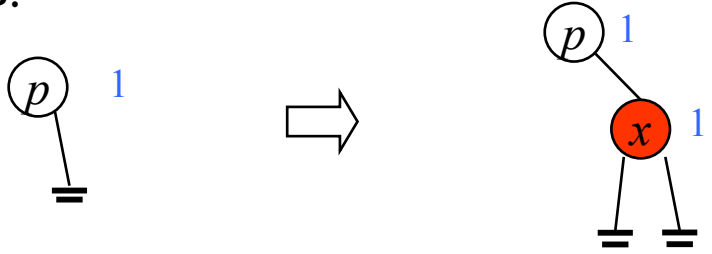
- Lema 1: Numa árvore rubro-negra, a sub-árvore enraizada em um nó x tem no mínimo $2^{an(x)} - 1$ nós não nulos.
- Prova por indução:
  - Para a árvore com apenas uma folha (por ser a raiz, será preta): $2^1 - 1 = 1$ nó não nulo.
  - Caso genérico: nó x não-terminal
    - Um filho de x, se for vermelho, terá altura negra an(x); se for preto, terá altura negra an(x) – 1.
    - Portanto, pela hipótese de indução, cada sub-árvore de x terá, no pior caso (isto é, quando os dois filhos de x forem pretos), $2^{an(x)-1} - 1$ nós não nulos.
    - Logo, a sub-árvore enraizada em x terá no mínimo $2(2^{an(x)-1} - 1) + 1 = 2^{an(x)} - 1$ nós não nulos.

Árvores rubro-negras

- Lema 2: Uma árvore rubro-negra com n nós não nulos tem no máximo altura 2.lg (n + 1).
  - Prova: Se uma árvore rubro-negra tem altura h, a altura negra da raiz será no mínimo h/2 (pois todo nó vermelho tem filho preto). Portanto, pelo lema anterior, a árvore terá $n \geq 2^{h/2} - 1$ nós não nulos, de onde segue o Lema 2.
- Isso permite a realização das operações de busca, inserção e eliminação numa árvore rubro-negra em tempo O(log n), desde que suas propriedades sejam mantidas.

Inserção em árvores rubro-negras

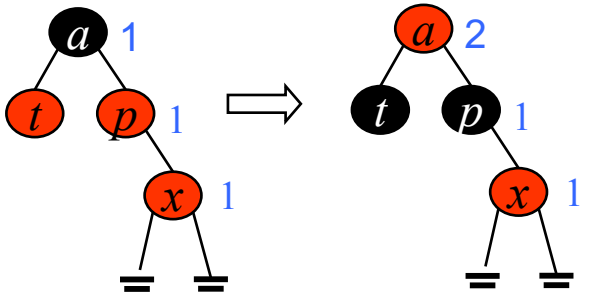
- Ao contrário das árvores AVL, agora é preciso ajustar outro critério: as cores dos nós.
- Se um nó x for inserido numa árvore vazia, ele será a raiz. Portanto, deverá ser preto.
- Caso contrário, ao inserir um nó x numa posição vazia da árvore (isto é, no lugar de um nó nulo), ele será pintado de vermelho para tentar não alterar a altura negra dos seus antecessores.



- Mas é preciso verificar o que acontecerá com p, pai de x...

Inserção em árvores rubro-negras

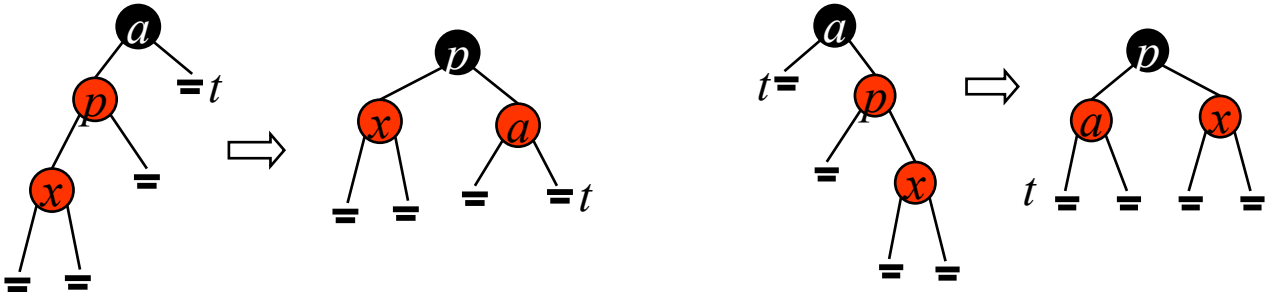
- [Caso 1] Se o nó p for preto, nada mais precisa ser feito.
- [Caso 2] Por outro lado, se p for vermelho (logo, não será a raiz), o nó a, pai de p e avô de x, será preto. Se houver um nó t vermelho, irmão de p e tio de x, então serão modificadas as cores de a, t e p.



