Árvores B, B* e B+

Rômulo César Silva

Unioeste

Setembro de 2016

Sumário

Árvore B

Histórico

- Desenvolvida por Bayer e McCreight no Boeing Scientific Research Labs em 1972, para organização e manutenção de arquivos.
- A origem do termo "árvore-B" é controverso: se "B" refere-se à Boeing ou Bayer.

Árvore B

Motivação

- O armazenamento de grande quantidade de dados de maneira persistente é geralmente feito em discos magnéticos, que possuem custo de acesso alto quando comparado ao acesso feito em memória RAM.
- Assim, é necessário diminuir o número de acessos a disco. Solução: **agrupar várias chaves em um mesmo nó**
- Criar estruturas em que as operações de inserção, busca e remoção possam ser feitas de maneira eficiente.

Árvore B

Aviso!!

A definição dos termos usados para caracterização das árvores B não é uniforme na literatura.

A definição desses termos usados como parâmetros da árvore B direcionam sua implementação.

Árvore B

Definição

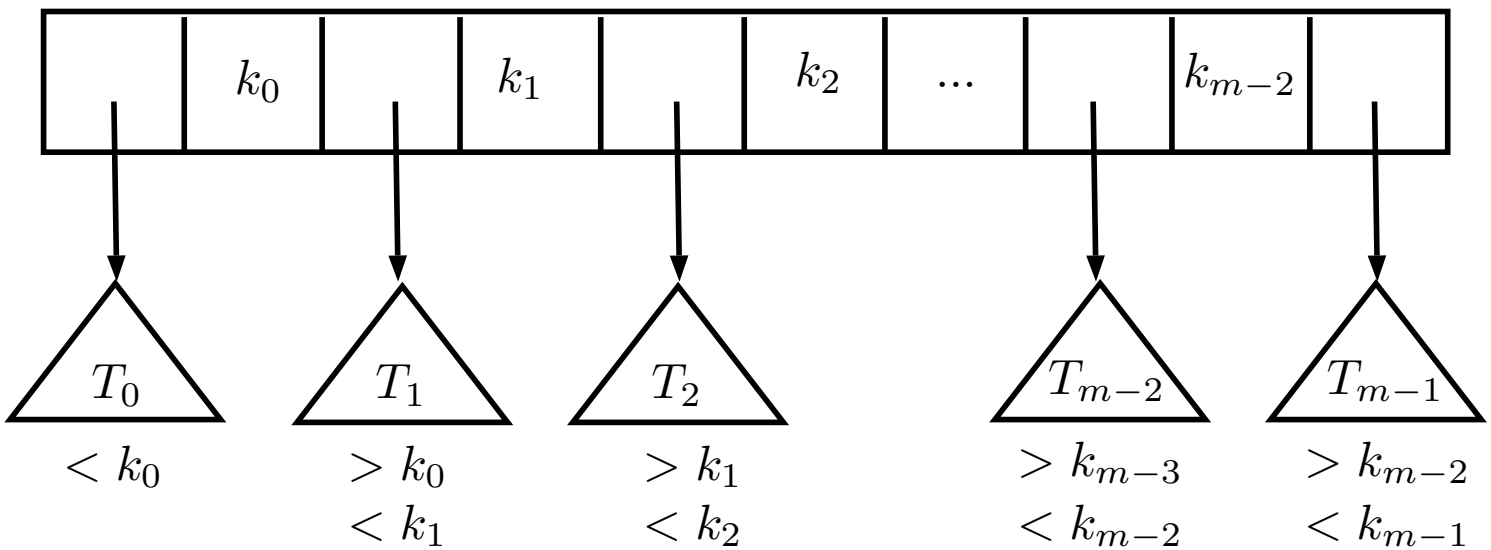
Uma **árvore B** de ordem m satisfaz as seguintes propriedades:

1. Cada nó tem no máximo m filhos
2. Cada nó não-folha (exceto a raiz) tem no mínimo $\lceil \frac{m}{2} \rceil$ filhos
3. A raiz tem no mínimo 2 filhos se ela não é folha.
4. Um nó não-folha com d filhos tem $d - 1$ chaves.
5. Todas as folhas aparecem no mesmo nível.

Árvore B

Definição

6. As chaves separam os intervalos de chaves armazenados em cada subárvore conforme o esquema abaixo:



Árvore B

Teorema

Para qualquer árvore B com $n \geq 1$ chaves de altura h e grau mínimo $t \geq 2$,

$$h \leq log_t \frac{n+1}{2}$$

Árvore B

Nomenclatura

- A **ordem** de uma árvore B, denotada por m, corresponde ao número máximo de filhos que um nó pode ter. Assim, cada nó de uma árvore B de ordem 5 pode ter no máximo 4 chaves e 5 filhos.
- É comum usar o termo **página** ao invés de **nó**.

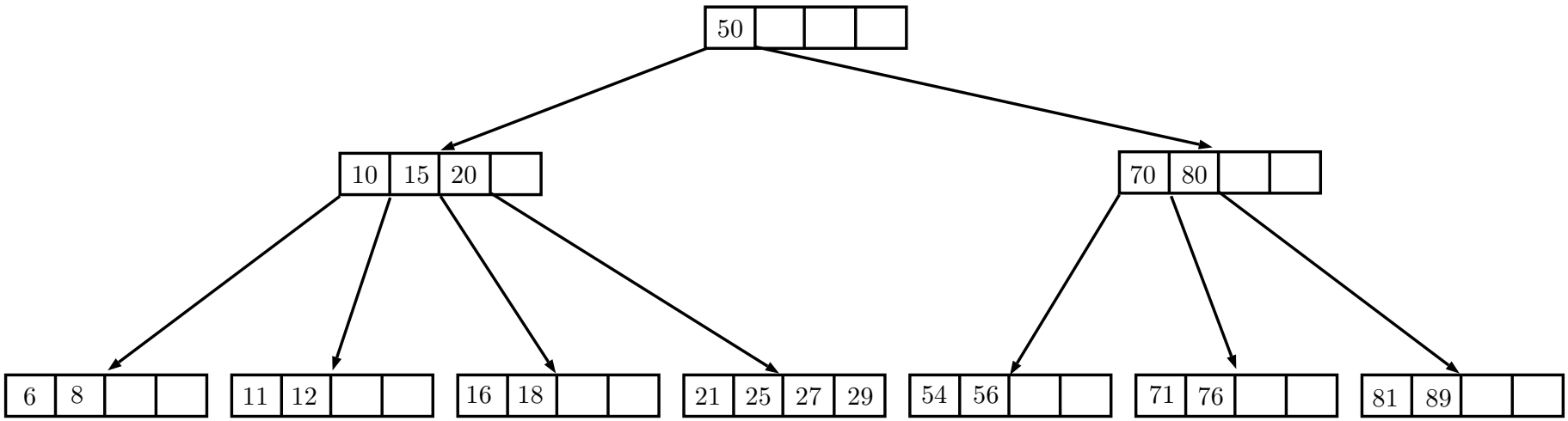
Obs.: Há autores que definem ordem como sendo o número mínimo de chaves em um nó.

Árvore B

Nomenclatura

- Um nó é dito **cheio, se contém o número máximo de chaves permitido para a ordem definida da árvore B.**

Árvore B: exemplo



Árvore B - Inserção

Inserção

A inserção em uma árvore-B sempre é feita nas folhas.

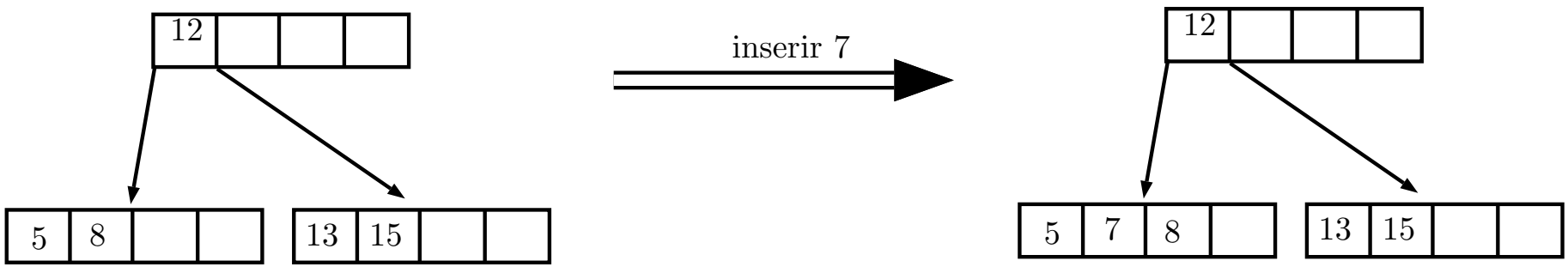
- quando o nó já está cheio, ele é particionado (*split*) em 2 nós e as chaves são distribuidos uniformemente pelos 2 nós, e a chave mediana é promovida para o nó pai.
- se necessário, o processo de *split* é repetido nos níveis superiores
- a altura da árvore aumenta quando o processo de *split* chega à raiz, situação em que uma nova raiz é criada.

Obs.: **para facilitar a implementação será possível um nó ter 1 chave e 1 filho a mais que o permitido, ficando com** *overflow*, **sendo responsabilidade do nó-pai restaurar as propriedades da árvore B.**

Árvore B - Inserção

Inserção - Caso 1

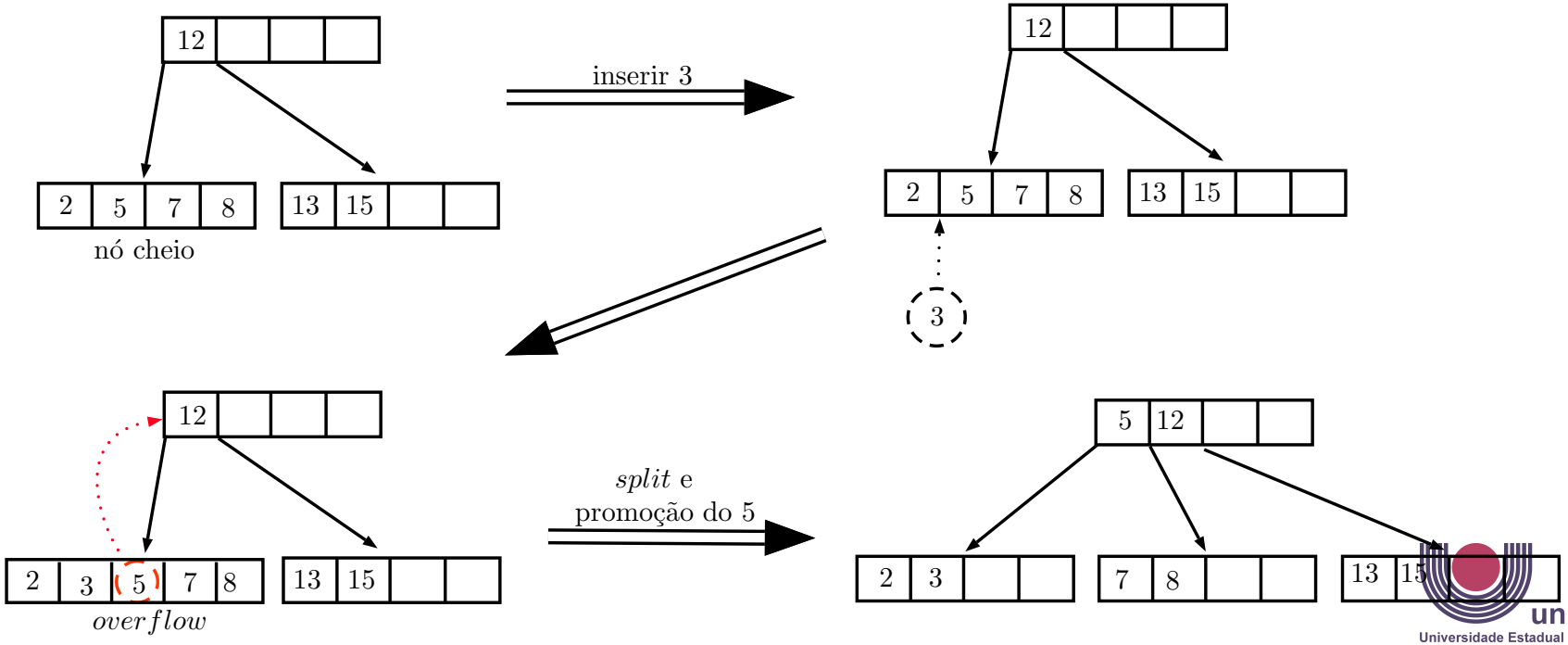
A chave é inserida em uma folha que ainda tem espaço. Faz-se deslocamento de chaves dentro do nó, se necessário



Árvore B - Inserção

Inserção - Caso 2

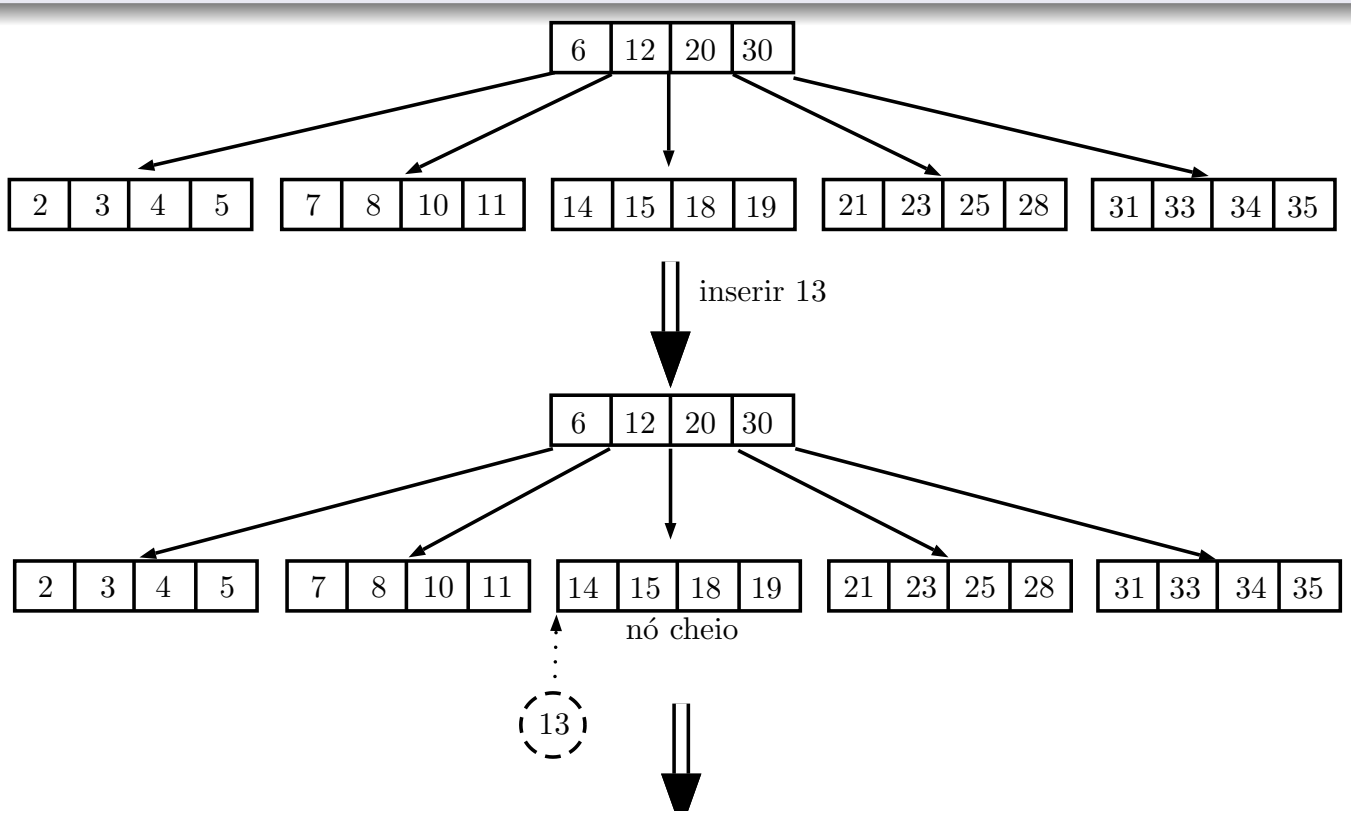
A folha na qual a chave deve ser inserida está cheia. Faz-se o *split* do nó e metade das chaves migram da folha cheia para a folha nova. A chave mediana é promovida para o nó pai e liga-se a folha nova ao pai.



