Árvores 2–3–4

ou também árvores 2–4

Evandro Eduardo Seron Ruiz
evandro@usp.br

Universidade de São Paulo

Recordando...

Estruturas de dados formatadas para busca

- Árvores binárias de busca
- Árvores AVL como árvores balanceadas (otimização da busca)
- Árvores M-múltiplas (otimizam a busca minimizando h)

Veremos nesta apresentação um tipo especial de árvore M-múltipla

Conteúdo

Árvores 2–3–4

- É um tipo especial de árvore M-múltipla de ordem 4
- Modelo sugerido por Rudolf Bayer em 1972 que a chamou de *árvore B binária simétrica*
- Um tipo especial de árvore B, uma forma reduzida
- Nestas árvores, 1, 2 ou, no máximo 3 elementos, podem ser armazenados num único nó da árvore

Árvores 2–3–4: detalhes técnicos

- Nós internos têm ao menos dois nós fihos
- Armazena elementos da forma (k, x), para:
  - k chave de um registro, e
  - x o registro, o elemento associado à chave
- Contém $d - 1$ elementos, para d número de nós filhos
- Nós externos, ou nós folhas, não armazenam elementos

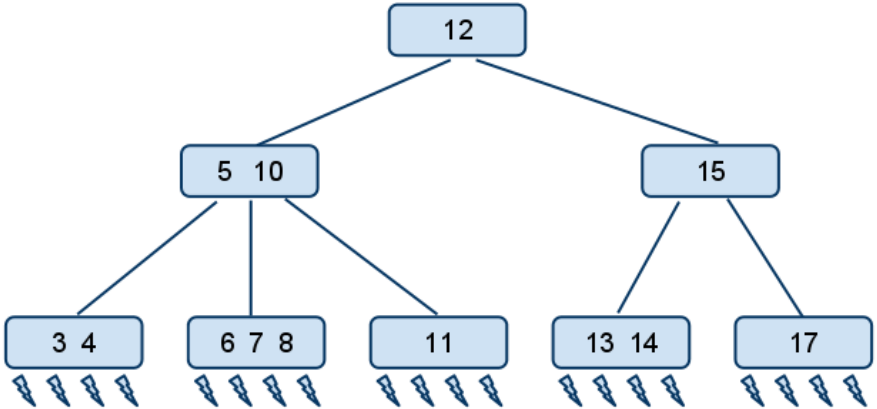
Mais detalhes...

- Cada nó tem, no máximo, 4 filhos
- Todos os nós externos estão na mesma altura, ou seja, têm o mesmo h
- Todos os nós externos apontam para o nulo
- Os elementos dentro de um nó estão ordenados crescentemente

Mais detalhes. . .

- Cada nó tem, no máximo, 4 filhos
- Todos os nós externos estão na mesma altura, ou seja, têm o mesmo h
- Todos os nós externos apontam para o nulo
- Os elementos dentro de um nó estão ordenados crescentemente
- Portanto, é um TAD balanceado

Visão gráfica de árvore 2–3–4



Comentários

- Assumimos, mais uma vez, que os nós externos são vazios ou nulos
- Manter o tamanho em 2–3–4 torna o TAD simples relativamente a uma M-múltipla
- Conteúdo de cada nó pode ser armazenado como um dicionário (vetor ordenado) para agilizar a busca

Altura

Questão

Qual seria a altura h de uma árvore 2–3–4 necessária para armazenar n nós

Altura

Questão

Qual seria a altura h de uma árvore 2–3–4 necessária para armazenar n nós

- Dado M o número de filhos de um nó
  1. Considerar a árvore com o mínimo de filhos, ou seja, $M = 2$; e, considerar também
  2. Considerar $M = 4$, ou seja, a árvore cheia.

Vejamos os dois casos...

Numericamente. . . Quantidade mínima de itens

Considerando que o nó raíz tem, no máximo, 2 nós filhos, e portanto é uma árvore binária. Portanto não é uma árvore 2–3–4.

altura(h)	número de nós	n, max. itens acumulados	$\log_2(n)$ $\sim$
1	1	1	0
2	2	3	2
3	4	7	3
4	8	15	4
5	16	31	5

Lembrem-se, os nós externos armazenam valores.

Numericamente. . . Quantidade máxima de itens

Considerando que o nó raíz tem, no máximo, 4 nós externos, e portanto tem limite de armazenamento de 3 chaves.

altura(h)	nós externos em (h)	nós internos	n, max. itens acumulados	$\log_4(n)$ $\sim$
1	4	1	3	0
2	16	4	15	2
3	64	16	60	3
4	256	64	240	4
5	1024	256	960	5

Lembrem-se, os nós externos não armazenam valores.

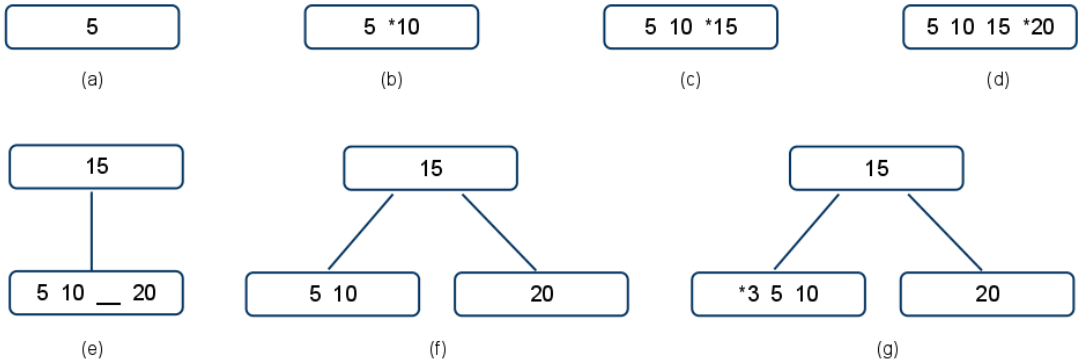
Inserção de um nó

Considerações

- Preserva a mesma altura h para todos nós externos
- Pode violar "instantaneamente" o tamanho de um nó
- Se for um nó de dimensão 3 pode, momentaneamente, ser um nó de dimensão 4. Ocorre um overflow momentâneo
- *Overflow deve ser corrigido* com operação split, ou seja, divisão de nós

Vejamos, graficamente, o mecanismo de inserção antes de formalizá-lo.
Os elementos com "*" são os elementos recém inseridos.