- Se o pai do nó a for vermelho, o rebalanceamento continuará, seguindo o mesmo algoritmo.
- Se esse processo chegar até a raiz, basta trocar sua cor (de vermelho para preto).

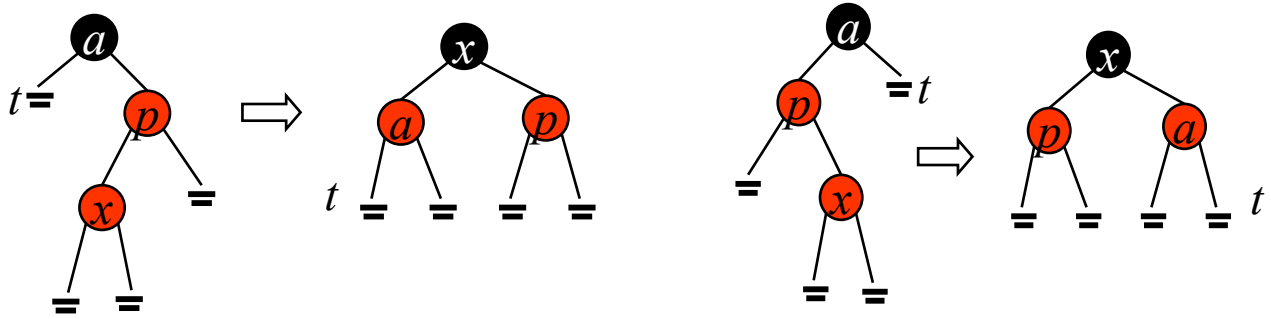
Inserção em árvores rubro-negras

- [Caso 3] Finalmente, se não houver um nó t vermelho, será preciso fazer rotações envolvendo a, t, p e x.
  Há 4 subcasos (que são simétricos, 2 a 2).
  - [Caso 3a] Rotações simples:



Inserção em árvores rubro-negras

- [Caso 3b] Rotações duplas:



- Não é preciso considerar o caso de t como folha preta. **Por quê?**

Exercício

- Pesquisar os algoritmos de eliminação em árvores rubro-negras e comprovar que também gastam tempo O(log n).
- Importante: repare que, na implementação de uma árvore rubro-negra, é suficiente que cada nó armazene a sua cor (ou seja, basta uma simples *flag booleana*).

Quando convém utilizar essa estrutura?

- As árvores binárias de busca balanceadas são muito adequadas para pesquisa em memória primária, pois satisfazem *simultaneamente* alguns requisitos conflitantes:
  - Acesso direto eficiente (tempo logarítmico)
  - Acesso sequencial eficiente (tempo linear)
  - Inserção e eliminação eficientes (tempo logarítmico)
  - Boa taxa de utilização da memória
- Para pesquisa em memória secundária, também há uma boa diversidade de modelos. O mais conhecido é a árvore B (Bayer e McCreight, 1972) com suas variantes.