Árvore B - Inserção

Inserção - Caso 3

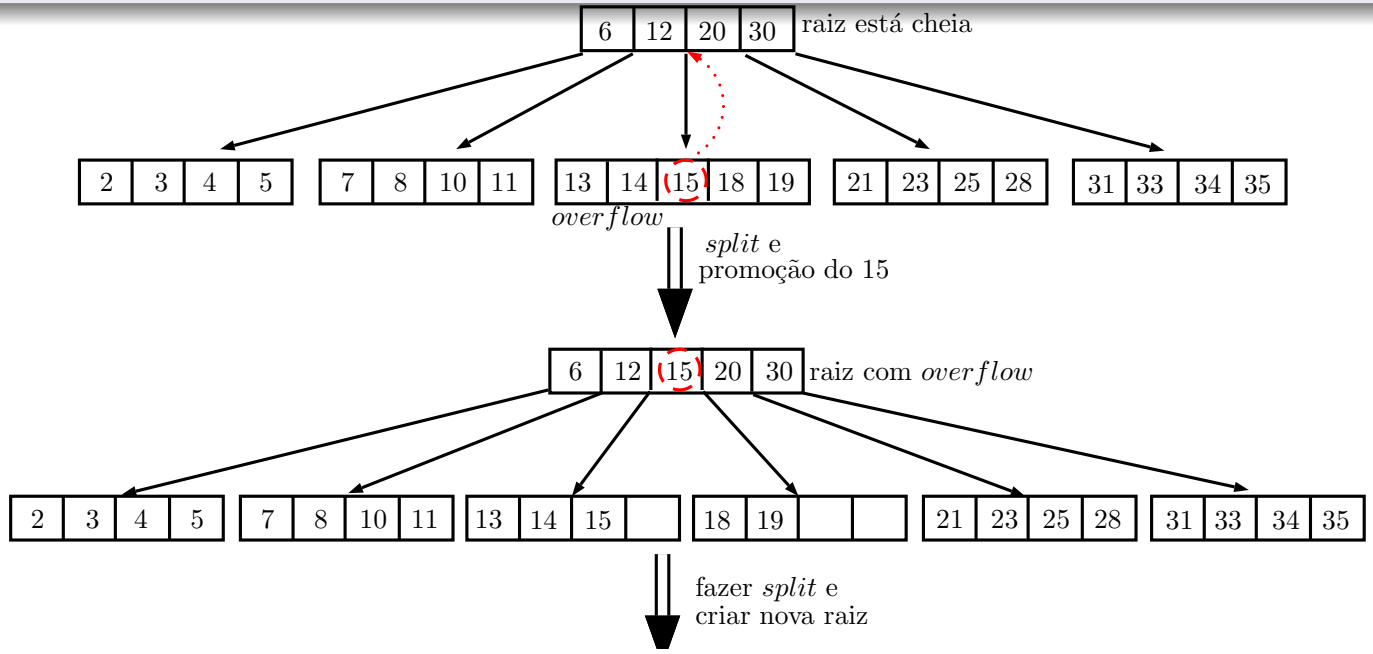
A raiz está cheia e a inserção provoca o *split* na raiz, criando uma nova raiz, e o aumento da altura da árvore.



Árvore B - Inserção

Inserção - Caso 3 (cont.)

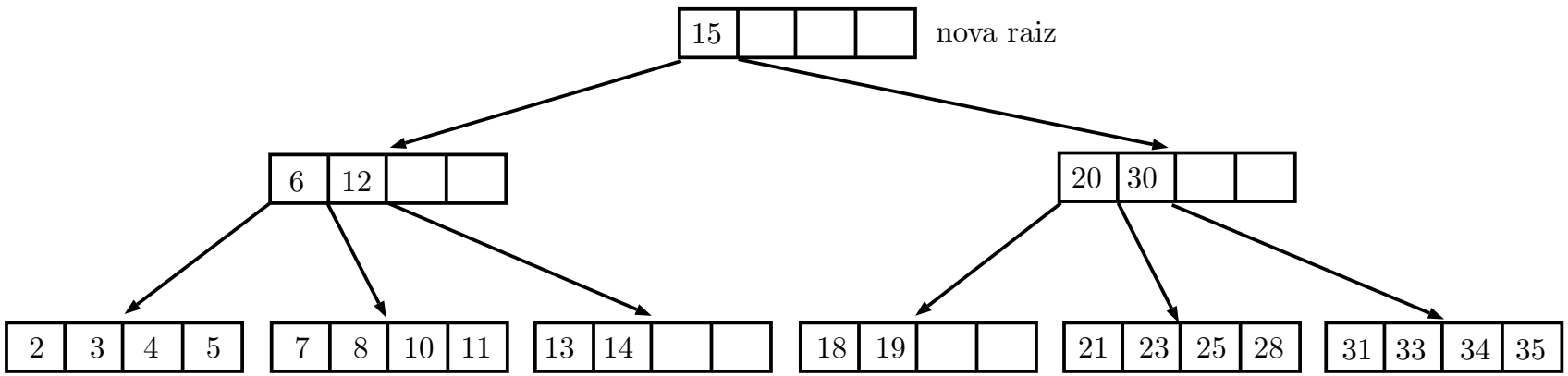
A raiz está cheia e a inserção provoca o *split* na raiz, criando uma nova raiz, e o aumento da altura da árvore.



Árvore B - Inserção

Inserção - Caso 3 (cont.)

A raiz está cheia e a inserção provoca o *split* na raiz, criando uma nova raiz, e o aumento da altura da árvore.



Árvore B - estrutura

```
#define ORDEM 5

//estrutura de nó para árvore B:
//há 1 posição a mais de chave e ponteiro de filho para
//facilitar a implementação da operação split

typedef struct no {
    int numChaves;
    int chave[ORDEM];
    struct no* filho[ORDEM+1];
} arvoreB;
```

Árvore B - Inserção

Funções auxiliares

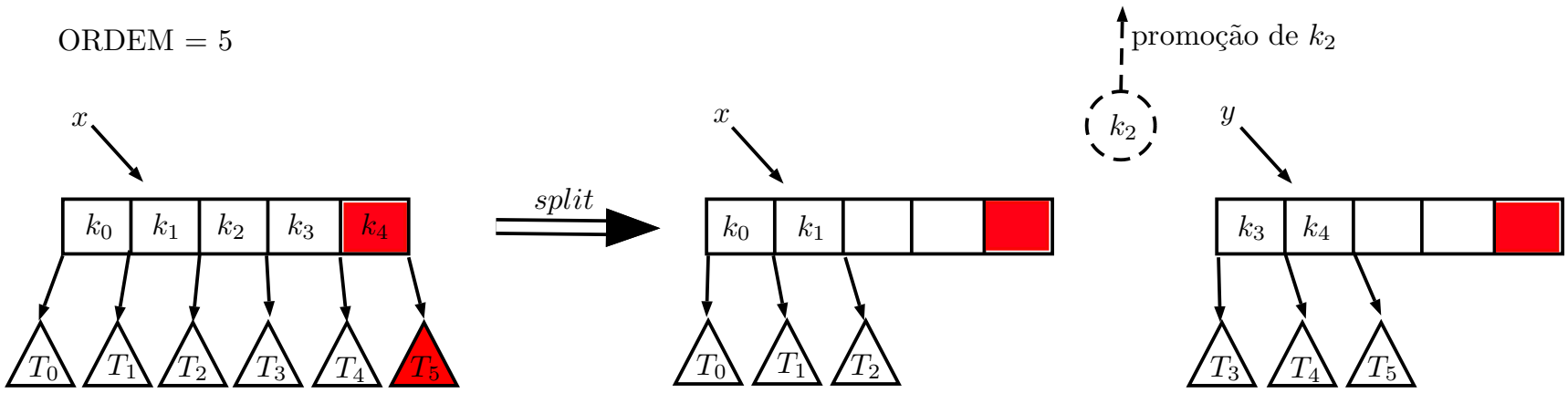
- `buscaPos`: busca a posição em que está ou estaria uma chave em um nó. Retorna 1 se a chave está presente ou 0 caso contrário
- `eh_folha`: testa se o nó é folha.
- `split`: quebra um nó com *overflow* em 2 nós e separa a chave mediana.
- `adicionaDireita`: adiciona a um nó 1 chave com 1 filho a um nó em uma determinada posição deslocando se necessário as outras chaves para a direita.
- `insere_aux`: insere uma chave em uma árvore B não vazia.

A função principal da inserção (`insere`) trata as condições especiais de árvore vazia e criação de nova raiz.

Árvore B - Split

Esquema de implementação do *Split*

Um nó x com *overflow*, que contém 1 chave e filho a mais que o permitido (marcados em vermelho) é quebrado em 2 nós (x e y), e a chave mediana é promovida para o nó-pai.



Árvore B: *split*

```
//Quebra o nó x (com overflow) e retorna o nó criado e chave m que
// deve ser promovida

arvoreB* split(arvoreB* x, int * m) {
  arvoreB* y = (arvoreB*) malloc(sizeof(arvoreB));
  int q = x->numChaves/2;
  y->numChaves = x->numChaves - q - 1;
  x->numChaves = q;
  *m = x->chave[q]; // chave mediana
  int i = 0;
  y->filho[0] = x->filho[q+1];
  for(i = 0; i < y->numChaves; i++){
     y->chave[i] = x->chave[q+i+1];
     y->filho[i+1] = x->filho[q+i+2];
  }
  return y;
}
```

Árvore B: inserção

Função auxiliar de busca:

```
// busca a posição em que a chave info está ou estaria em um nó
// retorna 1 se a chave está presente ou 0 caso contrário

int buscaPos(arvoreB* r, int info, int * pos) {
  for((*pos)=0; (*pos) < r->numChaves; (*pos)++)
    if(info == r->chave[(*pos)])
         return 1; // chave já está na árvore
    else if(info < r->chave[(*pos)])
           break; // info pode estar na subárvore filho[*pos]
  return 0; // chave não está neste nó
}
```

Função auxiliar para testar se um nó é folha:

```
int eh_folha(arvoreB* r) {
  return (r->filho[0] == NULL);
}
```

Árvore B: inserção

Função auxiliar para adicionar a um nó uma chave com filho:

```
void adicionaDireita(arvoreB* r, int pos, int k, arvoreB* p){
  int i;
  for(i=r->numChaves; i>pos; i--){
    r->chave[i] = r->chave[i-1];
    r->filho[i+1] = r->filho[i];
  }
  r->chave[pos] = k;
  r->filho[pos+1] = p;
  r->numChaves++;
}
```

Árvore B: inserção

Função auxiliar para adicionar uma chave à árvore B não vazia

```
void insere_aux(arvoreB* r, int info){
  int pos;
  if(!buscaPos(r,info, &pos)){ // chave não está no nó r
    if(eh_folha(r)) {
      adicionaDireita(r,pos,info,NULL);
    }
    else {
      insere_aux(r->filho[pos],info);
      if(overflow(r->filho[pos])){
        int m; // valor da chave mediana
        arvoreB* aux = split(r->filho[pos],&m);
        adicionaDireita(r,pos,m,aux);
      }
    }
   }
}
```

Árvore B: função principal de inserção

```
// Insere uma chave na árvore B fazendo split da raiz se necessário
// retorna a nova raiz
arvoreB* insere(arvoreB* r, int info){
  if(vazia(r)) {
    r = malloc(sizeof(arvoreB));
    r->chave[0] = info;
    r->filho[0] = NULL;
    r->numChaves = 1;
  }
  else {
    insere_aux(r,info);
    if(overflow(r)){
      int m;
      arvoreB* x = split(r,&m);
      arvoreB* novaRaiz = malloc(sizeof(arvoreB));
      novaRaiz->chave[0] = m;
      novaRaiz->filho[0] = r;
      novaRaiz->filho[1] = x;
      novaRaiz->numChaves = 1;
      return novaRaiz;
    }
  }
  return r;
}
```

Árvore B

Algumas considerações

- A função auxiliar `buscaPos` que retorna a posição de um nó em que está ou estaria uma chave é aconselhável usar busca binária dentro do nó caso haja mais de 10 chaves.
- Em geral, é adotado um número ímpar para a ordem da árvore B tal que o número máximo de chaves seja par e portanto facilite a operação *split* do nó.
- Valores típicos para o número de máximo de chaves em um nó em geral são bem maiores (ex.: 100, 1024) que os aqui apresentados.
- Árvores B são mais indicadas para armazenar dados em memória secundária.
- Árvores B e suas variações são bastante utilizadas em sistemas de bancos de dados e sistemas de arquivo de sistemas operacionais

Árvore B

Algumas considerações

Considerando uma árvore B de ordem m e o número mínimo de descendentes:

Nível	número mínimo de descendentes
1	2
2	$2 \times \lceil \frac{m}{2} \rceil$
3	$2 \times \lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil = 2 \times \lceil \frac{m}{2} \rceil^2$
4	$2 \times \lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil = 2 \times \lceil \frac{m}{2} \rceil^3$
...	...
d	$2 \times \lceil \frac{m}{2} \rceil^{d-1}$

Árvore B

Algumas considerações

Sendo N o número de chaves em um nó, tendo portanto $N+1$ descendentes, d a profundidade no nível das folhas, e o número mínimo de descentes no nível d, tem-se que $N+1 \geq 2 \times \lceil \frac{m}{2} \rceil^{d-1}$.
Logo, $d \leq 1 + log_{\lceil \frac{m}{2} \rceil} \frac{N+1}{2}$.
Ex.: $m = 512$ e $N = 1.000.000$ tem-se que $d \leq 3.37$. Ou seja a árvore não tem mais que 3 níveis de altura e portanto uma pesquisa necessita no máximo de 3 acessos a disco.