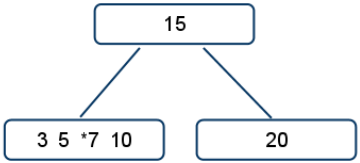
Inserção de um nó em árvore 2–3–4



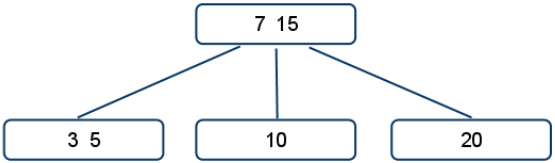
Inserção: detalhes

- Reparem o *overflow* momentâneo com a inserção do elemento **20**
- Cria-se um nó pai por conta deste *overflow*
- O nó pai recebe o **terceiro** elemento (os dados estão ordenados dentro do nó)
- Segue-se com a operação de *split* do nó

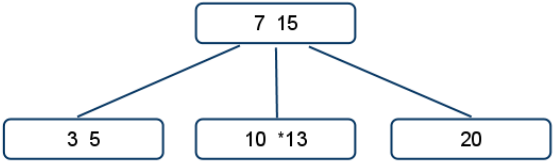
Inserção de um nó em árvore 2–3–4, continuação



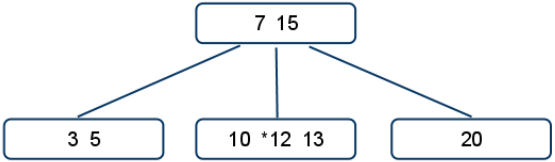
(h)



(i)



(j)



(k)

Inserção: interessante

- Estas árvores sempre crescem “para cima”
- Elas se mantém sempre balanceadas
- Pergunta: “Quem é o responsável por manter este balanceamento?”

Inserção: interessante

- Estas árvores sempre crescem “para cima”
- Elas se mantém sempre balanceadas
- Pergunta: “Quem é o responsável por manter este balanceamento?”
- R: A operação *split*

Inserção passo-a-passo

- Inserir um elemento (k, x) numa árvore T.
- Primeiro passo: busca pelo elemento
- Assumindo que não temos elementos repetidos, a busca termina num nó externo (folha) z, vazio
- Seja v o nó pai de z
- Inserimos um novo item em v (pai) e adicionamos um novo filho w (nó externo) de v a esquerda de z

Inserção: *overflow*

- A inserção pode causar *overflow* em v (pai)
- Sejam $v_1, v_2, \ldots, v_5$ filhos de v, e
- $k_1, \ldots, k_4$ as chaves armazenadas em v. Só 3 são permitidas. k_4 é o *overflow*

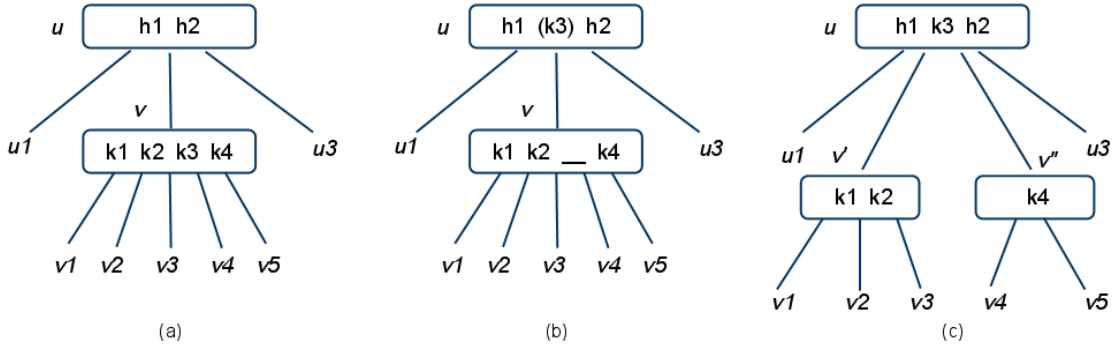
Inserção: *overflow*

- A inserção pode causar *overflow* em v (pai)
- Sejam $v_1, v_2, \ldots, v_5$ filhos de v, e
- $k_1, \ldots, k_4$ as chaves armazenadas em v. Só 3 são permitidas. k_4 é o *overflow*
- A operação de split de v faz:
  - Transforma v em dois nós: v' e v''
  - v' com chaves k_1, k_2 e filhos v_1, v_2, v_3; e
  - v'' com chave k_4 e filhos v_4, v_5

Inserção: *overflow*

- A inserção pode causar *overflow* em v (pai)
- Sejam $v_1, v_2, \ldots, v_5$ filhos de v, e
- $k_1, \ldots, k_4$ as chaves armazenadas em v. Só 3 são permitidas. k_4 é o *overflow*
- A operação de split de v faz:
  - Transforma v em dois nós: v' e v''
  - v' com chaves k_1, k_2 e filhos v_1, v_2, v_3; e
  - v'' com chave k_4 e filhos v_4, v_5
- Se v for raíz de T, crie um novo nó raíz u e faça u pai de v
- Insira k_3 em u e faça v' e v'' serem filhos de u

Genericidade da inserção de um nó em árvore 2–3–4



Análise da inserção

- *split* afeta um número constante de nós, logo $O(1)$ corresponde a complexidade inferior
- Um *split* pode gerar *overflow* no nó pai e propagar este *overflow* até a raíz, assim, o número de operações de *split* é delimitado pela altura da árvore
- $O(\log n)$, para $h = \log n$, é a complexidade superior, máxima, para uma operação de *split*

Análise da inserção

- *split* afeta um número constante de nós, logo $O(1)$ corresponde a complexidade inferior
- Um *split* pode gerar *overflow* no nó pai e propagar este *overflow* até a raíz, assim, o número de operações de *split* é delimitado pela altura da árvore
- $O(\log n)$, para $h = \log n$, é a complexidade superior, máxima, para uma operação de *split*
- Portanto $O(\text{inserção}) = \log n$