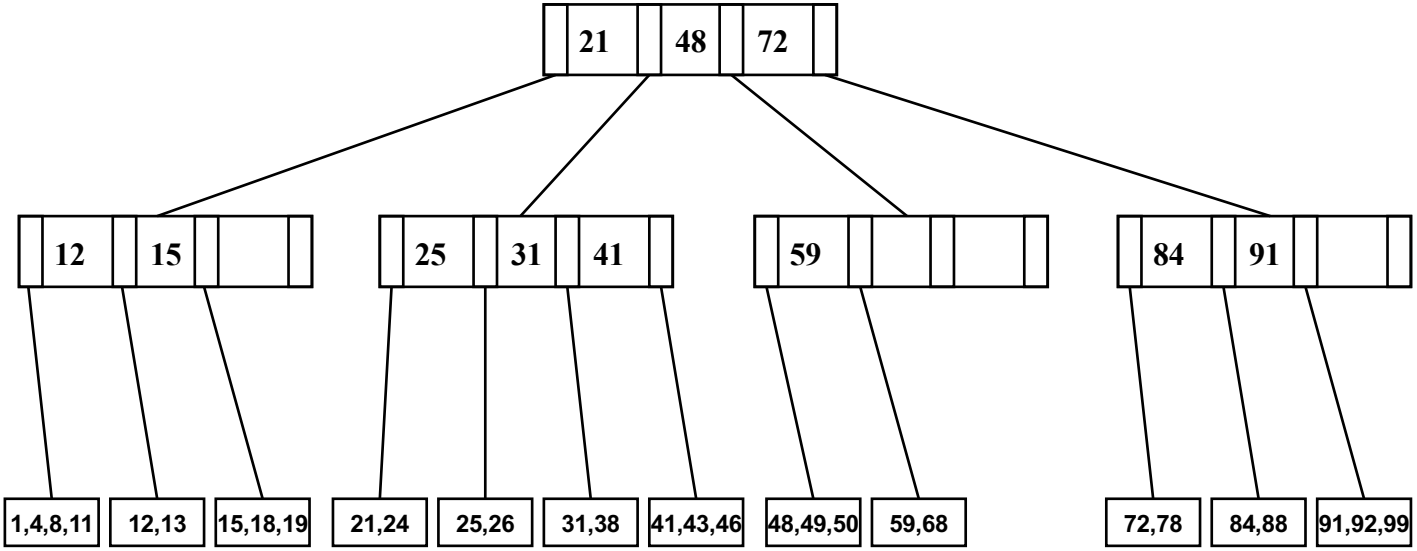
Árvores B (Bayer e McCreight, 1972)

- São árvores de pesquisa balanceadas projetadas para dispositivos de armazenamento secundário (discos, fitas magnéticas, etc.).
- O *fator de ramificação* (número máximo de filhos) é escolhido em função do tamanho do bloco de leitura/escrita dessa memória, que costuma ser chamado de *página*.
- Geralmente, os nós contêm informações armazenadas numa mesma página. Por isso, os conceitos de *nó* e *página* acabam se identificando.
- As *B-Trees* são apropriadas, por exemplo, para sistemas de bancos de dados.

Árvores B*

- Veremos apenas um caso particular: a árvore B*
- Em uma árvore B* de ordem M:
  - Todas as folhas estão no mesmo nível
  - A raiz, se não for folha, tem de 2 a M filhos
  - Exceto a raiz, todos os nós internos têm de $\lceil M/2 \rceil$ a M filhos
  - Os nós internos guardam de $\lceil M/2 \rceil - 1$ a M-1 chaves (para identificar os valores armazenados nos filhos)
  - Os dados associados às chaves estão guardados nas folhas (é o que a diferencia das demais árvores B)
  - Cada folha tem de $\lceil M/2 \rceil$ a M dados armazenados
- Ela também é chamada de árvore B* $\lceil M/2 \rceil$-M