Árvore B - Busca

Busca

A busca em árvore B segue o esquema:

1. procura-se a posição que a chave ocuparia se estivesse presente
2. Se encontrou a chave retorna-se o endereço do nó e a posição da chave. Caso contrário busca-se recursivamente na subárvore correspondente à posição que a chave ocuparia.

Árvore B: Busca

```
// retorna o nó que contem info e sua posição no nó ou
// NULL se info não está na árvore.
arvoreB* busca(arvoreB* r, int info, int * pos){
  if(vazia(r))
    return NULL;
  int i = 0;
  while(i < r->numChaves && r->chave[i] < info) i++;
  if((i+1) > r->numChaves || r->chave[i] > info)
    return busca(r->filho[i], info, pos);
  *pos = i;
  return r;
}
```

Árvore B - Remoção

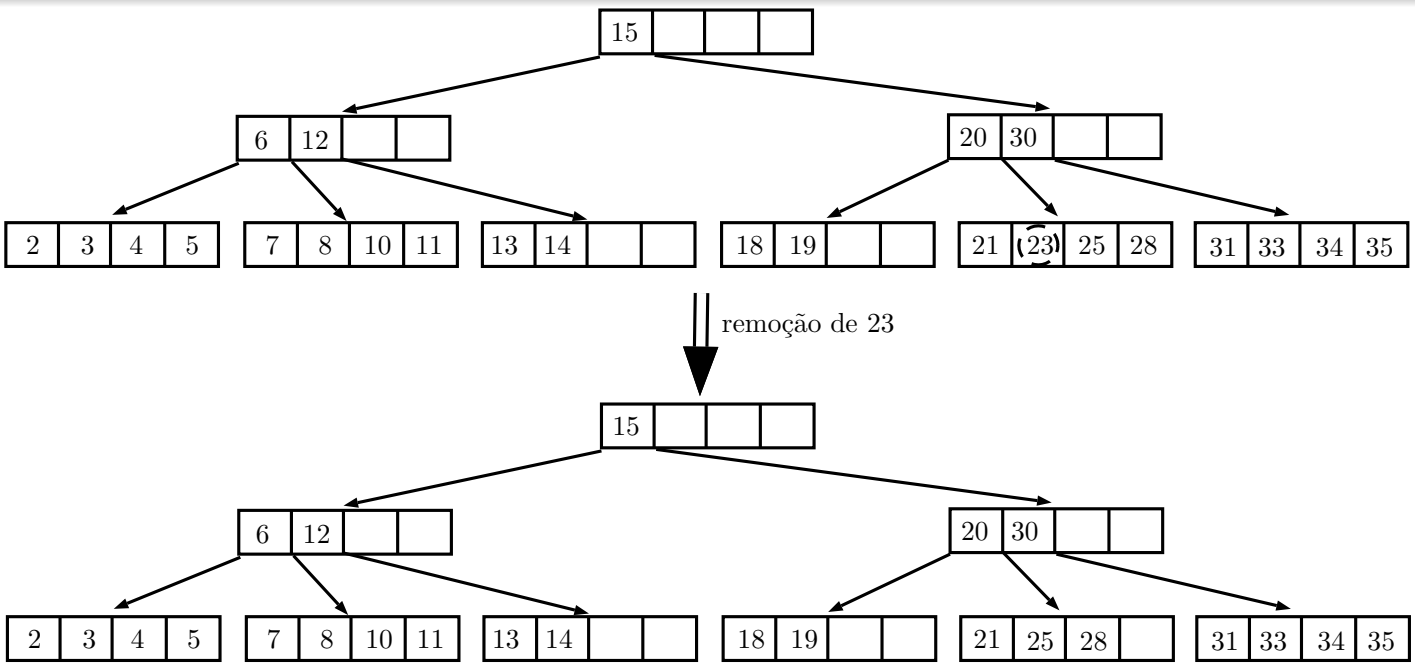
Remoção

A remoção em uma árvore B é mais complexa que a inserção porque existem vários casos. Há diferentes maneiras de implementar a remoção dependendo de otimizações em relação à quantidade de acessos ao disco ou mesmo simplicidade do código que se quer fazer.

Árvore B - Remoção

Remoção - Caso 1

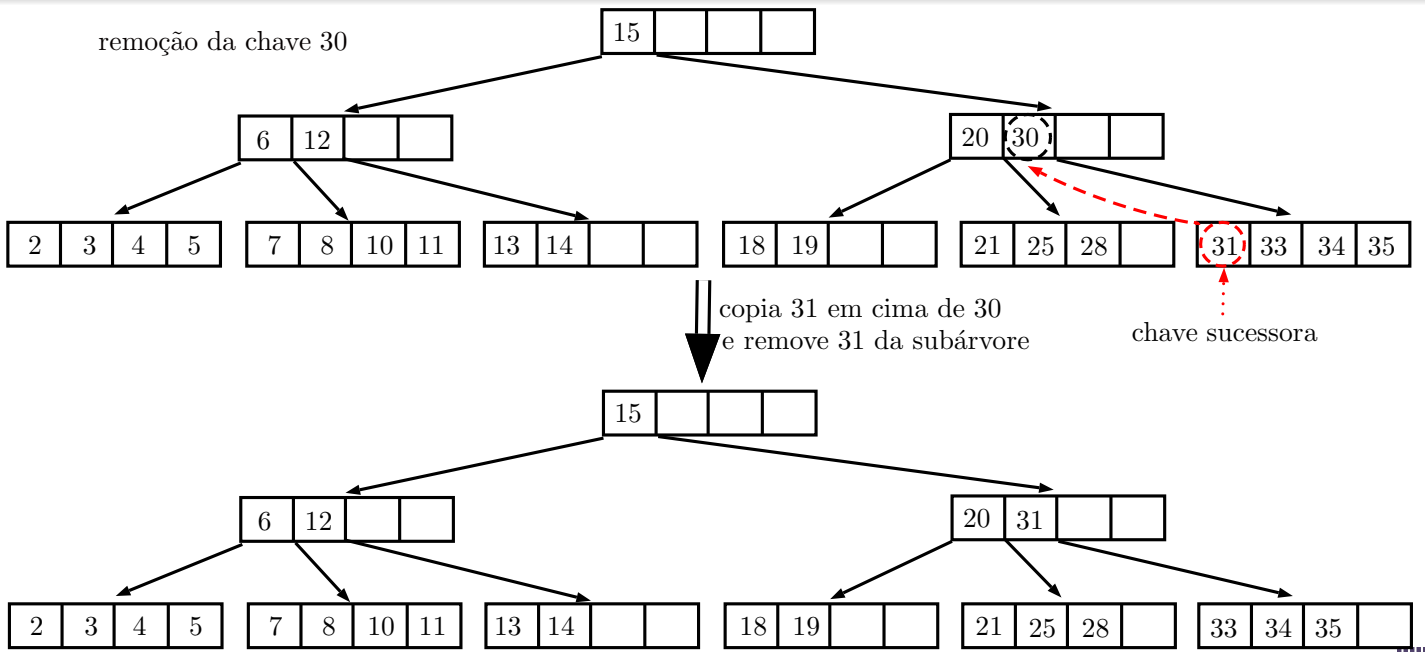
A remoção de chave que está em uma folha cujo número de chaves é maior que o mínimo. Faz-se a remoção da chave, deslocando as demais se necessário.



Árvore B - Remoção

Remoção - Caso 2 (ex. 1)

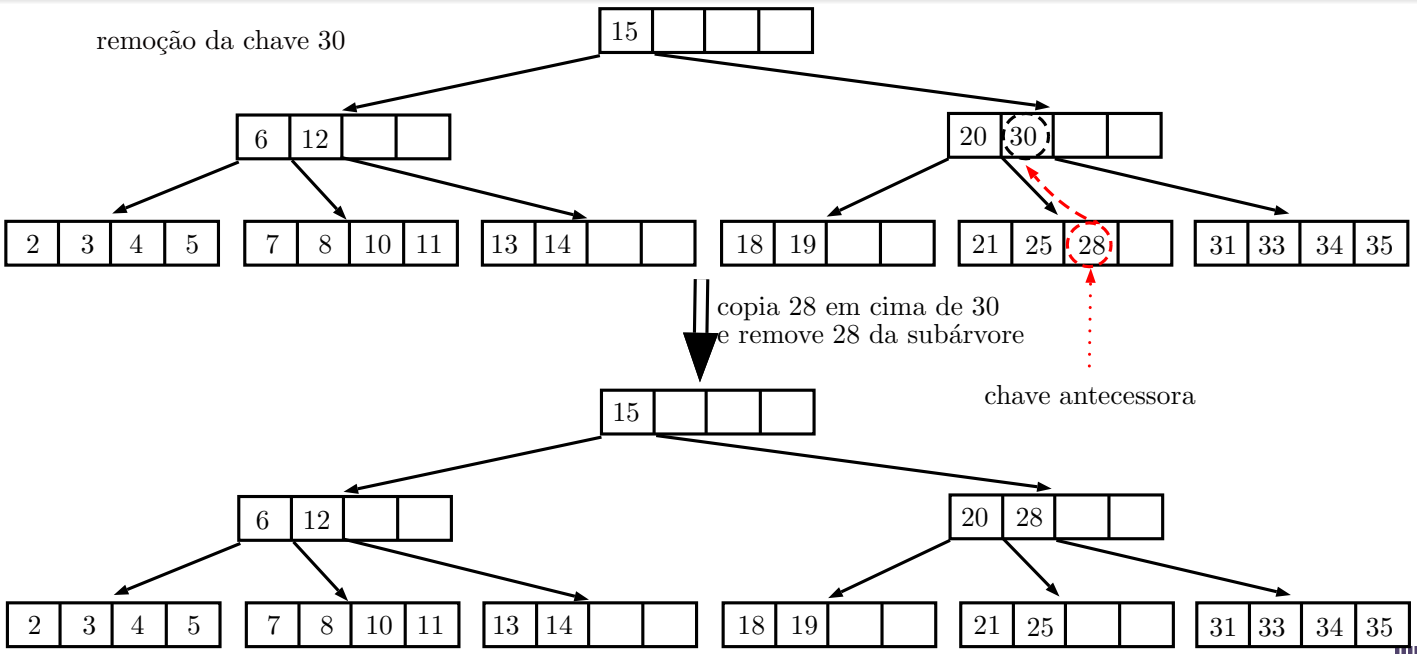
A remoção de chave que está em um nó não-folha. Copiar a chave sucessora ou antecessora (que está em uma folha) por cima da chave a ser removida, e remove-se recursivamente a chave copiada.



Árvore B - Remoção

Remoção - Caso 2 (ex. 2)

A remoção de chave que está em um nó não-folha. Copiar a chave sucessora ou antecessora (que está em uma folha) por cima da chave a ser removida, e remove-se recursivamente a chave copiada.



Árvore B - Remoção

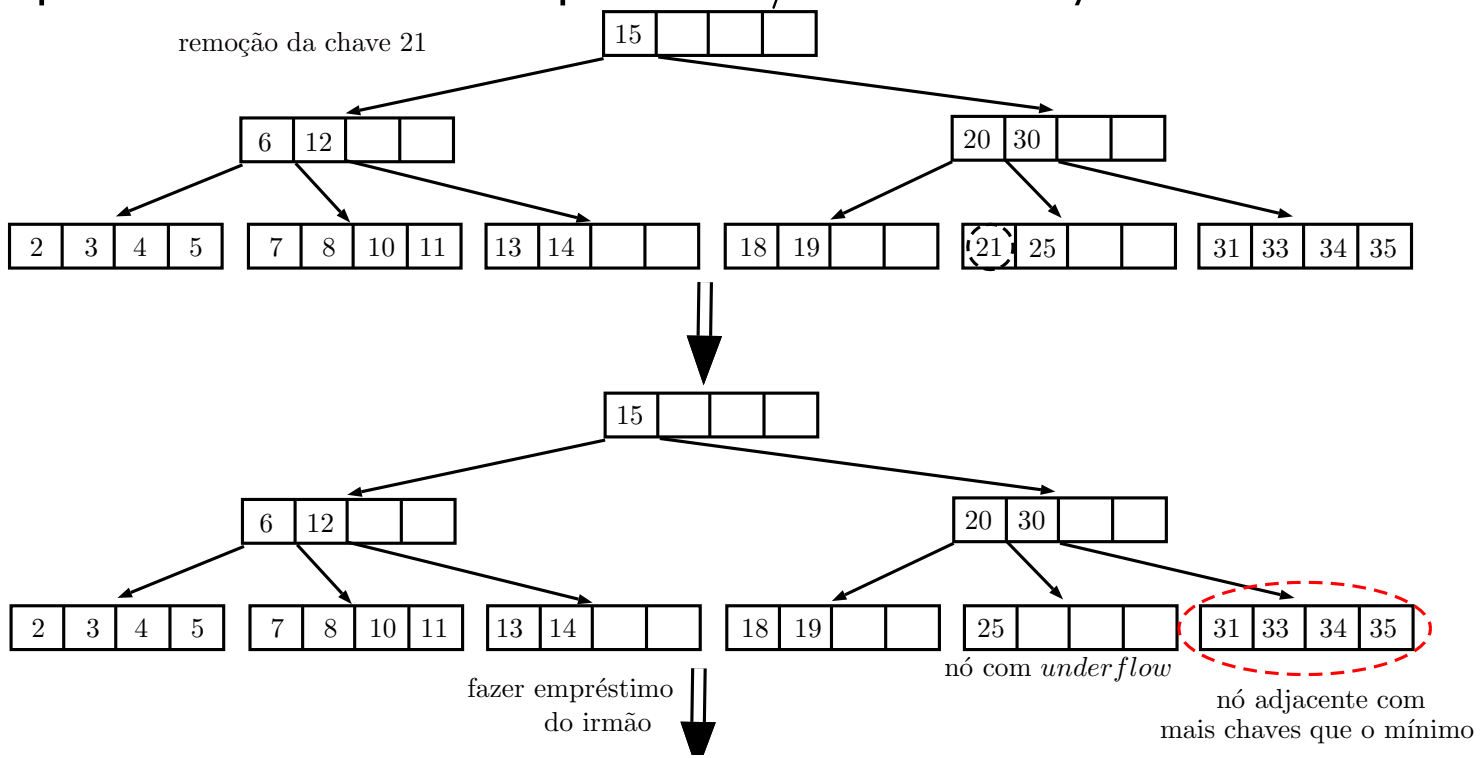
Remoção - Caso 3

A remoção de chave que está em um nó que tem o mínimo de chaves. Nesse caso a remoção causa *underflow*. Há 2 subcasos a tratar:

- Caso 3.1 (**empréstimo/redistribuição**): o nó da chave removida tem irmão adjacente cujo número de chaves é maior que o mínimo. Nessa situação é feita a redistribuição das chaves entre os nós irmãos e a chave separadora no nó-pai.
- Caso 3.2 (**concatenação**): ambos irmãos adjacentes do nó da chave removida têm o mínimo de chaves e portanto o empréstimo não pode ser efetuado. Nessa situação, será feita a concatenação do nó que sofreu a remoção com um irmão adjacente e a chave separadora do nó pai para formar um único nó. Repetir esse processo no nó-pai se ele ficou com *underflow*.