Remoção de um nó

Considerações

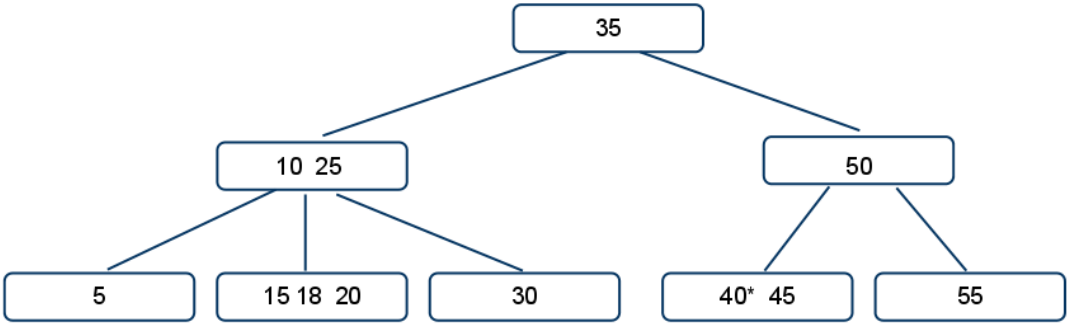
- Remover um item de chave k em T
- Operação de busca de k em T
- Remoção pode ser reduzida a remoção de um item em v, dados que os filhos de v são nós externos

Remoção de um nó

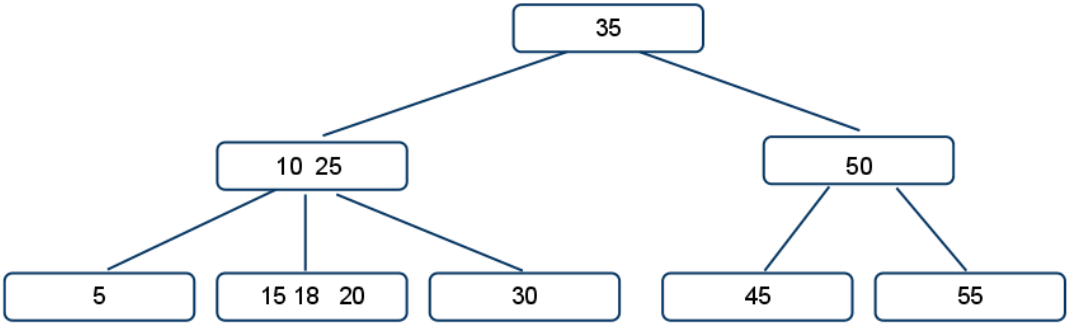
Considerações

- Remover um item de chave k em T
- Operação de busca de k em T
- Remoção pode ser reduzida a remoção de um item em v, dados que os filhos de v são nós externos
- Vamos remover o elemento **40** da árvore a seguir