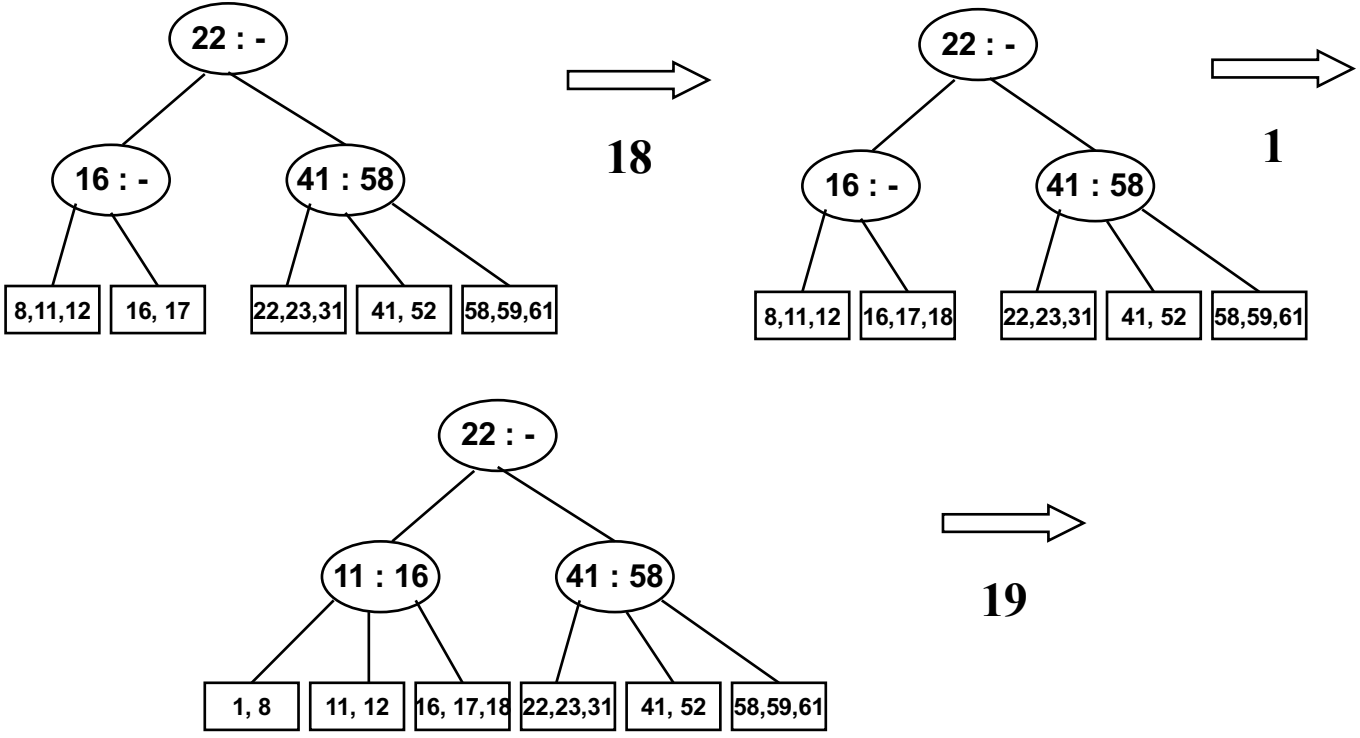
Exemplo: árvore B* de ordem 4 (ou 2-4)