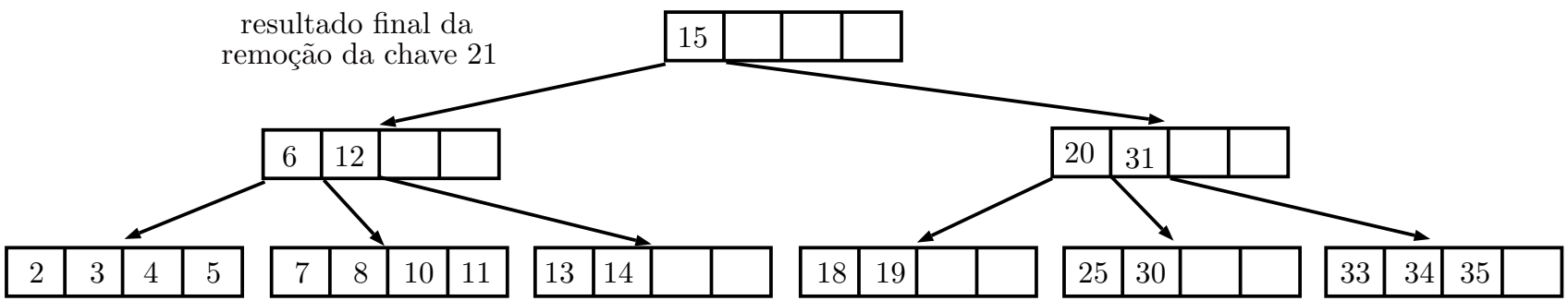
Árvore B - Remoção

Exemplo do Caso 3.1: Empréstimo/Redistribuição



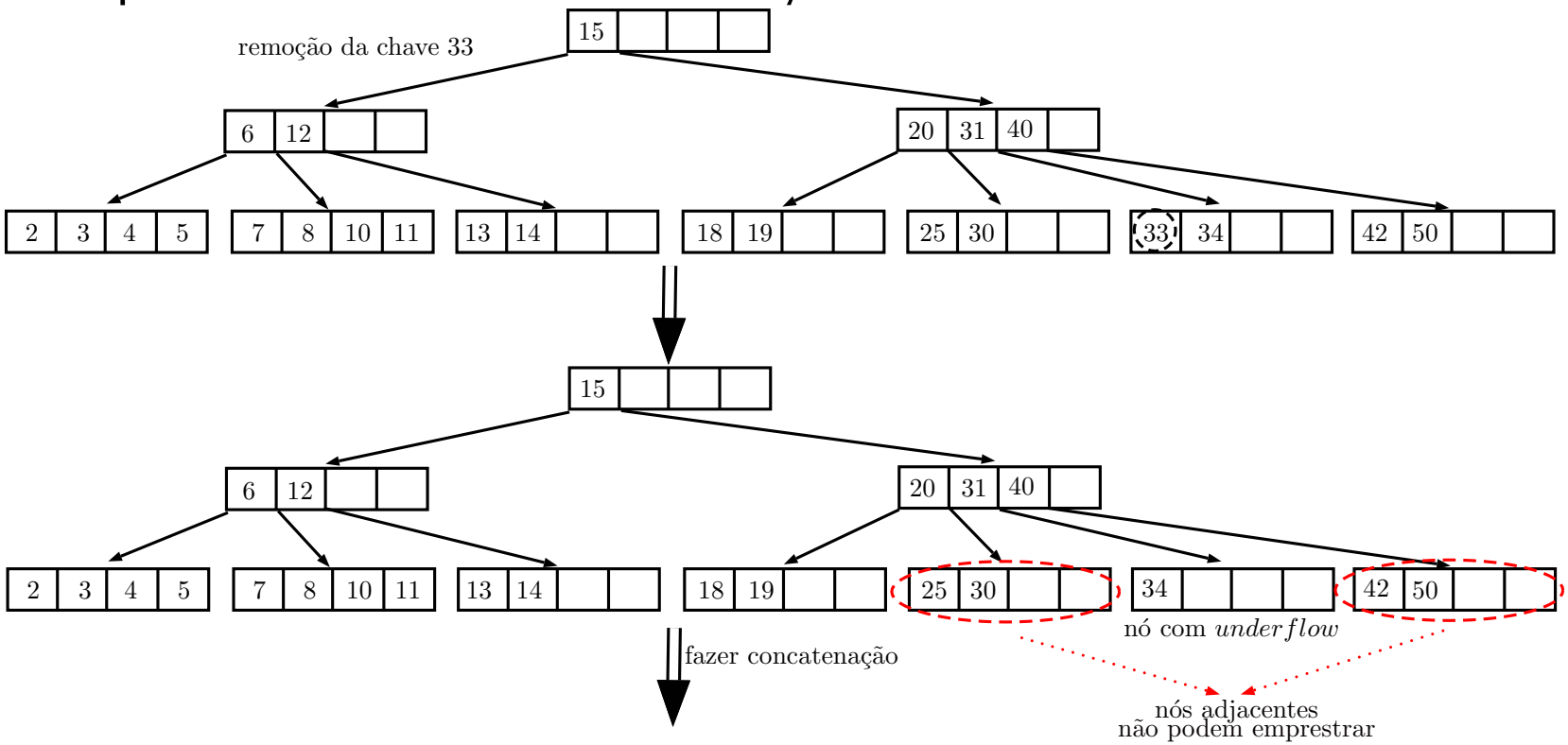
Árvore B - Remoção

Exemplo do Caso 3.1: Empréstimo/Redistribuição (cont.)



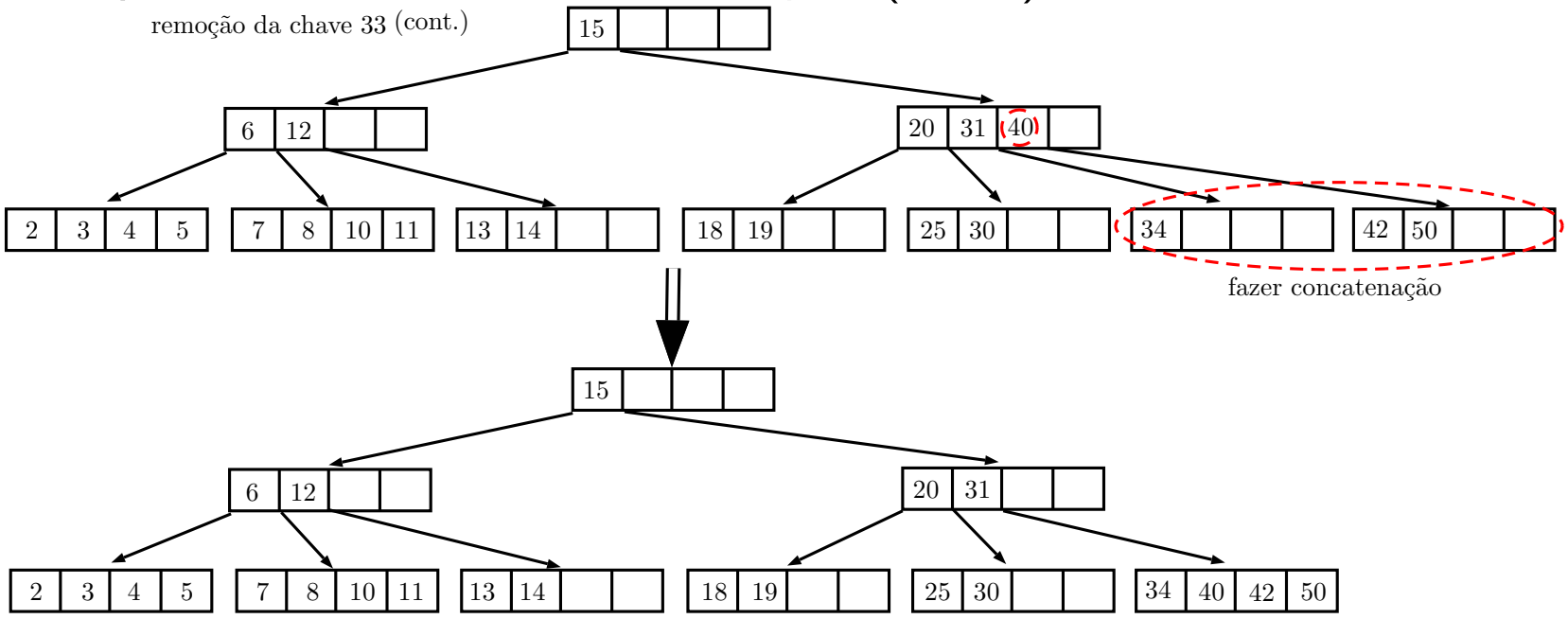
Árvore B - Remoção

Exemplo do Caso 3.2: Concatenação



Árvore B - Remoção

Exemplo do Caso 3.2: Concatenação (cont.)



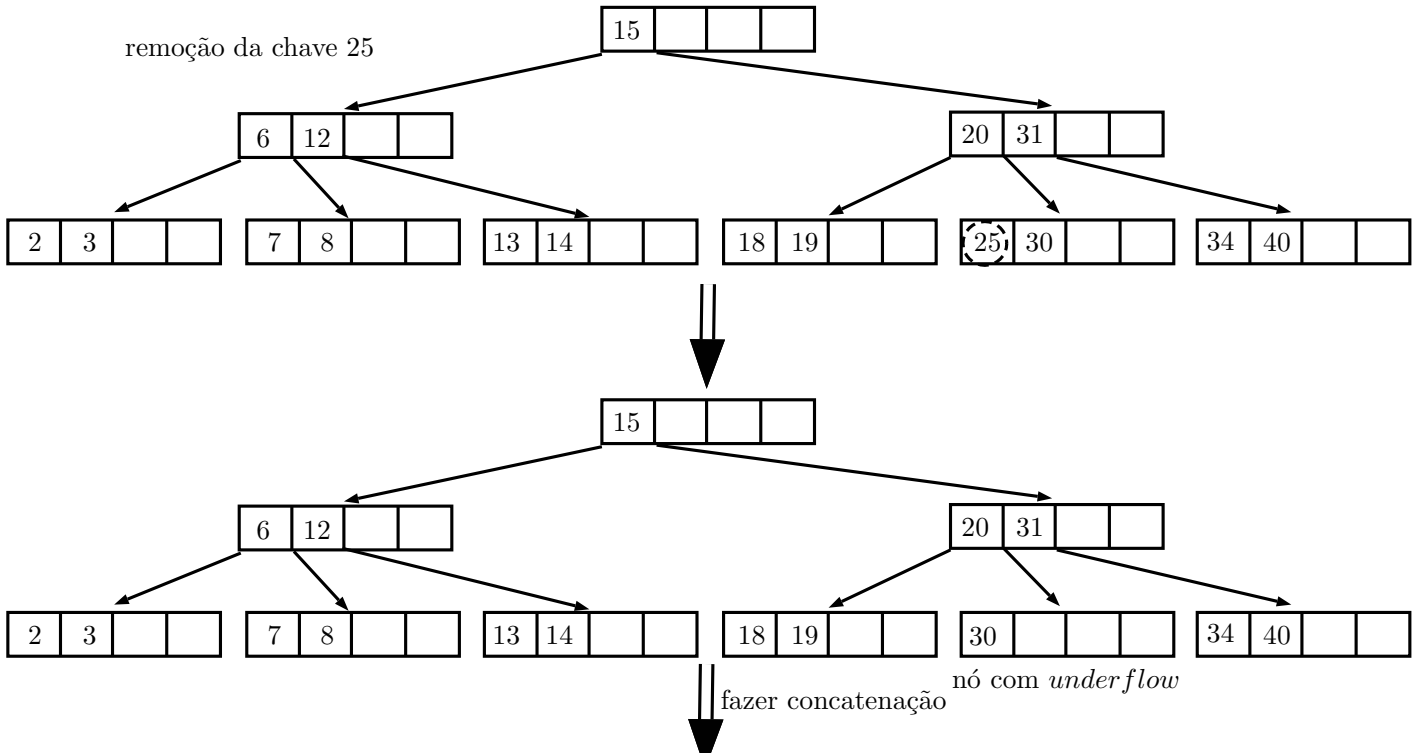
Árvore B - Remoção

Remoção - Caso 4

A raiz tem uma única chave, que é absorvida pela concatenação dos nós-filhos. Nessa situação, a raiz é eliminada e o nó resultante da concatenação dos filhos torna-se a nova raiz.

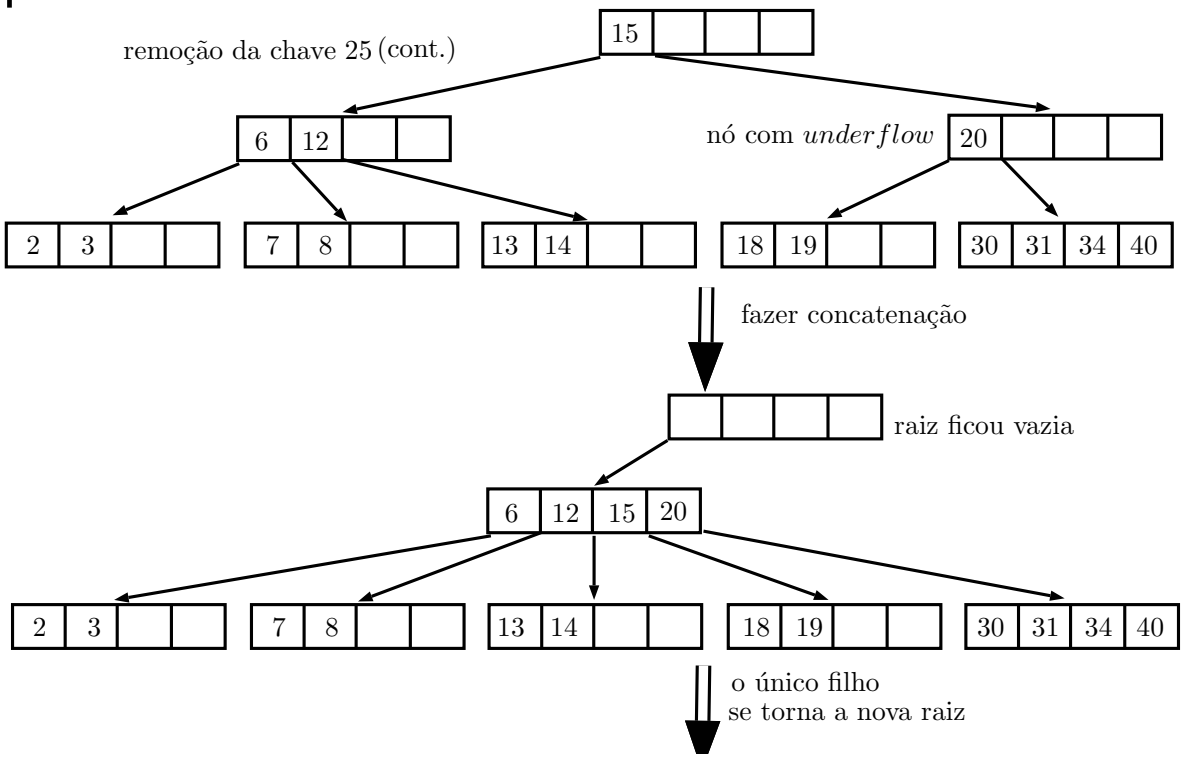
Árvore B - Remoção

Exemplo do Caso 4: a raiz tem uma única chave que é absorvida pelos filhos.