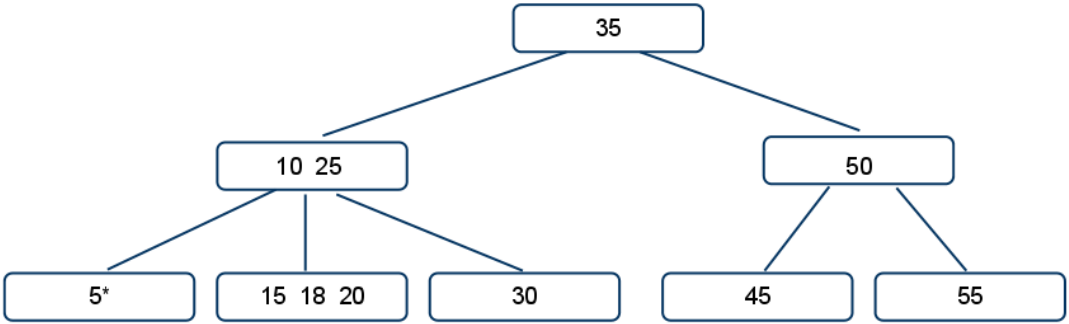
Remoção em nó folha da árvore, **40**



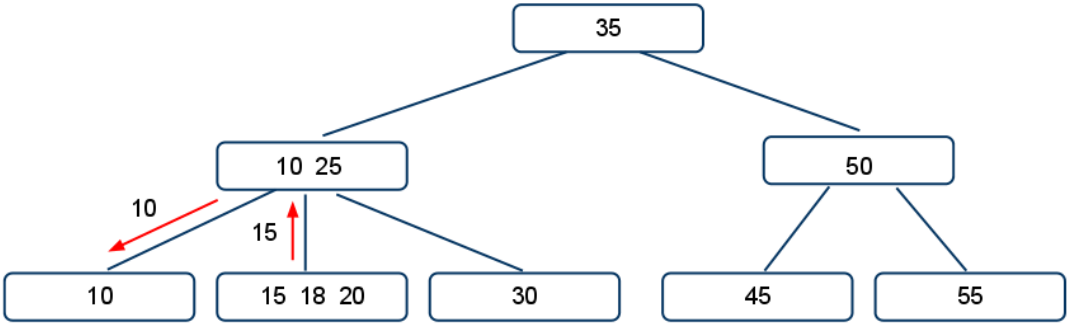
Elemento **40** removido



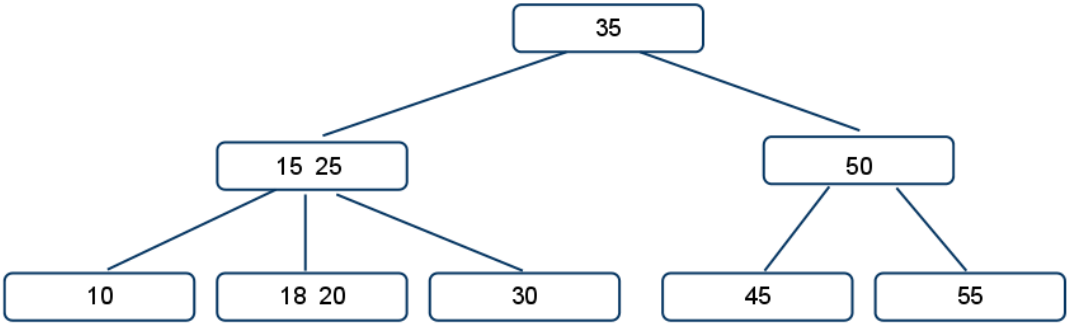
Removendo o **5**



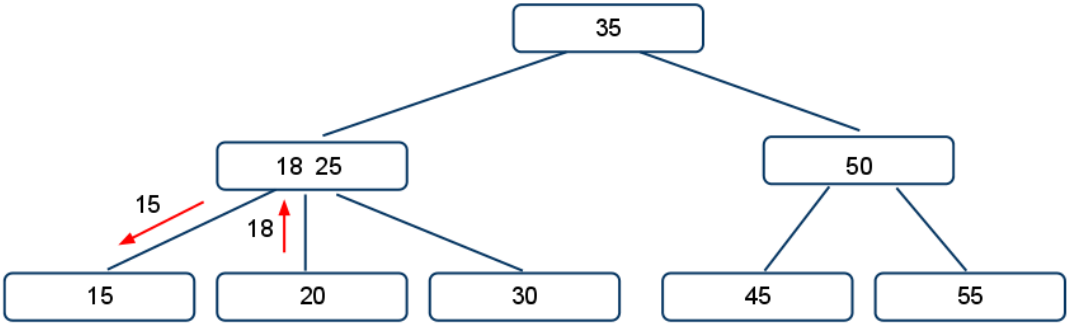
Pede ao pai e empresta do irmão



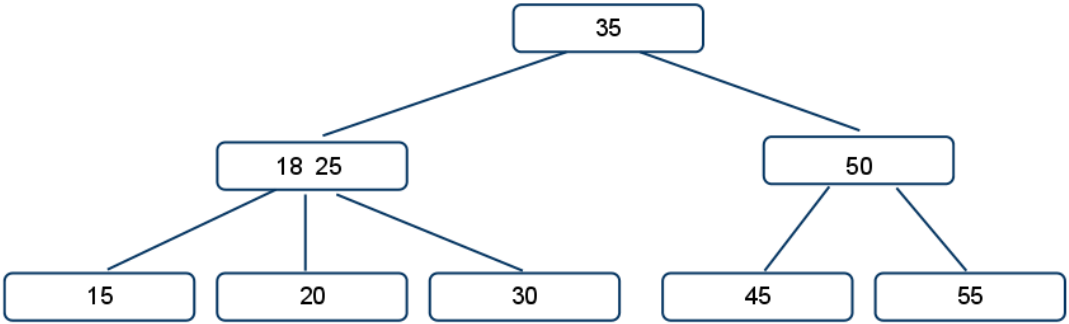
5 eliminado



Mais uma remoção, agora o **10**



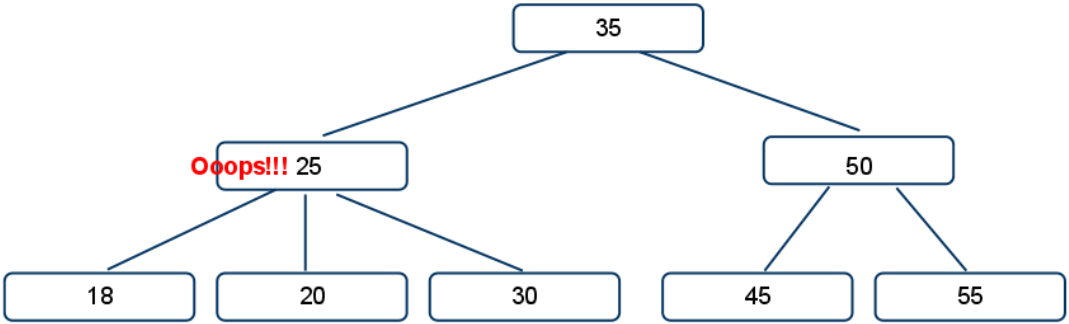
E se tirar o **15**?



Remoção... ainda não está OK!

- Se tirarmos o 15, o pai fica com dois elementos (18 e 25) e dois filhos, o que não pode
- Se 'subirmos' o 20 para pai e 'descermos' o 18 para filho, teremos dois elementos para o nó pai com dois elementos e dois nós filhos (18 e 30), o que não pode
- então...
- Retirar o 15 e 'descer' o 18 para filhos...

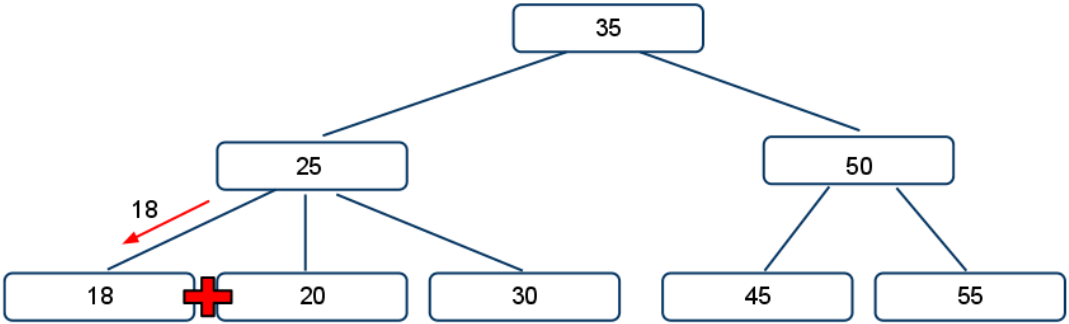
Erro! Muitos filhos para um só pai!



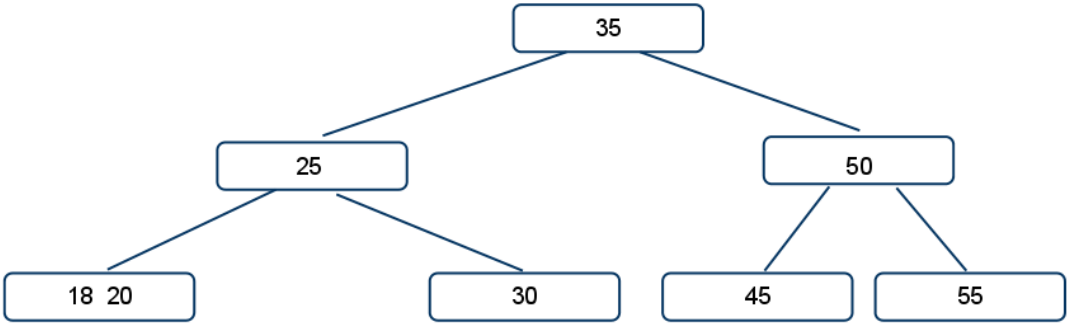
Remoção. . . ainda não está OK!