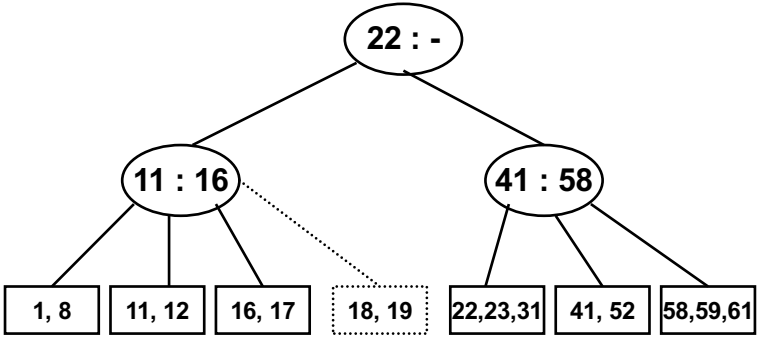
M = 4

- Nós internos: 2 a 4 filhos (1 a 3 chaves)
- Folhas: 2 a 4 dados

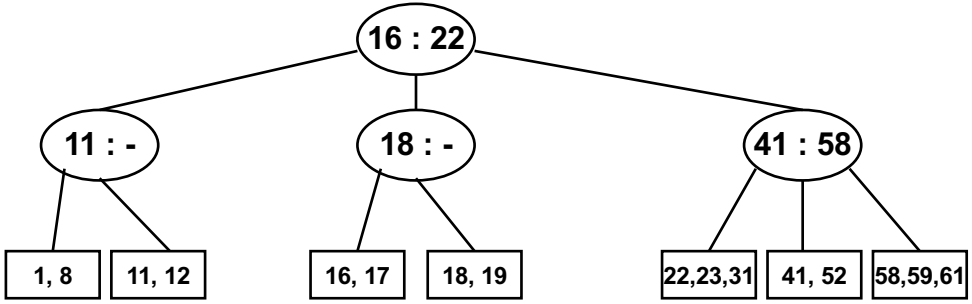
Inserções em árvore B* 2-3



Inserções em árvore B* 2-3

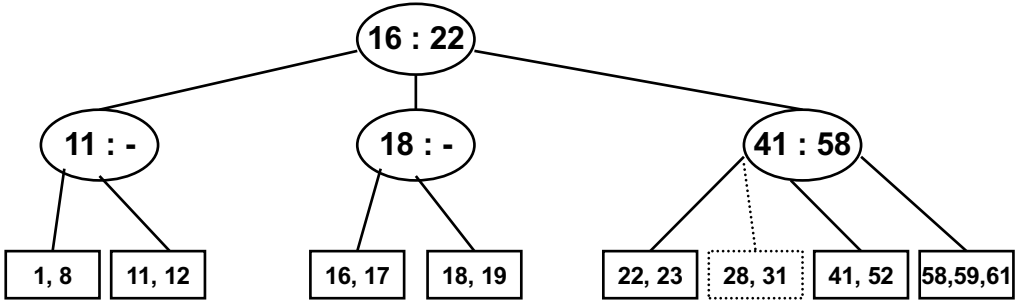


Ultrapassaria o número máximo de filhos...

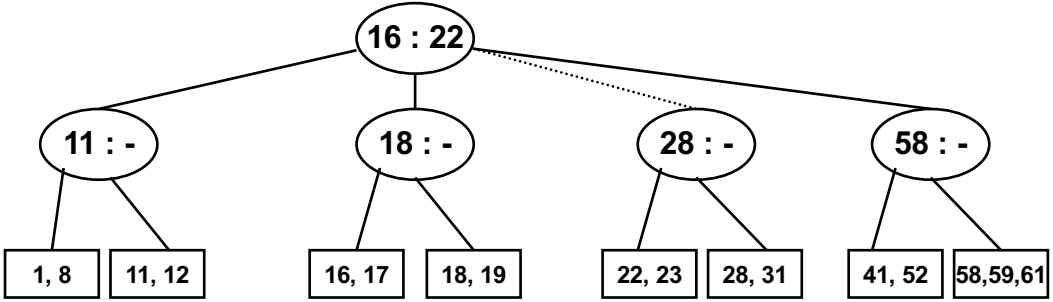


28

Inserções em árvore B* 2-3

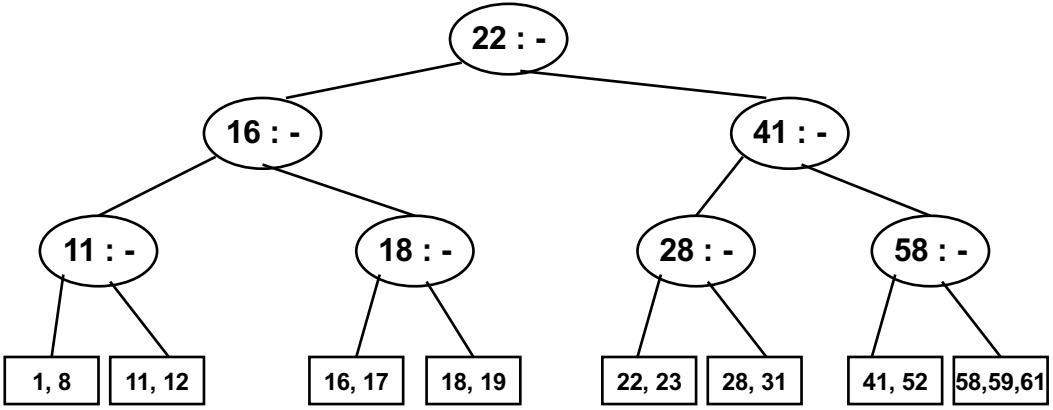


Idem...



Idem...

Inserções em árvore B* 2-3



No pior caso, as inserções se propagam até a raiz, e a altura da árvore cresce 1 unidade