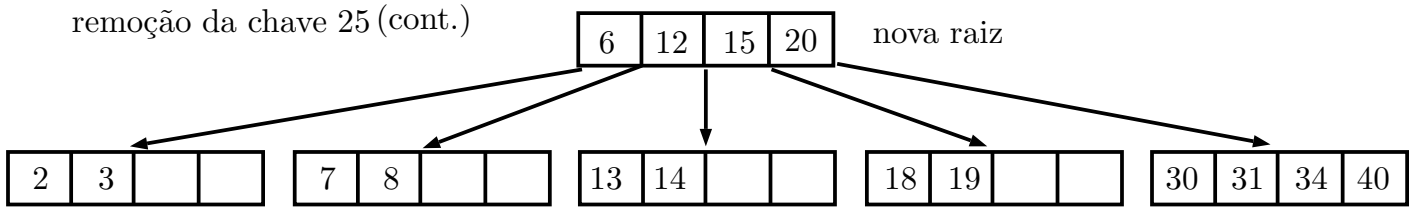
Árvore B - Remoção

Exemplo do Caso 4 (cont.): a raiz tem uma única chave que é absorvida pelos filhos.



Árvore B - Remoção

Exemplo do Caso 4 (cont.): a raiz tem uma única chave que é absorvida pelos filhos.



Árvore B - Observações

- A operação de redistribuição (empréstimo) de chaves não se propaga para os níveis superiores.
- A redistribuição pode ser usada na inserção subsituindo a operação de *split*, em que a chave que causou o *overflow* junto com outras chaves do mesmo nó podem ser movidas para um nó adjacente. Isso permite melhorar a taxa de utilização do espaço alocado para a árvore, pois evita ou adia a criação de novos nós.

Árvore B*

Árvore B*

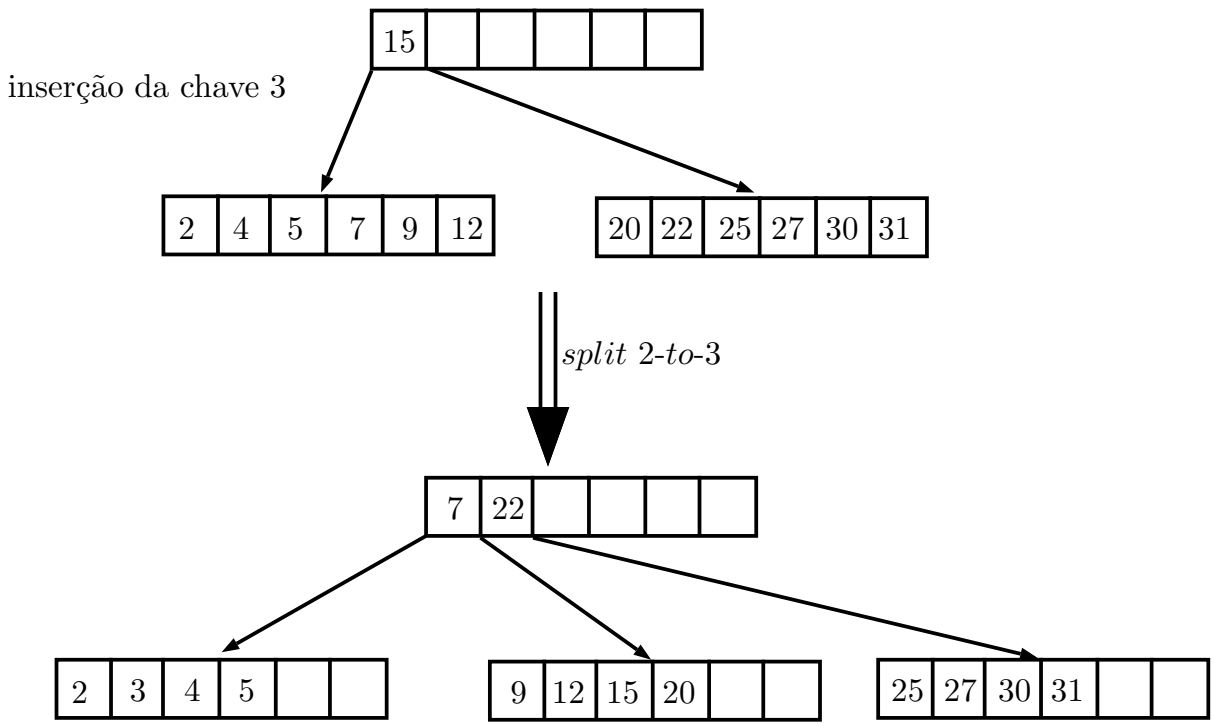
A árvore B* foi proposta por Knuth em 1973:

- possui as mesmas propriedades que uma árvore B
- cada nó possui pelo menos $\frac{2}{3}$ do número máximo de chaves
- usa a redistribuição de chaves durante a inserção para postergar a operação de *split*.

Árvore B*

Árvore B*

- o *split* é adiado até que 2 nós irmãos adjacentes estejam cheios, quando são quebrados em 3 nós (*split 2-to-3*).



Árvore B*

Árvore B*

O particionamento da raiz deve ser tratado separadamente, pois não tem irmão. Soluções possíveis:

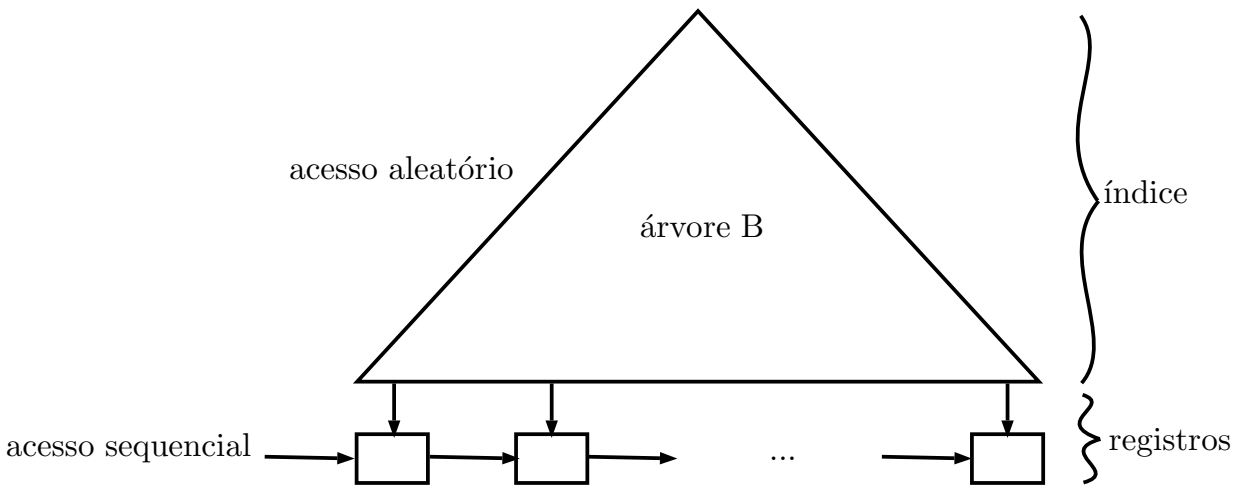
- fazer o *split* convencional quebrando a raiz em 2 nós.
- permitir que a raiz tenha mais chaves que os outros nós.

Árvore B+

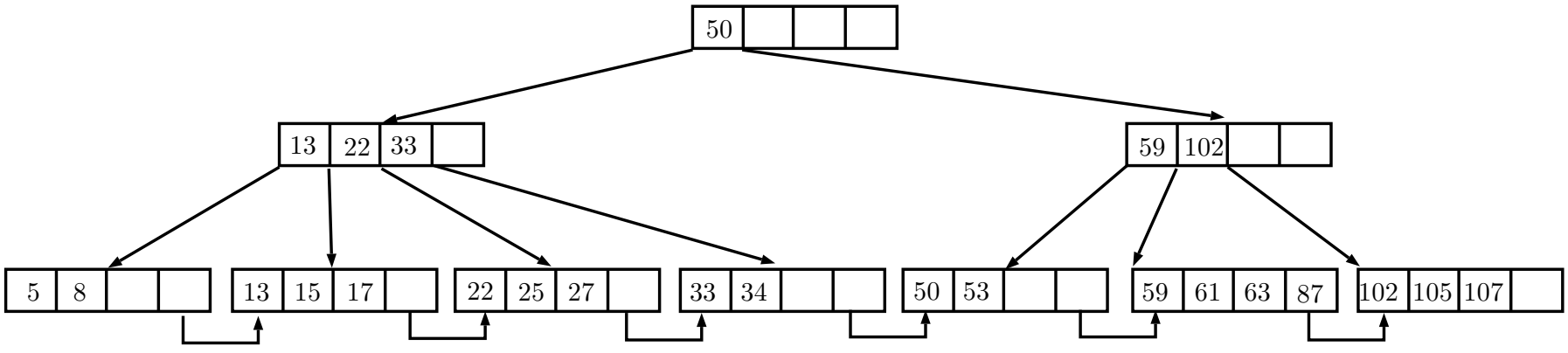
Árvore B+

A árvore B+ é uma variante da árvore B tal que:

- todas as chaves são mantidas em folhas, podendo então serem repetidas em nós internos
- os níveis acima das folhas são organizados na forma de uma árvore B
- as folhas são ligadas permitindo acesso sequencial às chaves.



Árvore B$^+$ - exemplo



Árvore B^+

Árvore B^+ - estrutura

A estrutura da árvore B^+ pode ser implementada:

- usando um mesmo tipo de nó para representar tanto folhas quanto nós internos ou usando tipo de nós diferentes para folhas e nós internos.
- usando o último ponteiro de subárvore dos próprios nós ou um ponteiro separado para fazer o encadeamento das folhas

Árvore B$^+$ - estrutura

Usando um único tipo de nó e o último ponteiro de subárvore para encadeamento das folhas:

```
/* Tipo representado um nó na árvore B+, sendo genérico suficiente
 * para representar ambos folhas e nós internos. A relação entre chaves
 * e ponteiros diferem em folhas e nós internos.
 * Na folha, o índice da chave é igual ao índice do ponteiro de dados
 * correspondente (de 0 a ordem -1). O último ponteiro aponta para a folha
 * seguinte (à direita) ou NULL no caso de ser a folha mais à direita.
 * Em um nó interno, o ponteiro i aponta para subárvore de nós com
 * chaves  menores que à chave i enquanto o ponteiro i+1 aponta para subárvore
 * de nós com chaves maiores ou iguais à chave i.
 */
typedef struct nodeBMais {
    void ** ponteiro;        // vetor de ponteiros
    int * chave;             // vetor de chaves
    struct nodeBMais * pai; // ponteiro para o nó-pai
    int eh_folha;            // booleano, verdadeiro quando nó é folha
    int numChaves;           // número de chaves no nó
} noBMais;
```

Árvore B$^+$

Árvore B$^+$

Não há uma definição única de árvore B$^+$, existindo uma variedade de implementações. Apresentamos aqui uma das variantes.

Árvore B+

Árvore B+

- a complexidade de tempo de inserção e busca é a mesma da árvore B
- a busca do próximo registro tem complexidade $O(1)$ devido à possibilidade de acesso sequencial
- não há necessidade de manter ponteiros para registros de dados (dados satélites) em nós internos
- existe 2 opções de estratégias básicas a serem consideradas na remoção de chaves:
  - remover sempre nas folhas mantendo as chaves dos nós internos; ou
  - remover tanto as chaves nas folhas quantos dos nós internos

Árvore B+

Inserção

Existem 3 casos na inserção, dependendo se o nó-folha e seu pai estão cheios ou não.

Árvore B+

Inserção

Caso 1: o nó-folha onde a chave deve ser inserida não está completo. Então, a chave é inserida no nó-folha na posição apropriada.

Árvore B^+

Inserção

Caso 2: o nó-folha onde a chave deve ser inserida está completo, e o nó-pai (de índice) não está completo.

1. Fazer *split* do nó-folha
2. Colocar a chave mediana no nó-pai (de índice) na posição ordenada
3. nó-folha esquerdo fica com chaves $<$ chave mediana
4. nó-folha direito fica com chaves $>=$ chave mediana

Árvore B+

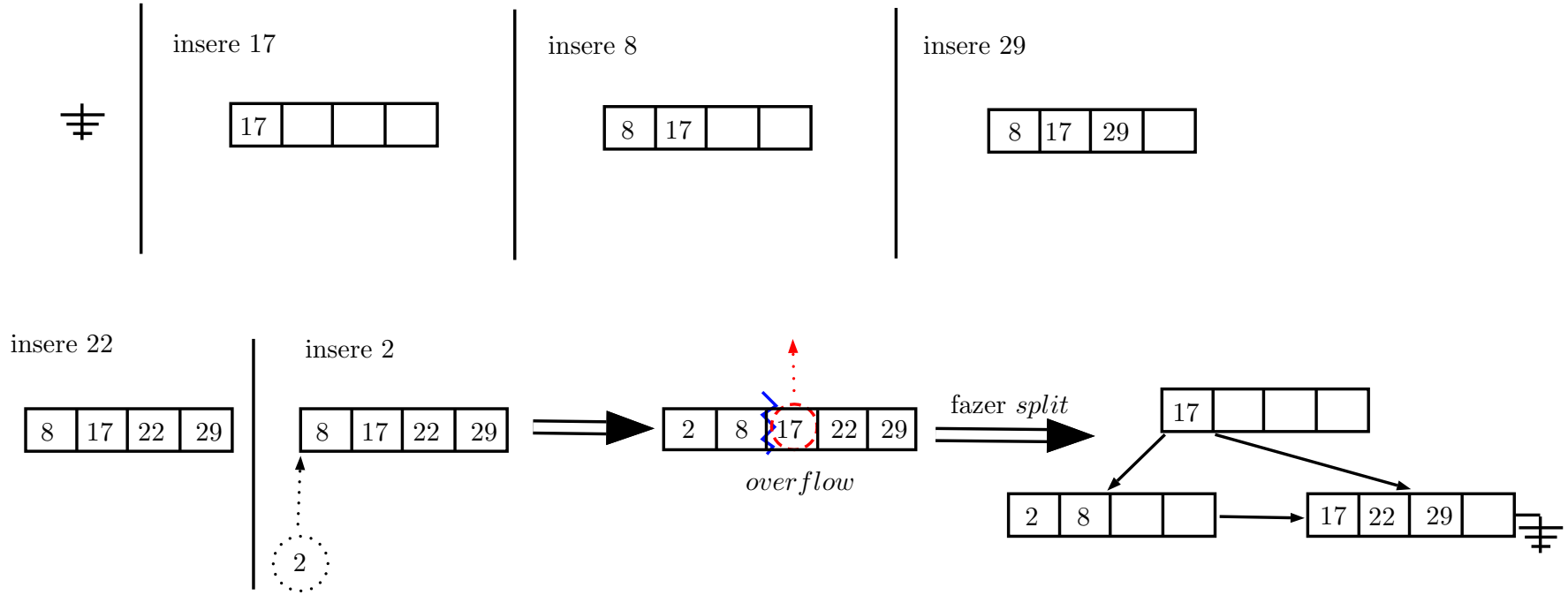
Inserção

Caso 3: o nó-folha onde a chave deve ser inserida e o nó-pai (de índice) estão completos.