- Seriam muitos filhos para um só elemento na raíz
- Vamos fazer a fusão de nós

Fusão dos nós filhos



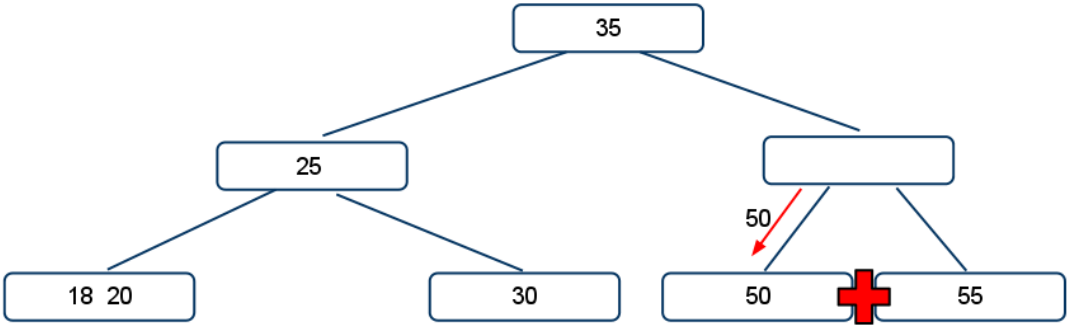
Fusão dos nós filhos terminada



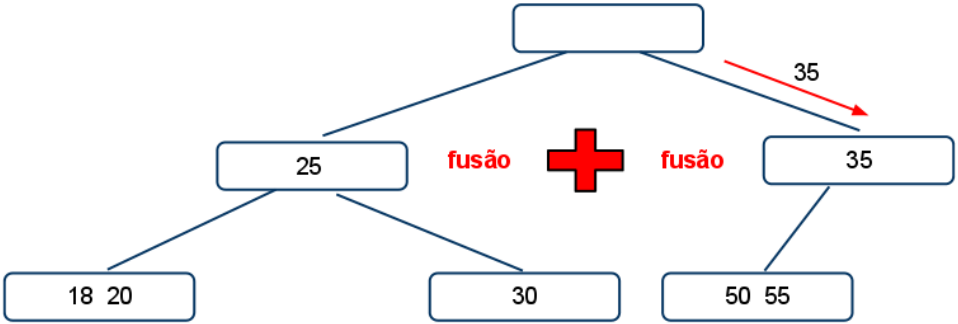
Só mais um último caso

- E se o pai tiver um único elemento?
- Por exemplo, se tirarmos o **45**...
- Quem pego para repor?
- Empresto de quem?

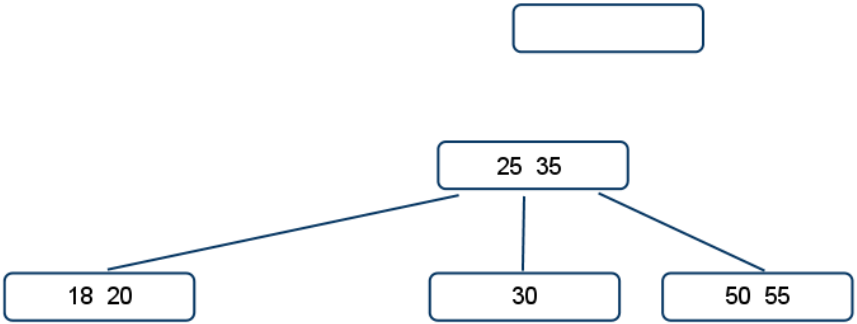
Removendo o **45**. Empresta do nó pai e funde os nós



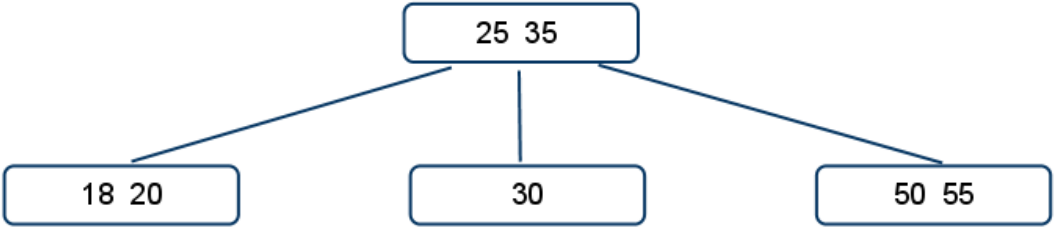
Underflow propaga até a raíz



Nova árvore. Temos que eliminar a antiga raíz



Finalmente



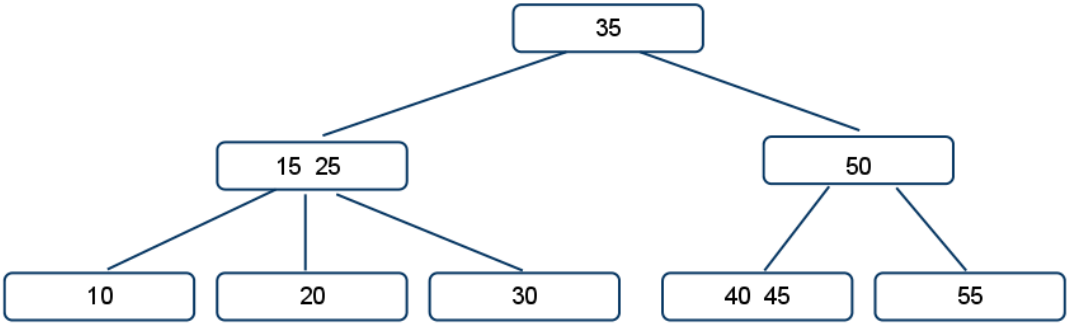
Mais um detalhe

- E se o nó a ser excluído não for um nó folha?
- E se for um nó interno?

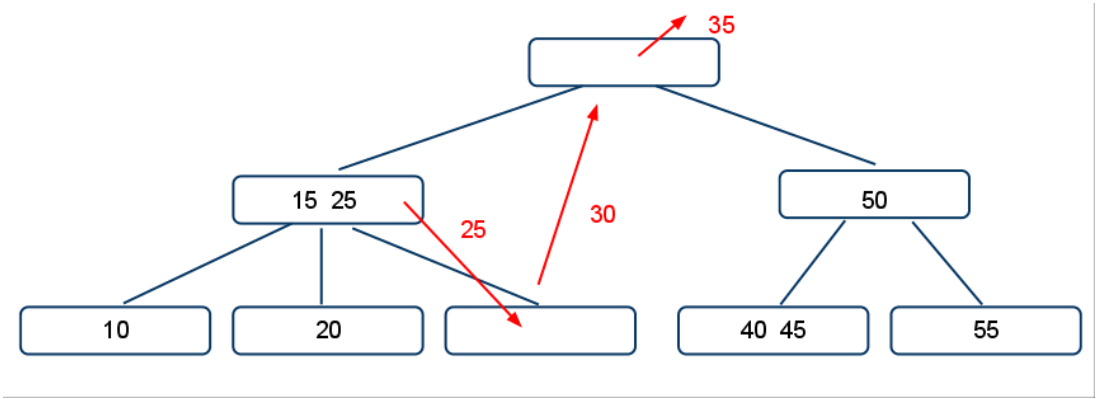
Mais um detalhe

- E se o nó a ser excluído não for um nó folha?
- E se for um nó interno?
- Trocar este nó por um nó "externo"
- Trocar o nó precedente no percurso *in-ordem*, a partir deste nó
- Vamos excluir o **35**...