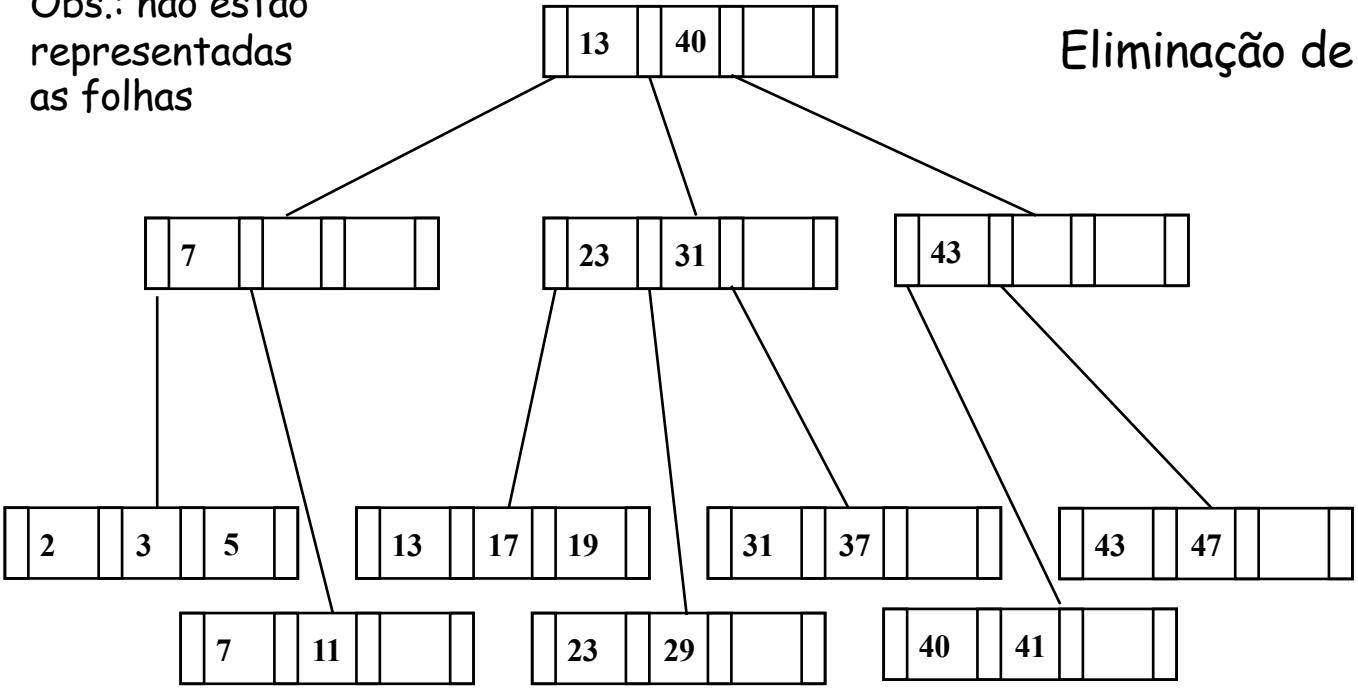
Eliminações em árvores B*

- Busca da folha com o dado a ser eliminado:
  - Se a folha ficar com um suficiente número de dados: fim.
  - Se a folha ficar com um número de dados abaixo do mínimo:
    - Se a folha irmã tiver um número de dados acima do mínimo: pegar um dado dessa folha.
    - Se a folha irmã não tiver um número de dados acima do mínimo: fundir ambas as folhas.
  - Quando ocorre fusão de nós, a eliminação prossegue nos níveis superiores da árvore.
  - Se a fusão de nós deixar a raiz com apenas um filho, então ela se juntará ao filho e a altura da árvore diminuirá um nível.