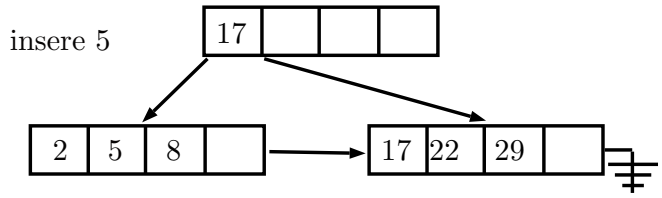
1. Fazer *split* do nó-folha
2. chaves $<$ a chave mediana ficam no nó-folha esquerdo
3. chaves $>=$ a chave mediana ficam no nó-folha direito
4. Fazer *split* do nó-índice (pai)
5. chaves $<$ chave mediana ficam no nó-índice esquerdo
6. chaves $>$ chave mediana ficam no nó-índice direito
7. chave mediana é inserida no nó-índice acima.

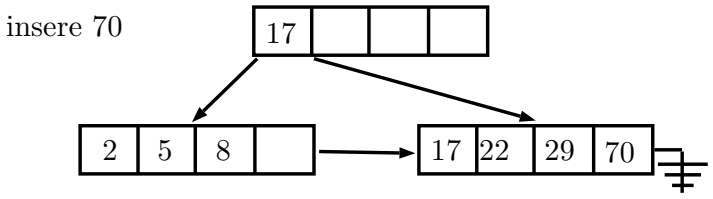
O *split* de nós-índices é repetido recursivamente se necessário. Quando feito na raiz, a altura da árvore aumenta.