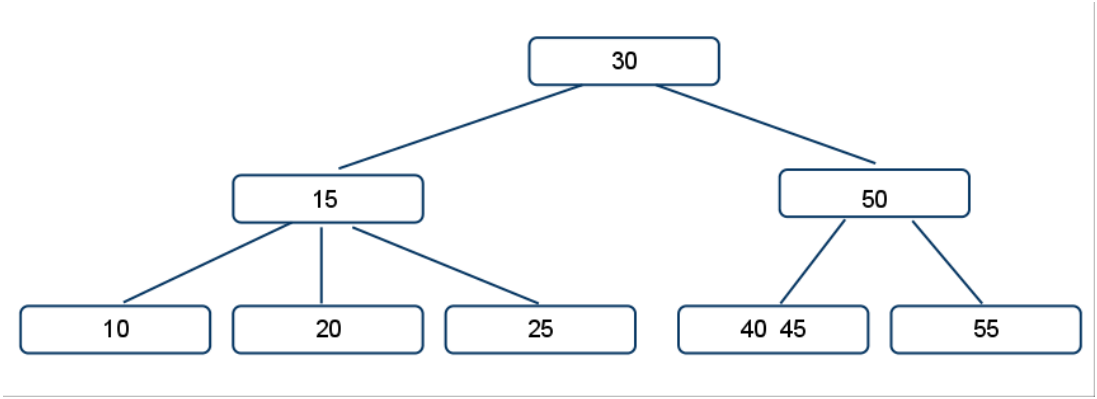
Removendo a chave **35**



Removendo a chave **35**

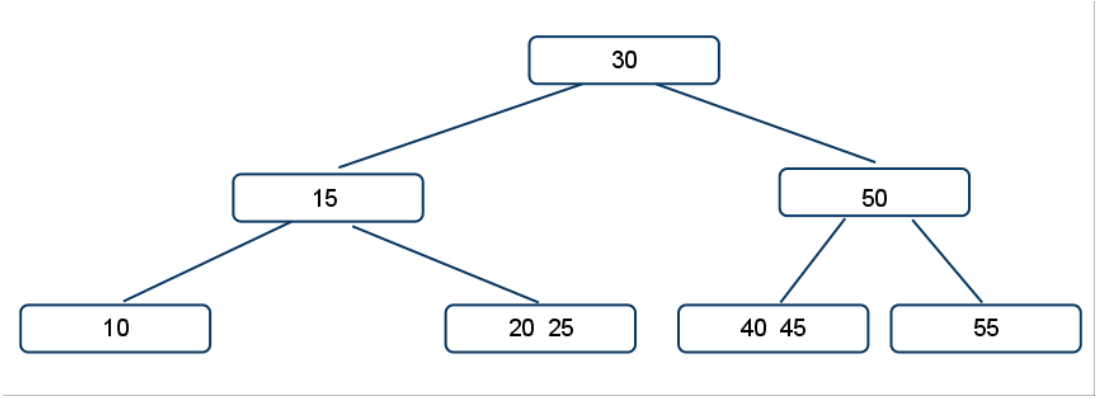


Removida a chave **35**



Reparem como o nó de chave **15** também ficou com *underflow* e não tem irmão (ou pai) que possa ajudá-lo.

Fusão de chaves!

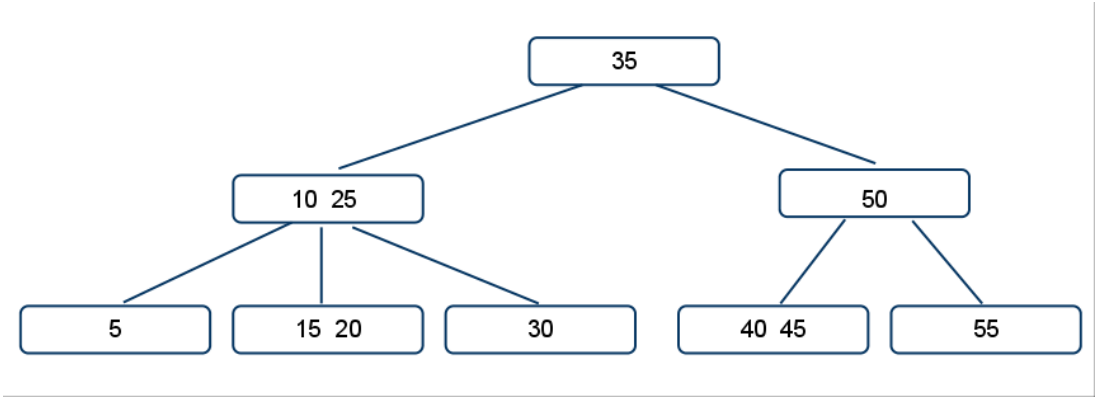


Ocorreu uma *fusão* de dois nós irmãos, os nós **20** e **25**

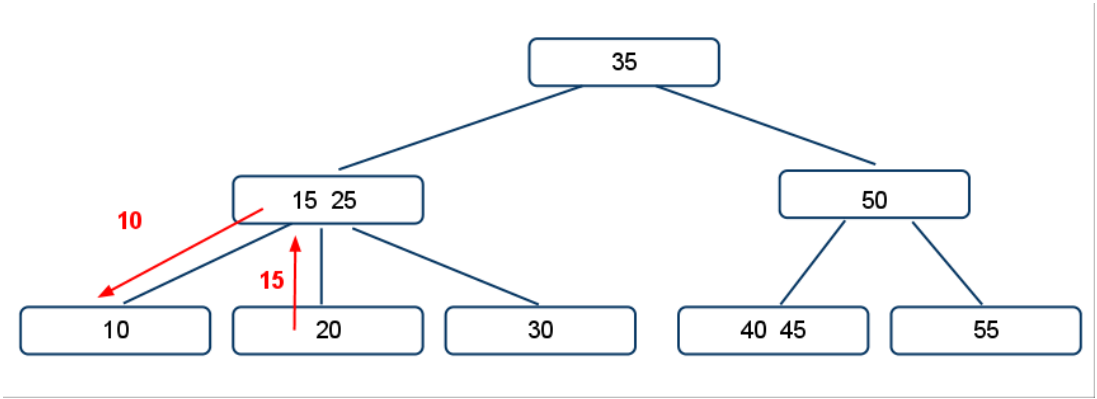
Remoção com *underflow*, continuação

- Sem itens no nó: *underflow*
- Pegue uma chave do nó pai
- Reponha esta chave com um irmão
- Exemplo: remoção do **5**