Eliminações em árvore B* 2-4

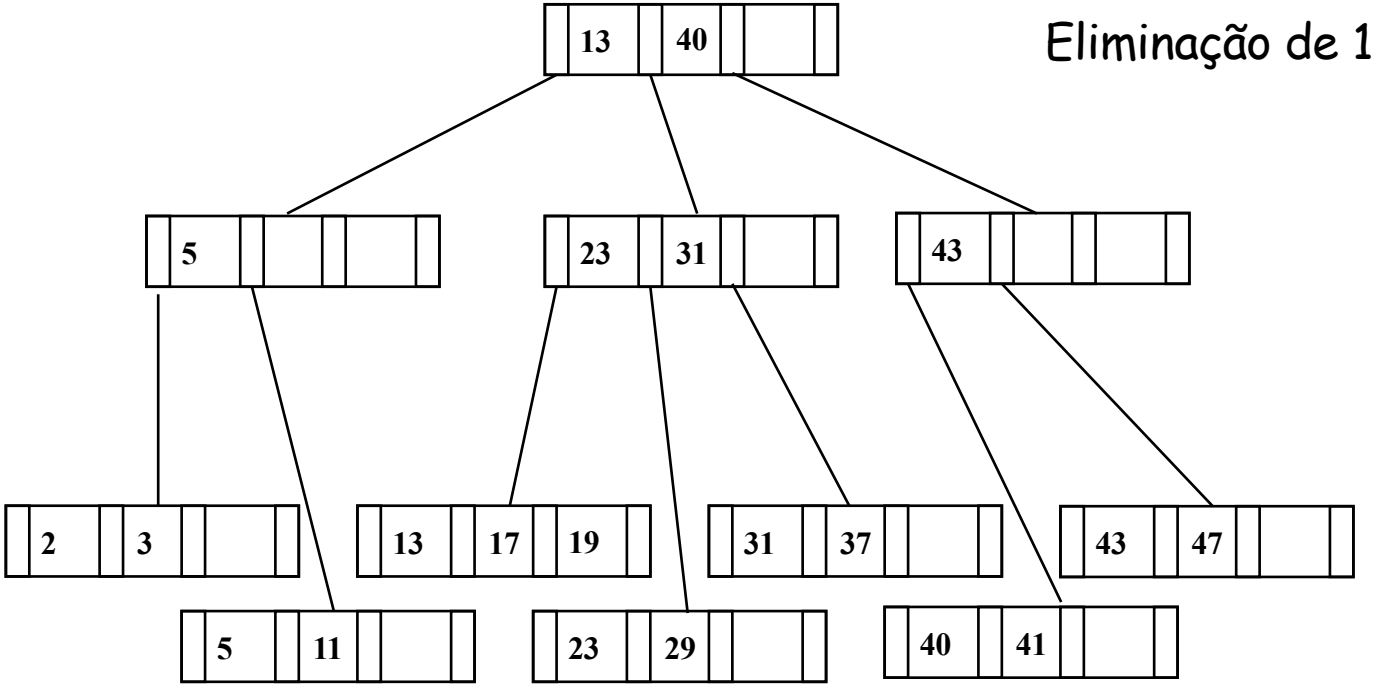
Obs.: não estão representadas as folhas

Eliminação de 7

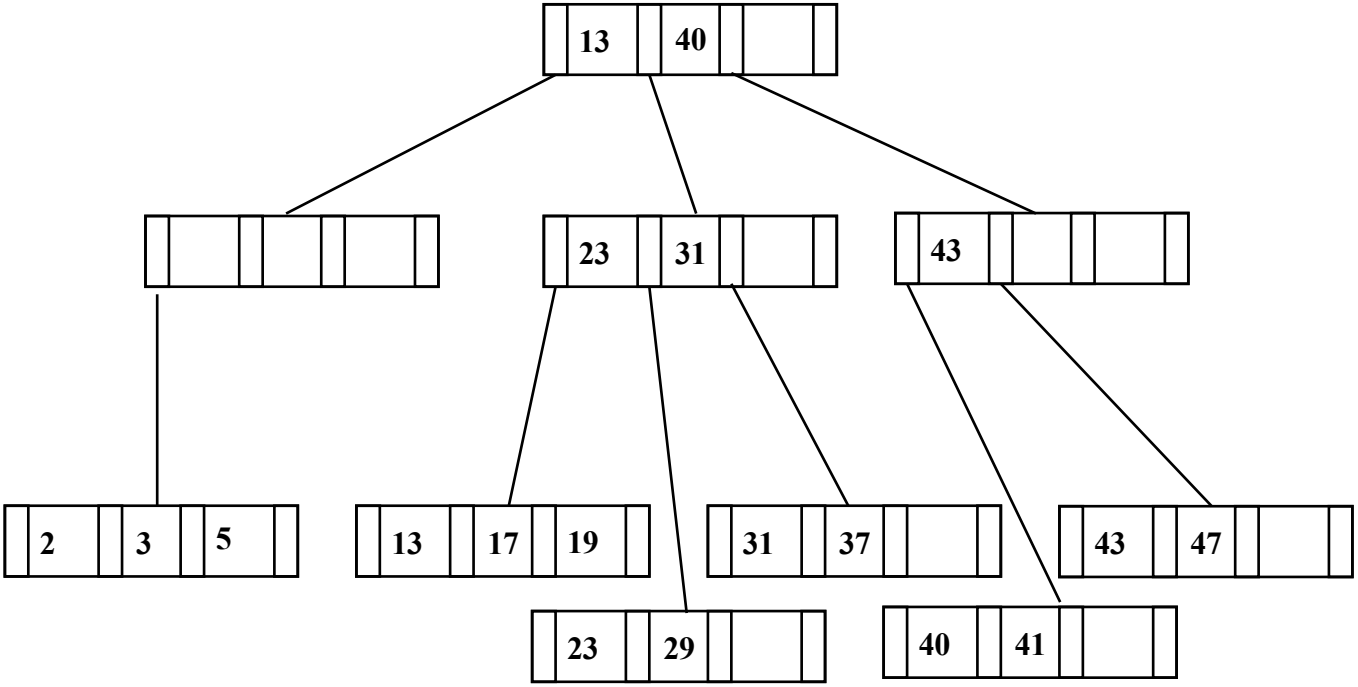


Eliminações em árvore B* 2-4

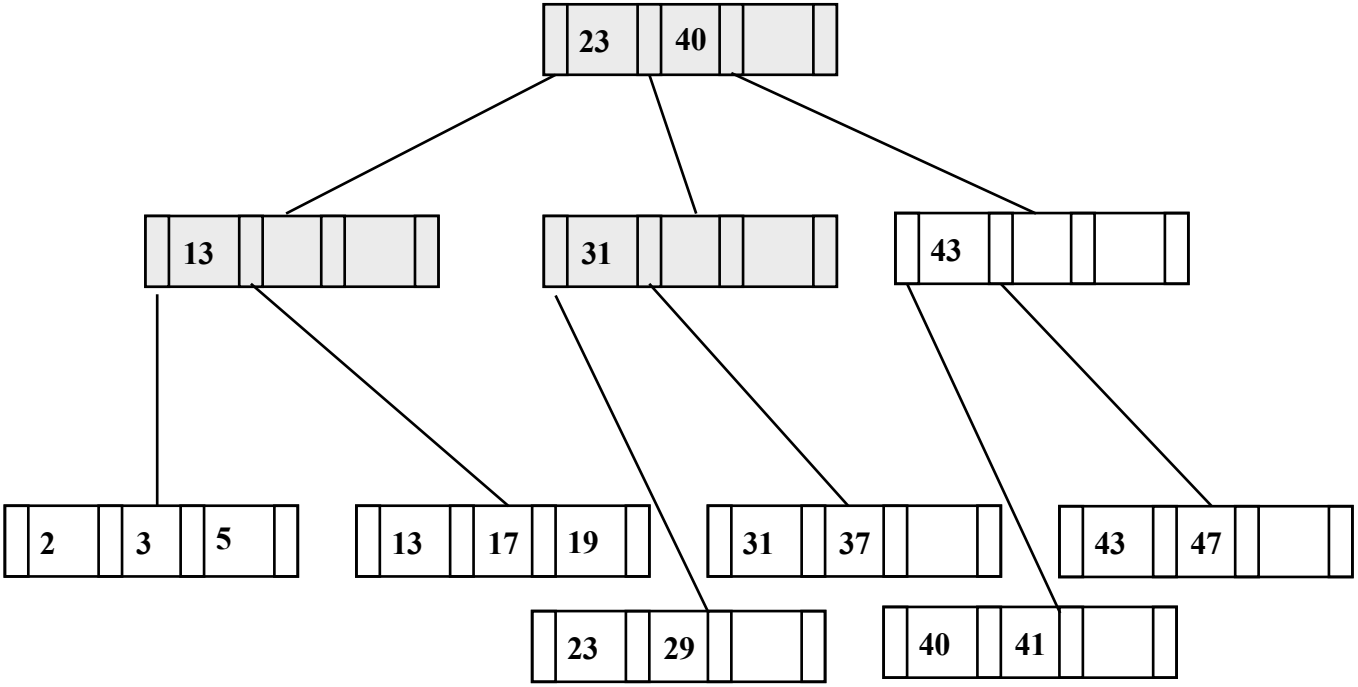
Eliminação de 11



Eliminações em árvore B* 2-4



Eliminações em árvore B* 2-4



Características de uma árvore B*

- Considere uma árvore B* de ordem M.
- Sendo h a altura dessa árvore, o número n de folhas será:

$$\lceil M/2 \rceil^h \leq n \leq M^h$$

  - Ideia da demonstração: as árvores B* mínima e máxima corresponderão, respectivamente, a árvores completas com $\lceil M/2 \rceil$ e M filhos por nó.
- Portanto, as operações de inserção, eliminação, busca e extração de mínimo podem ser realizadas com O(log n) acessos a páginas da memória secundária.