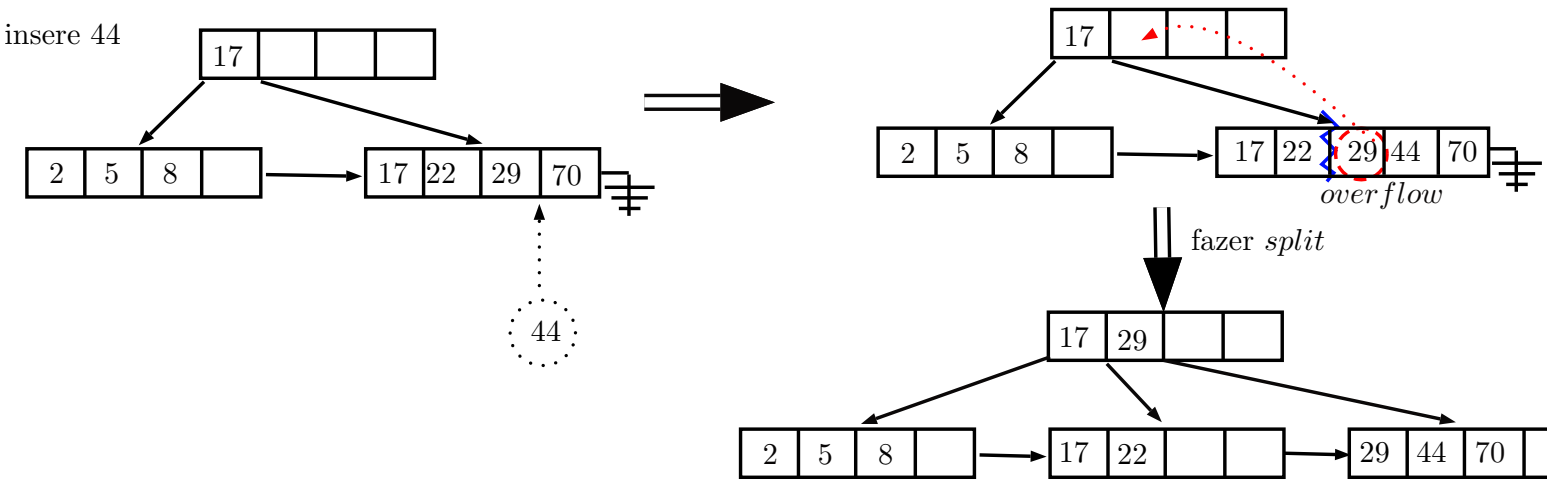
Árvore B$^+$ - Inserção: exemplo



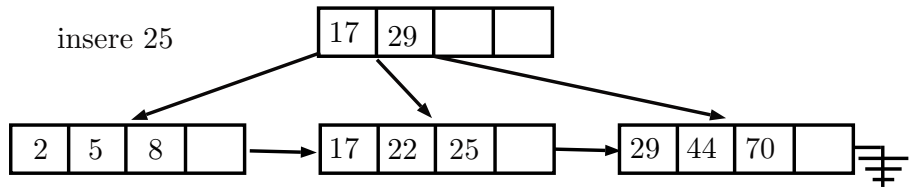
Árvore B$^+$ - Inserção: exemplo

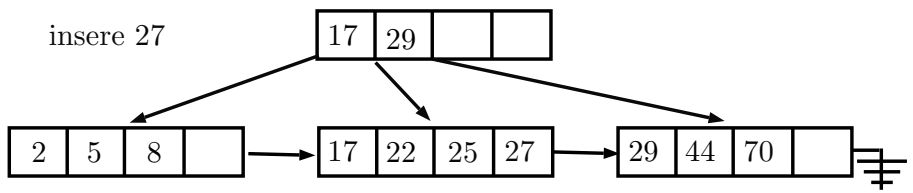




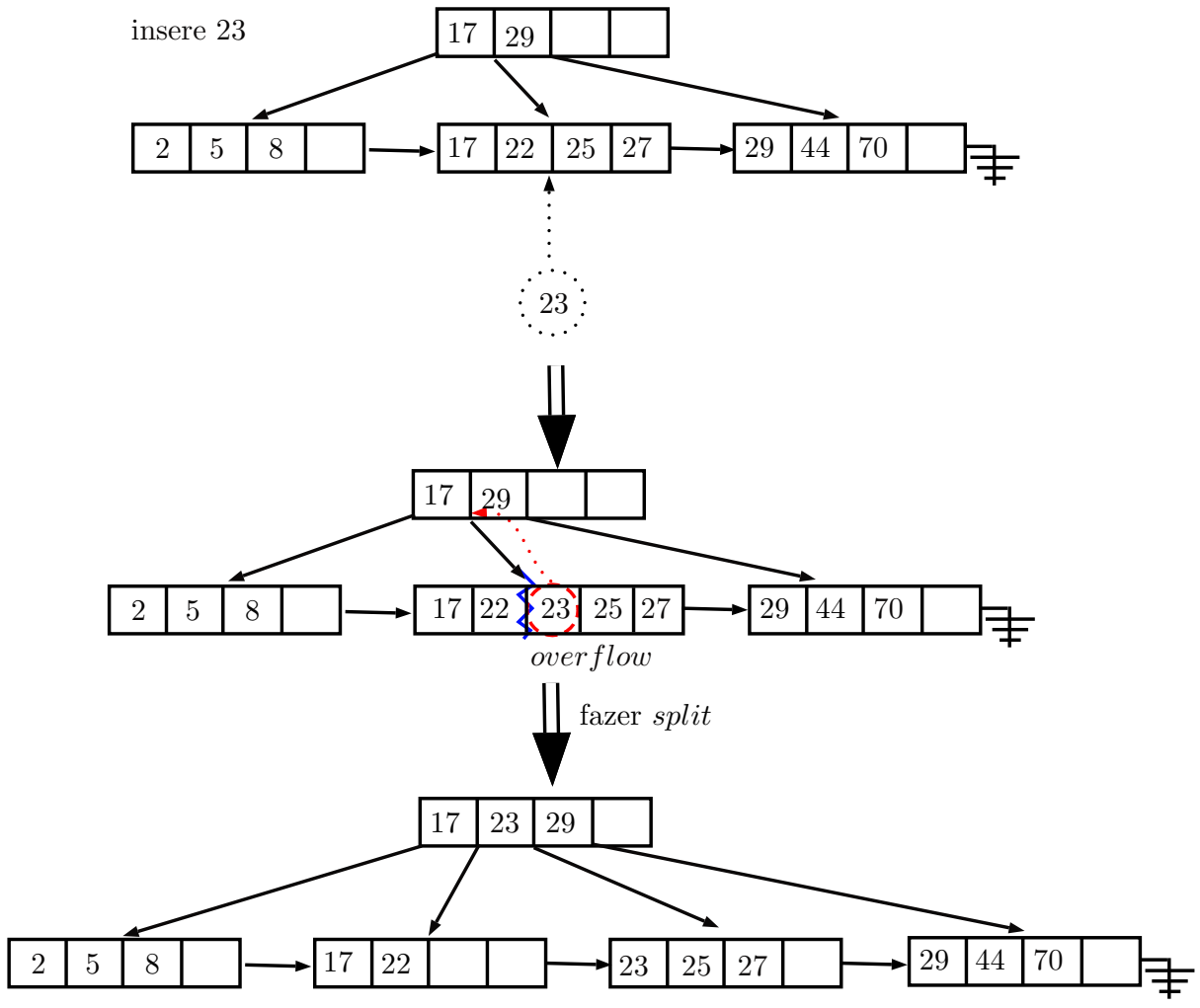


Árvore B$^+$ - Inserção: exemplo

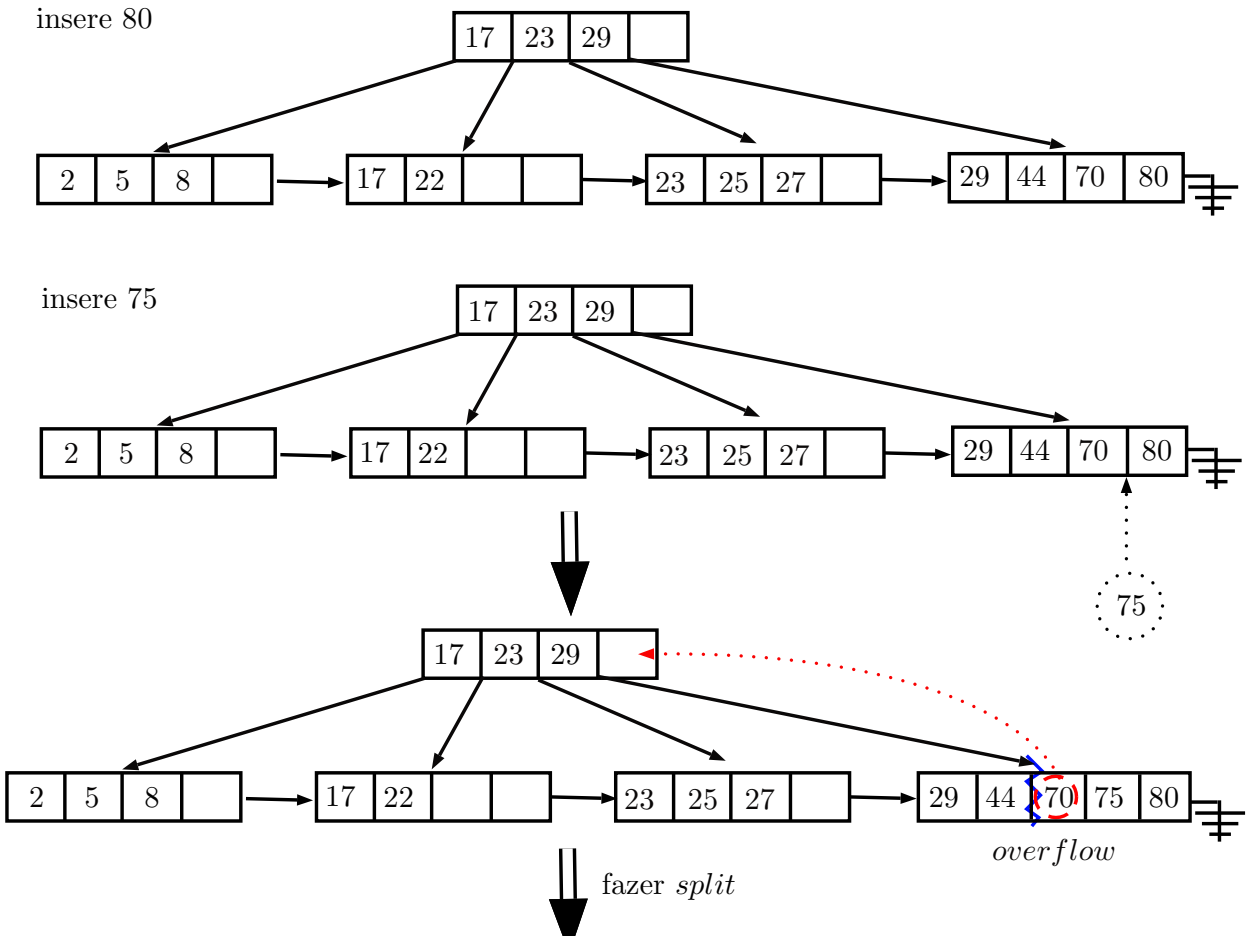




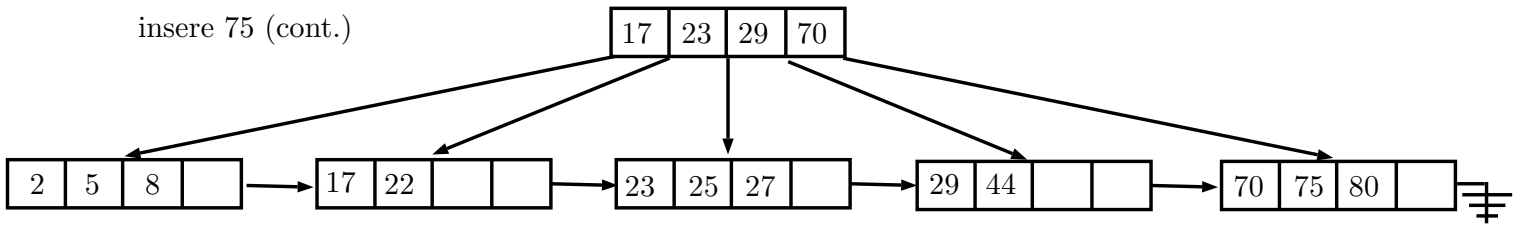
Árvore B$^+$ - Inserção: exemplo

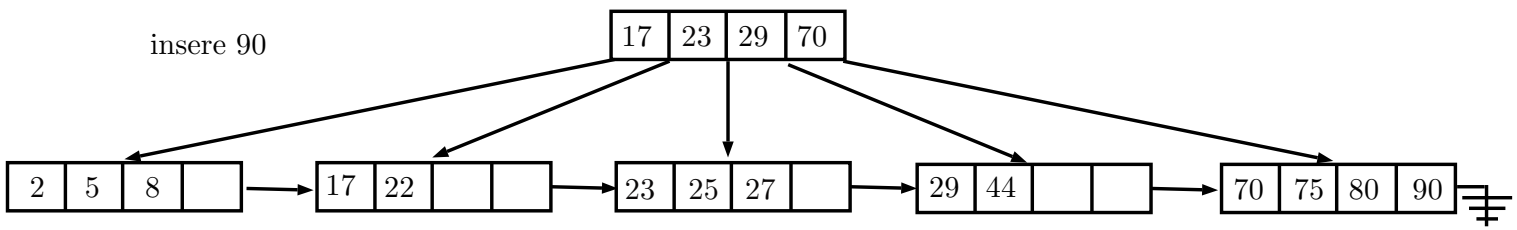


Árvore B$^+$ - Inserção: exemplo

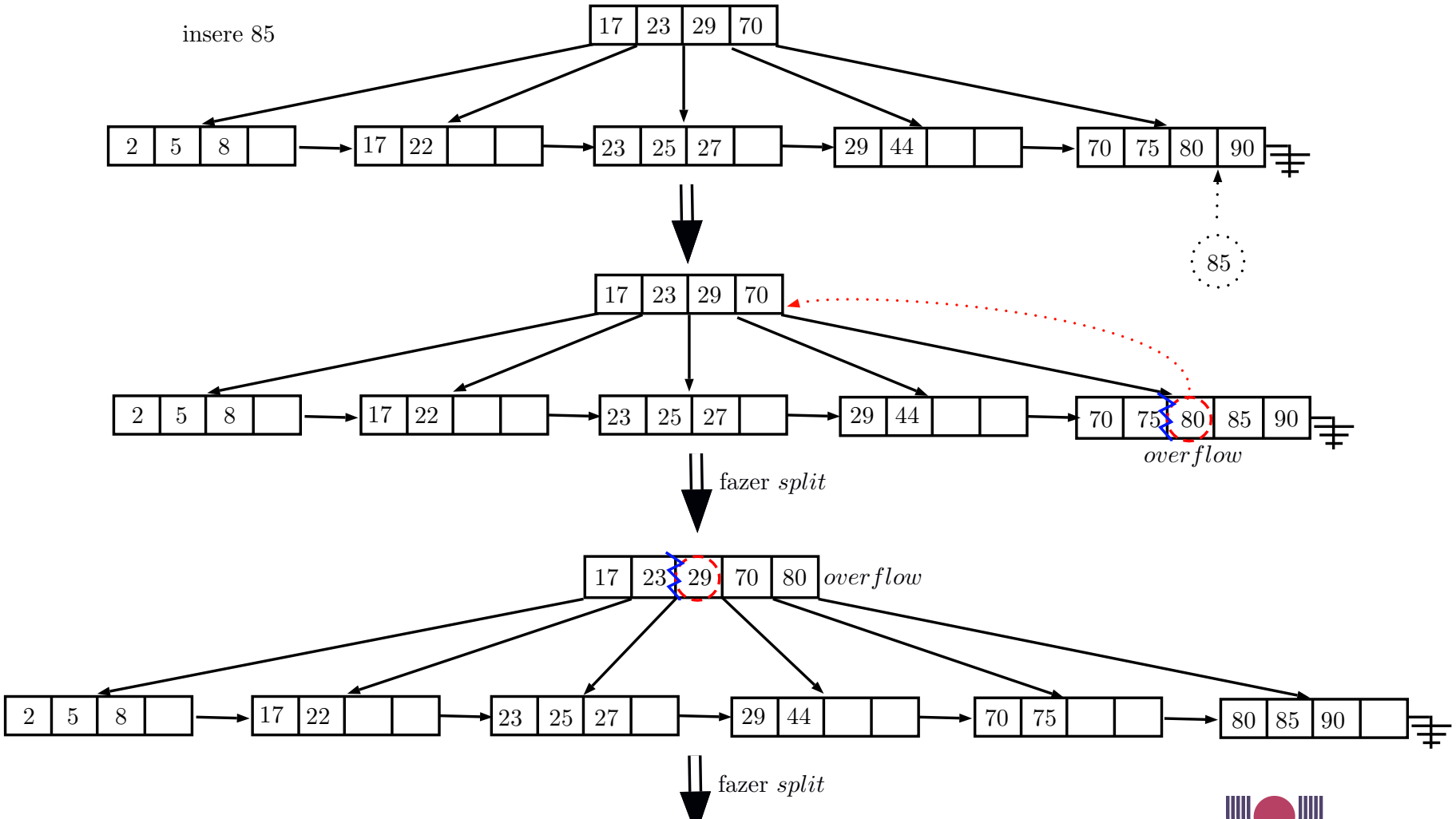


Árvore B+ - Inserção: exemplo

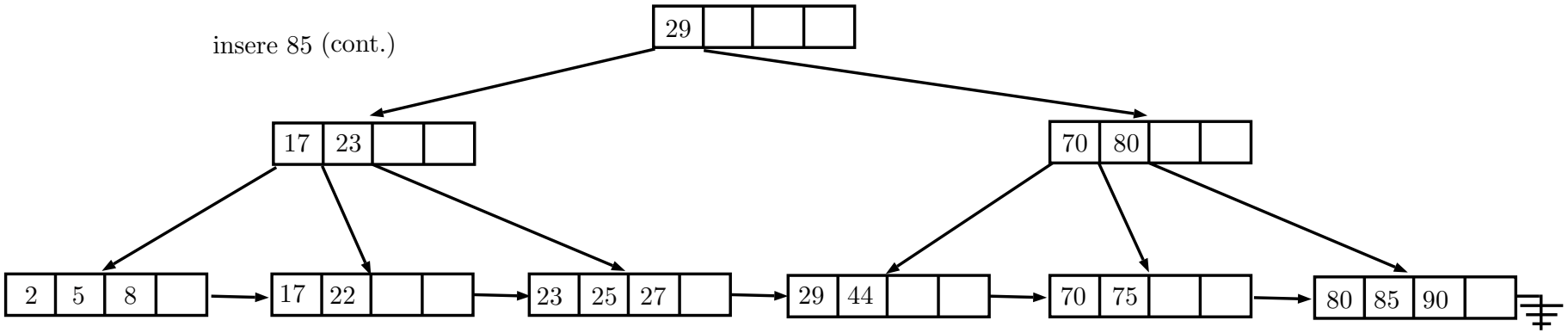




Árvore B$^+$ - Inserção: exemplo



Árvore B$^+$ - Inserção: exemplo



Árvore B^+

Busca

A **Busca** em uma árvore B^+ **deve sempre ir até a folha, mesmo que a chave seja encontrada em um nó interno**. Isso porque os nós internos servem apenas para direcionar o processo busca, reduzindo o universo da pesquisa. Mas em geral eles não contêm o endereço dos dados-satélites, estando essa informação somente nos nós-folhas. Além disso, quando a estratégia de remoção adotada é de remover somente as chaves nas folhas e não remover nos índices, encontrar uma chave no índice não garante que de fato exista dados-satélites associados à chave.

Árvore B$^+$

Remoção

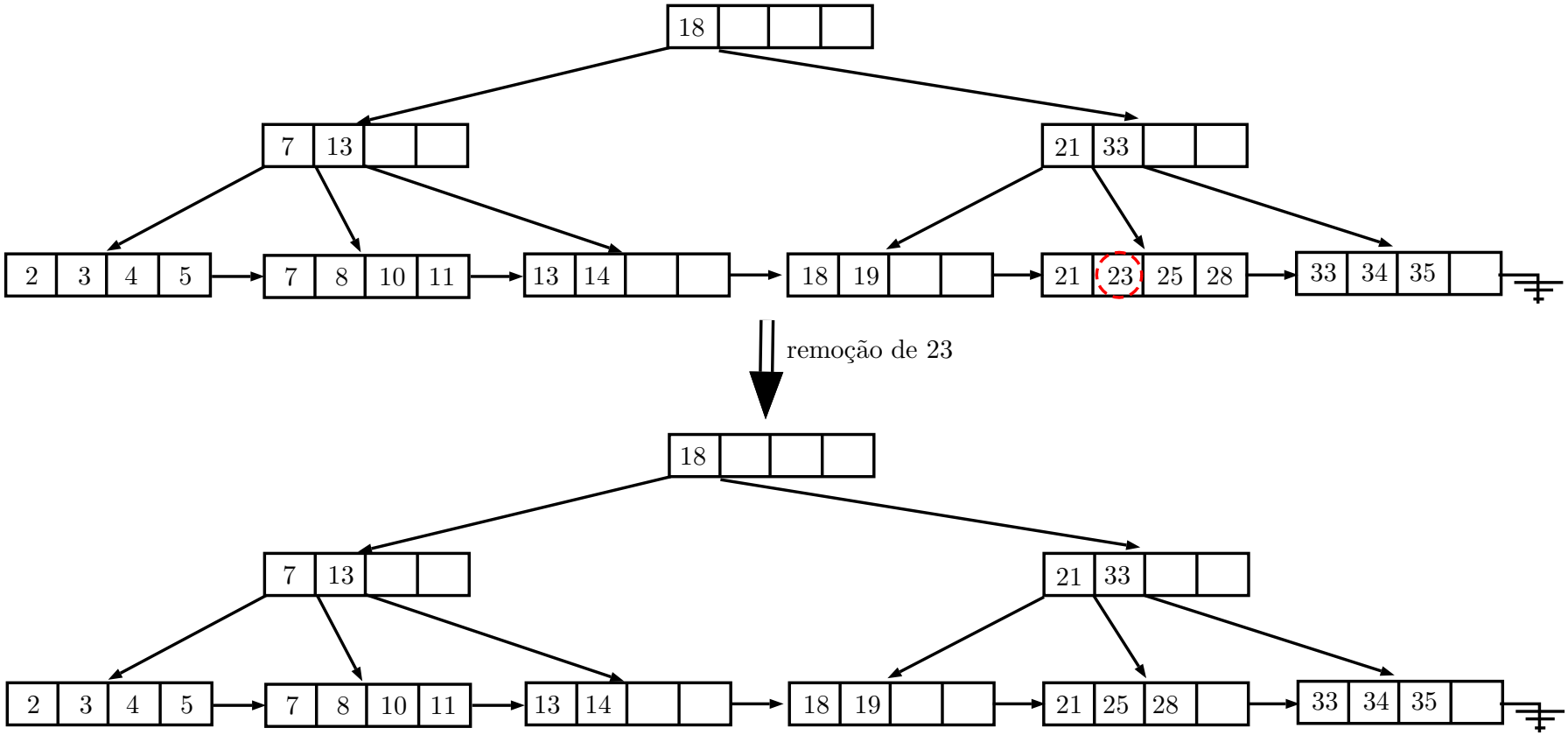
Adotaremos a estratégia de remoção em que as chaves são removidas dos nós-folhas e também de nós-índices. Nessa condição, a remoção em árvore B$^+$ tem 3 casos.

Árvore B+

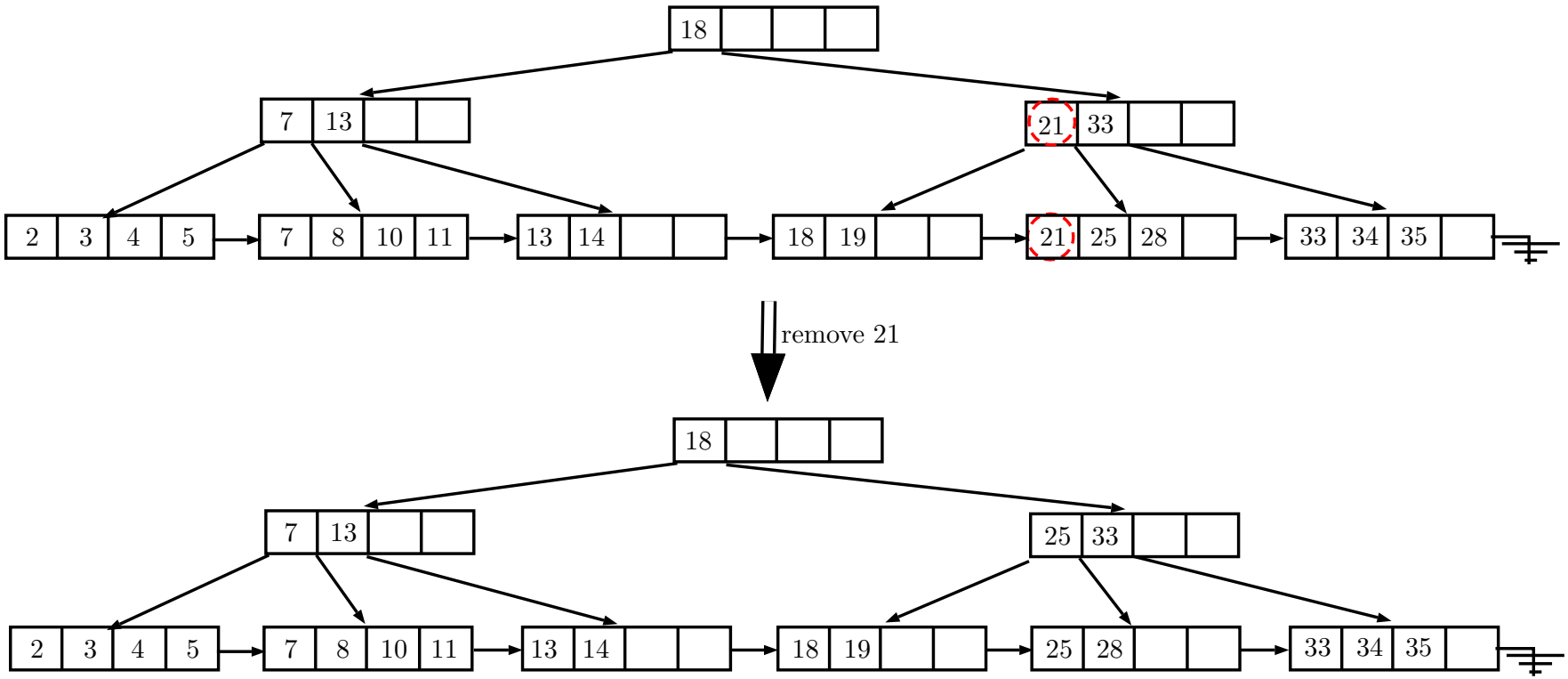
Remoção

Caso 1: O nó-folha tem mais chaves que o mínimo. Nesse caso, a chave é simplesmente e nó-folha reordenado. Caso a chave ocorra no nó-índice, atualiza-se o nó-índice com a próxima chave.