Remoção com *underflow*



Remoção com *underflow*. Empreste do pai



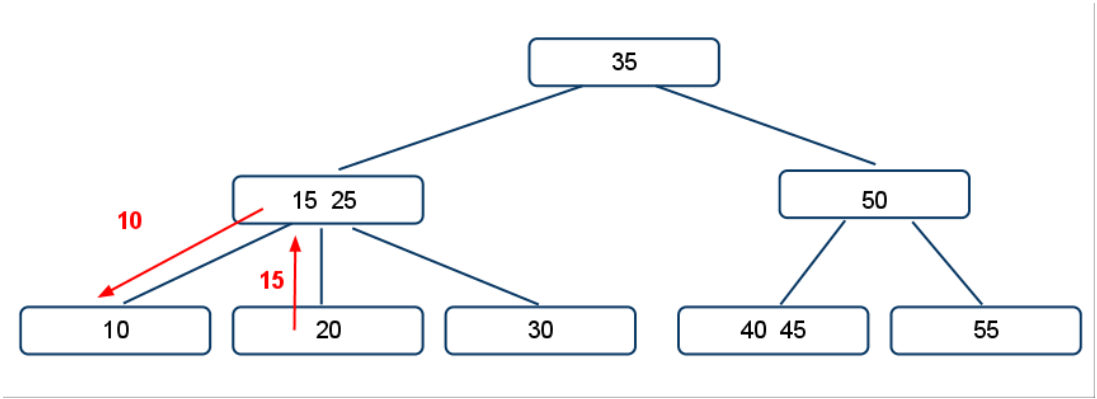
Problemas

- E caso os irmãos tiverem $M = 2$?
- Não seria bom o suficiente somente pegar emprestado do pai?
- Precisamos repor?

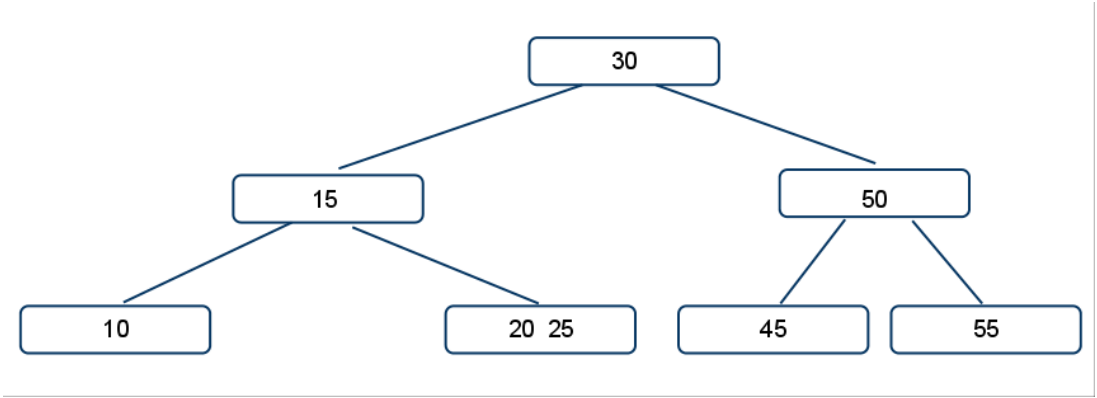
Problemas

- E caso os irmãos tiverem $M = 2$?
- Não seria bom o suficiente somente pegar emprestado do pai?
- Precisamos repor?
- Vamos chegar lá...

vamos voltar a esta situação e remover **40**



Removida a chave 40

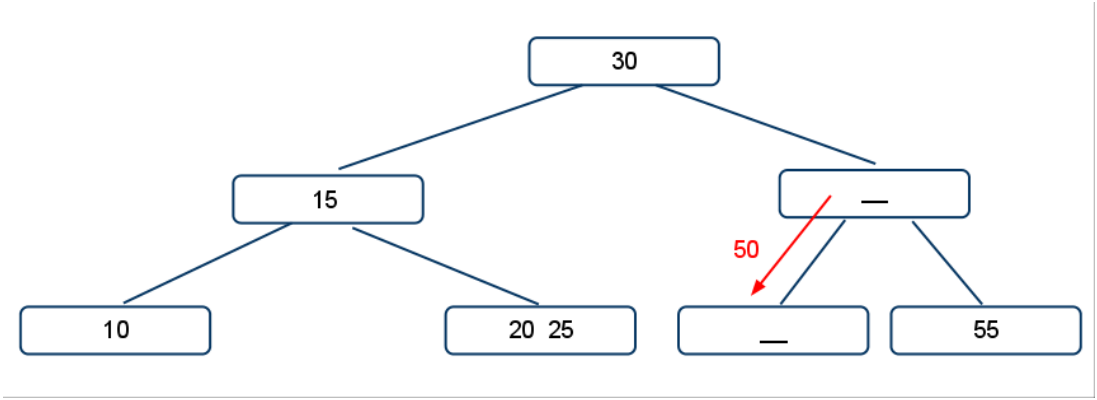


Cascata da *underflow*

Complicando...

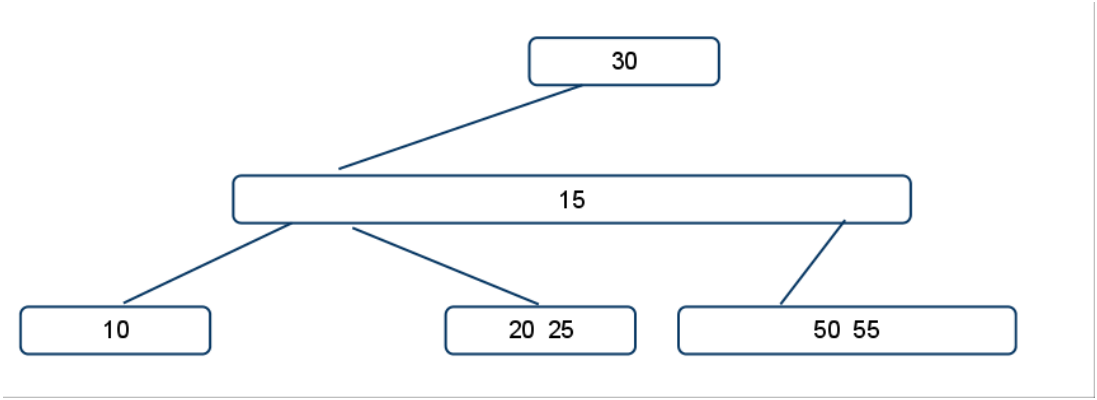
o *underflow* pode complicar, pode ocorrer uma cascata rumo ao topo da árvore.
Vamos remover a chave **45**...