Árvore B$^+$ - Remoção: exemplo (caso 1)



Árvore B+ - Remoção: outro exemplo (caso 1)

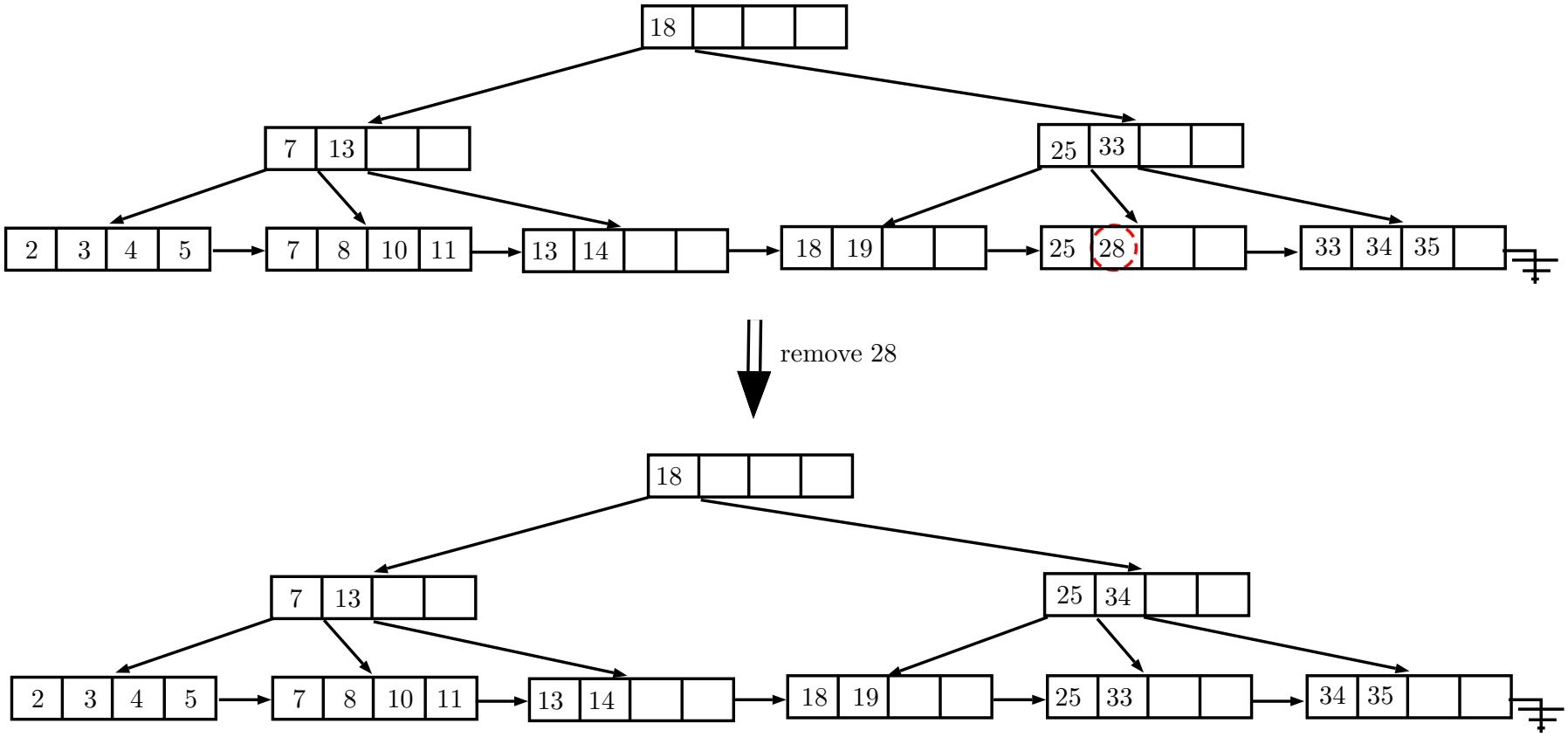


Árvore B+

Remoção

Caso 2: O nó-folha tem o mínimo de chaves, há nó-irmão com mais chaves que o mínimo. Nesse caso, a chave é removida e o nó-folha é combinado com o nó-irmão. O nó-índice é alterado para refletir a mudança.

Árvore B$^+$ - Remoção: exemplo (caso 2)



Árvore B+

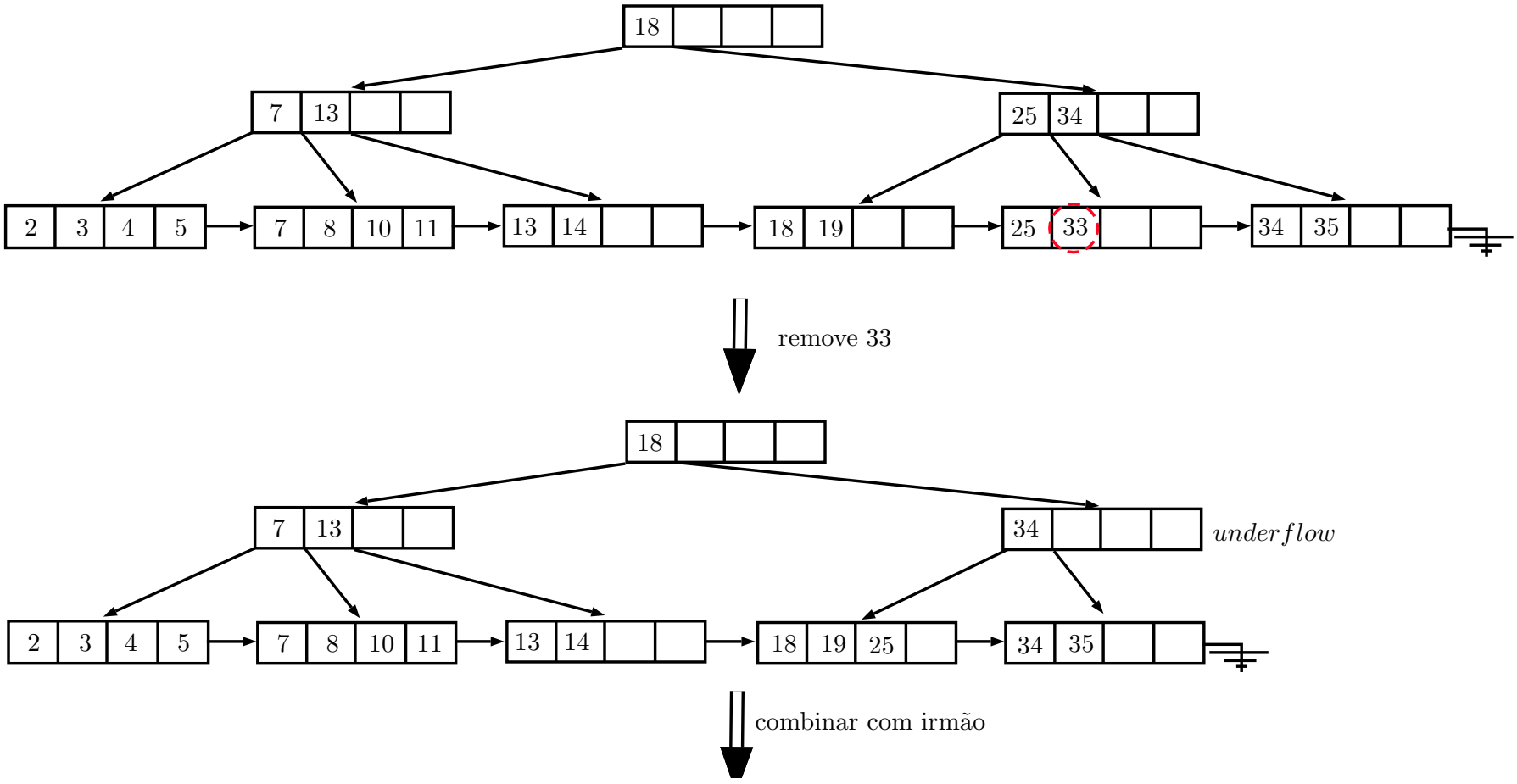
Remoção

Caso 3: Tanto o nó-folha, nós-irmãos e o nó-índice têm o mínimo de chaves

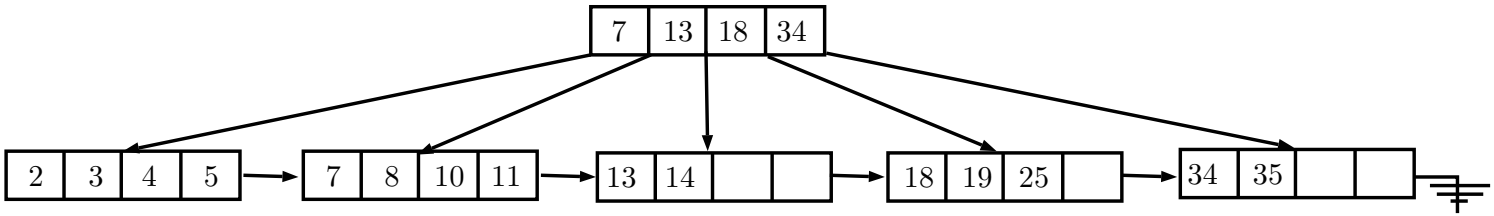
1. Remover a chave no nó-folha e combiná-lo com seu irmão
2. Ajustar o nó-índice para refletir essa alteração
3. Combinar o nó-índice com seu irmão

A combinação de nó-índice com seu irmão é repetida recursivamente nos níveis superiores se necessário. A altura da árvore diminui quando esse processo chega à raiz.

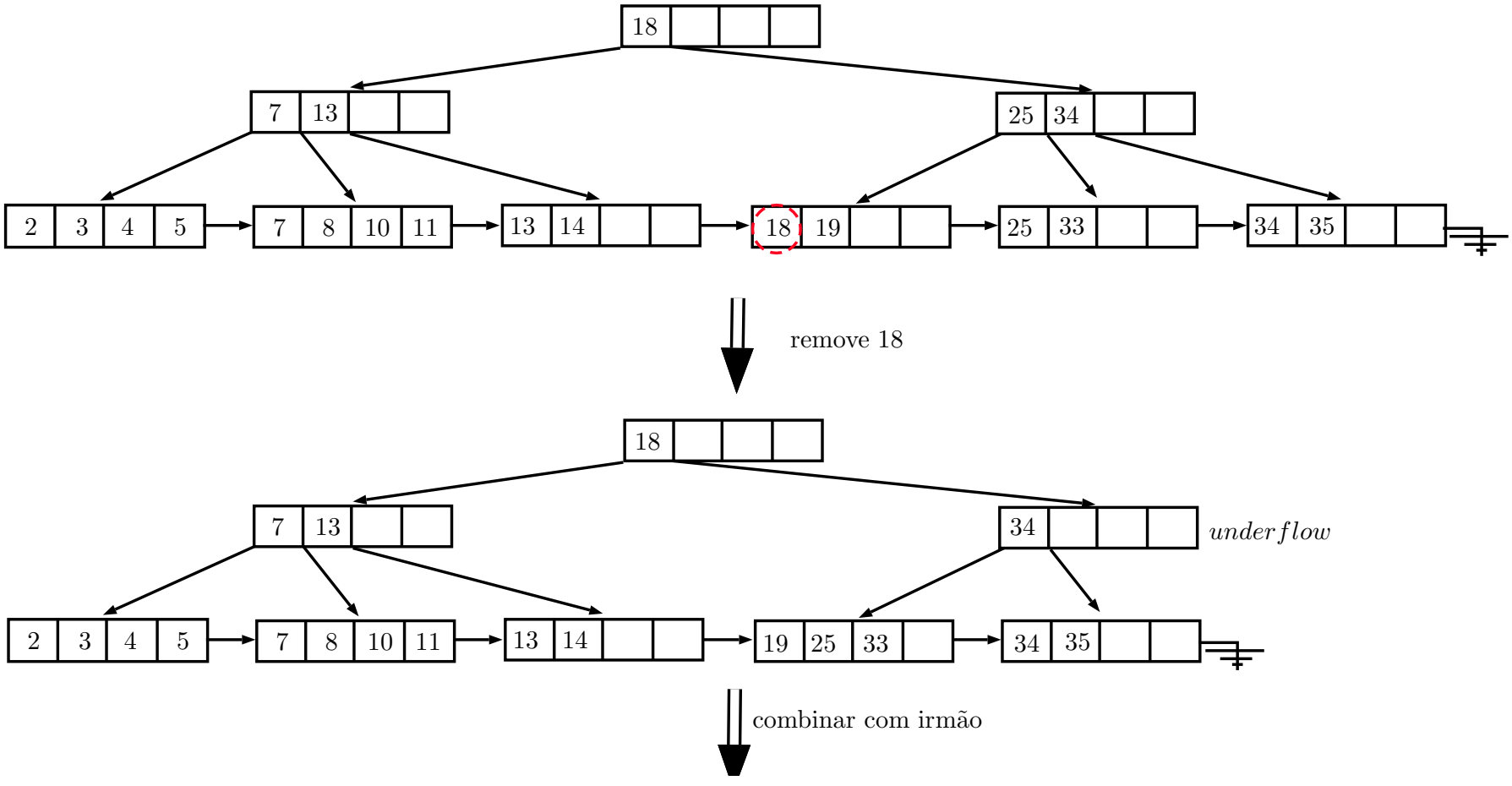
Árvore B$^+$ - Remoção: exemplo (caso 3)



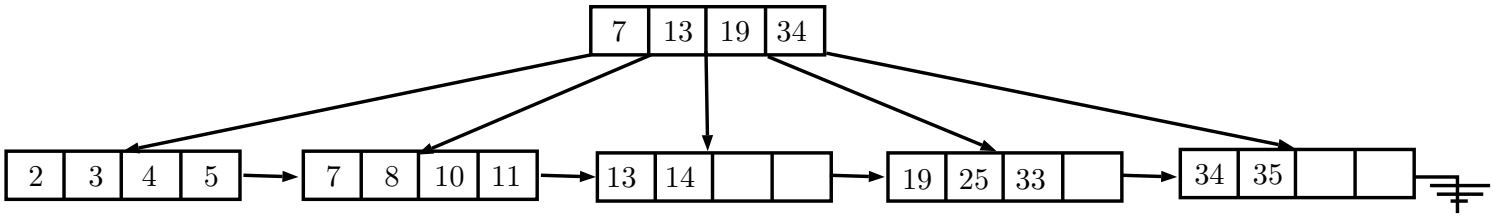
Árvore B$^+$ - Remoção: exemplo (caso 3) - cont.



Árvore B$^+$ - Remoção: outro exemplo (caso 3)



Árvore B$^+$ - Remoção: outro exemplo (caso 3) - cont.



Bibliografia I

[Folk 1992] Folk, Michael; Zoellick, Bill.
File Structures, Ed. Addison-Wesley, 2a edição, 1992.

[Knuth 1998] Knuth, Donald (1998).
The Art of Computer Programming. Sorting and Searching, vol. 3, Ed. Addison-Wesley, 2a edição, 1998.

[Cormen 1997] Cormen,T.; Leiserson, C.; Rivest, R.
Introduction to Algorithms. McGrawHill, New York, 1997.

[Comer 1997] Comer,D.
The Ubiquitous B-Tree. Computing Surveys, Vol 11, No 2, Junho, 1979, New York.