Removida a chave 45



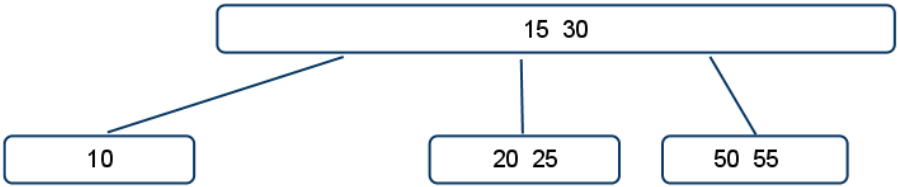
pede emprestado o pai...

Ocorrem duas fusões, 50 e 55, e entre os pais



vejam que o nó 15 também se fundiu...

Encurtamento da árvore



As fusões alteraram a altura da árvore mas as propriedades se mantiveram. Vejam como o nó pai ficou vazio.

Operações básicas

- Pesquisa (busca); e
- Criação da árvore

Convenções

- Árvore 2–3–4 está sempre na memória principal
- A leitura do disco, para qualquer elemento, nunca é exigida
- A escrita do nó raíz é exigida sempre que o nó raíz for modificado

Pesquisa numa árvore 2–3–4

- A pesquisa numa árvore 2–3–4 é semelhante a pesquisa em árvores binárias de busca
- Note que a decisão não é binária
- A decisão é ramificada em várias vias, de acordo com o número de filhos
- Vejamos agora um pseudo-código clássico que emula uma árvore 2–3–4

Pesquisa em árvore 2–3–4

Raíz t, chave p a ser pesquisada. Retorna o valor x na posição i

```
btree_search (t, p)
  i=1
  while i <= 3 (numero chaves) e p > chave ki
    do i = i+1
  if i <= 3 e p==ki
    return (x, i)
  if folha(x)
    return NULL
  else
    acesse o filho (ci) // filho
    return btree_search (ci, p)
```

Pesquisa, segundo Ziviani

Busca pela chave p na árvore de raíz t. Retorna o valor x

```
btree_search (t, p)
  i=1
  while i <= 3 e p > chave ki
    do i = i+1
  if p==ki
    x = ki
  else
  if p < ki
    btree_search(k(i-1),p)
  else
    btree_search(k, p)
```

Criação de uma árvore 2–3–4

- Para construir uma árvore 2–3–4 precisamos:
  1. Criar um nó raíz vazio, e depois...
  2. Inserir novas chaves
- Usamos, nos dois caso, um procedimento de alocação de nós

Algoritmo de criação da árvore 2–3–4

```
btree_create (t2_4)
  x = aloca_no()
  x.folha = true
  x.n = 0 //numero de chaves
  escreva_em (x)
  t2_4.raiz = x
```

Algoritmo genérico de inserção em árvores 2–3–4

- Obviamente temos que fazer uma busca do nó antes da inserção e;
- Só prosseguimos na inserção se não encontrarmos este nó na árvore.

Algoritmo genérico de inserção em árvores 2–3–4

```
Inserir (registro k)
  encontre uma chave para inserir k
  enquanto
     encontre posicao apropriada para k
     se pagina nao cheia
       insira k; retorne
     senao
        divida pagina em p1 e p2
        distribua as chaves e ponteiros
        se pagina for raiz
          crie nova raiz como ascendente de p1 e p2
          coloque mediano na raiz; ajuste ponteiros
          retorne
        senao
          pagina = seu ascendente
          processe ascendente do no
```

Remoção de chaves em árvores 2–3–4

- Mais uma vez, iniciaremos com um algoritmo simples e;
- Vamos ajustando conforme os novos casos;
- Por hora, o nó k está armazenado no nó v cujos filhos são nós externos, ou seja, vazios.

Algoritmo inicial de remoção em árvores 2–3–4

Simples!

```
Se k esta_em v
  elimina k
  elimina filhos apontados por k
```

- Removemos somente nós externos, portanto
- h da árvore não foi alterada;
- Mas...

Algoritmo inicial de remoção em árvores 2–3–4

Simples!

```
Se k esta_em v
  elimina k
  elimina filhos apontados por k
```

- Removemos somente nós externos, portanto
- h da árvore não foi alterada;
- Mas...
- Nó v pode ter duas chaves e;
- Eliminando-se uma delas ocorre *underflow* do nó

Remoção em caso de *underflow*

Seja nó u o pai do nó v

```
v tem irmão w ordem 3 ou 4
  operação de transferência
    move elemento de w para v
    move chave de w para u
    move chave de u para v
```

Remoção em caso de *underflow*, refinamento

- Mas v pode não ter irmão

Remoção em caso de *underflow*, refinamento

- Mas v pode não ter irmão
- Fusão de dois nós

Remoção em caso de *underflow*

Seja nó u o pai do nó v

```
se v não tem irmão
  operação de fusão
    fusão de v com w
    criação de v'
    /* v' só tem uma chave */
    move chave de u para v'
```

Assim as propriedades da árvore ficam preservadas

Remoção: caso final

- Lembram-se que estas remoções só funcionam se a chave estiver num nó externo...
- O que ocorre se k estiver num nó **interno**

Remoção: caso final

- Lembram-se que estas remoções só funcionam se a chave estiver num nó externo...
- O que ocorre se k estiver num nó **interno**
- Troca!

Remoção: caso final

- Lembram-se que estas remoções só funcionam se a chave estiver num nó externo...
- O que ocorre se k estiver num nó **interno**
- Troca!
- Troque a chave interna com o maior elemento da sub-árvore esquerda;
- Esta será a menor chave mais próxima de k; então
- Remova k como procedemos

Projeto

Codificar

As principais operações numa árvore 2–3–4.