Árvore 2-3

Rômulo César Silva

Unioeste

Julho de 2016

unioeste
Universidade Estadual do Oeste do Paraná

Sumário

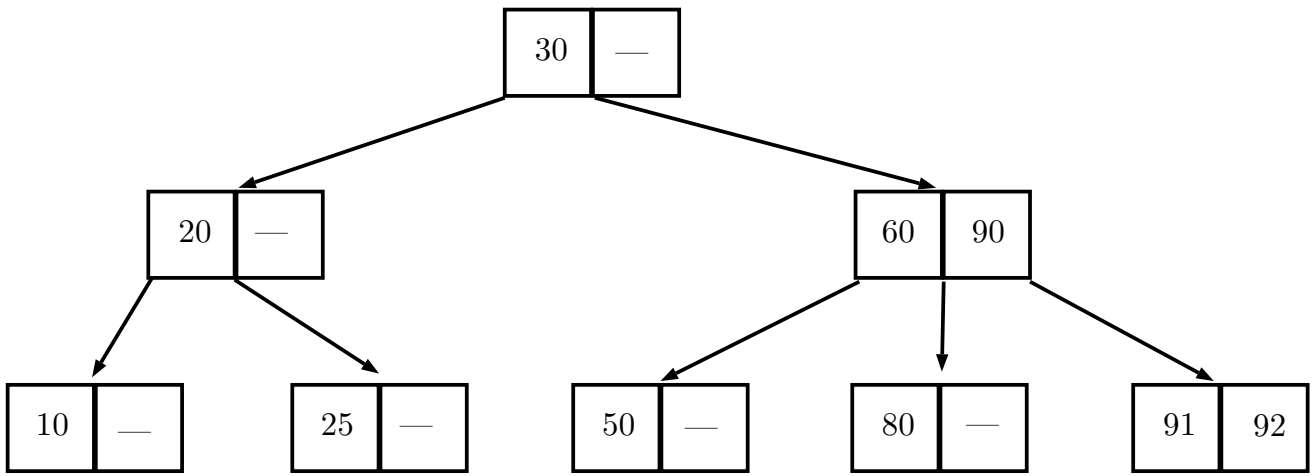
Árvore 2-3

Definição

Uma **árvore 2-3** é uma árvore balanceada com as seguintes propriedades:

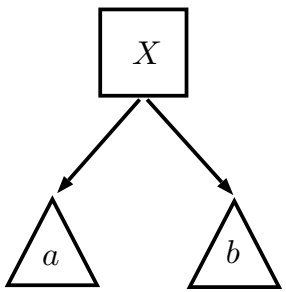
- os elementos estão ordenados da esquerda (mínimo) para a direita (máximo).
- todo caminho na árvore da raiz às folhas tem o mesmo comprimento, isto é, todas as folhas estão no mesmo nível.
- nós internos têm 2 ou 3 subárvores

Árvore 2-3: exemplo



Árvore 2-3

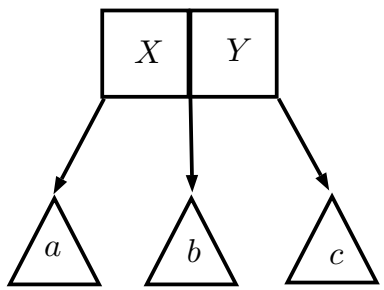
Representação de um nó com 2 subárvores:



- todo valor v na subárvore a deve ser $\leq X$
- todo valor v na subárvore b deve ser $\geq X$

Árvore 2-3

Representação de um nó com 3 subárvores:



- todo valor v na subárvore a deve ser $\leq X$
- todo valor v na subárvore b deve ser $\geq X$ e $\leq Y$
- todo valor v na subárvore c deve ser $\geq Y$

Árvore 2-3

Considerando uma árvore 2-3 de n nodos, sua altura está entre $\log_2 n$ e $\log_3 n$. Assim, a operação de busca tem complexidade de tempo $O(lg\ n)$.

Balanceamento em árvore 2-3

Balanceamento em árvore 2-3

O **balanceamento** deve ser feito durante as operações de inserção e remoção de tal maneira que todas as folhas estejam no mesmo nível e as propriedades de ordenação da árvore 2-3 sejam mantidas.

Árvore 2-3 - estrutura

Estrutura:

```
struct no23 {
     int chave_esq;       // chave esquerda
     int chave_dir;       // chave direita
     struct no23 * esq;   // subárvore esquerda
     struct no23 * meio;  // subárvore do meio
     struct no23 * dir;   // subárvore direita
     int n;               // número de chaves no nó
  };

 typedef struct no23* arvore23; //árvore é um ponteiro
                                //  para um nó
```

Árvore 2-3: busca

```
// Retorno: ponteiro para o nó contendo a chave ou
//          NULL caso a chave não pertença à àrvore
arvore23 busca(arvore23 r, int chave){
  if(vazia(r))
     return NULL;
  if(r->chave_esq == chave)
      return r;
  if(r->n == 2 && r->chave_dir == chave)
      return r;
  if(chave < r->chave_esq)
     return busca(r->esq, chave);
  else if(r->n == 1)
         return busca(r->meio, chave);
  else if(chave < r->chave_dir)
         return busca(r->meio, chave);
  else return busca(r->dir, chave);
}
```

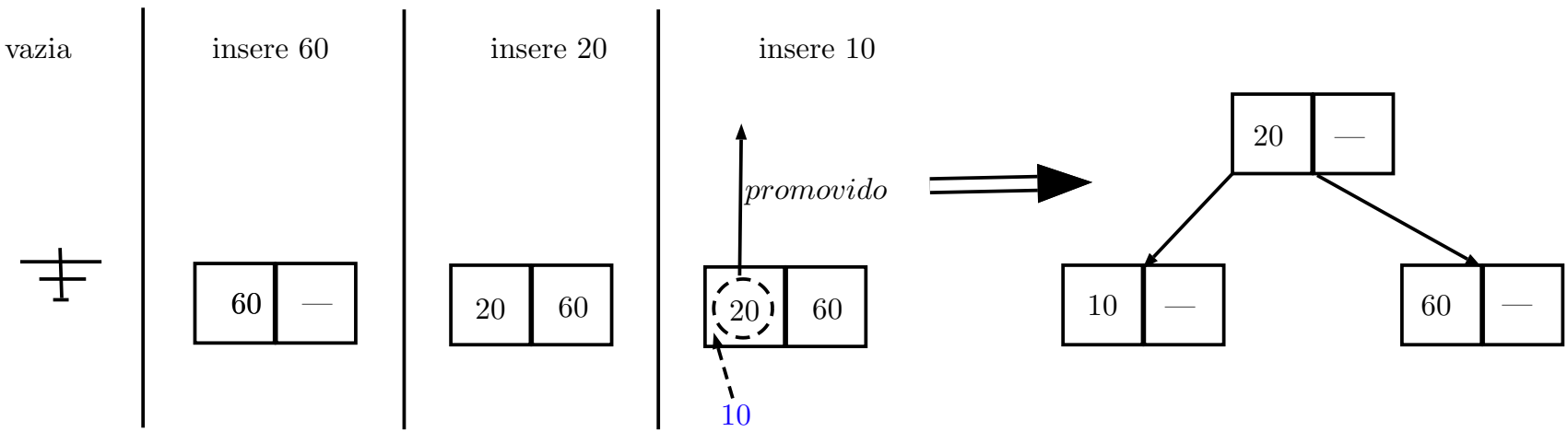
Árvore 2-3: inserção

Inserção

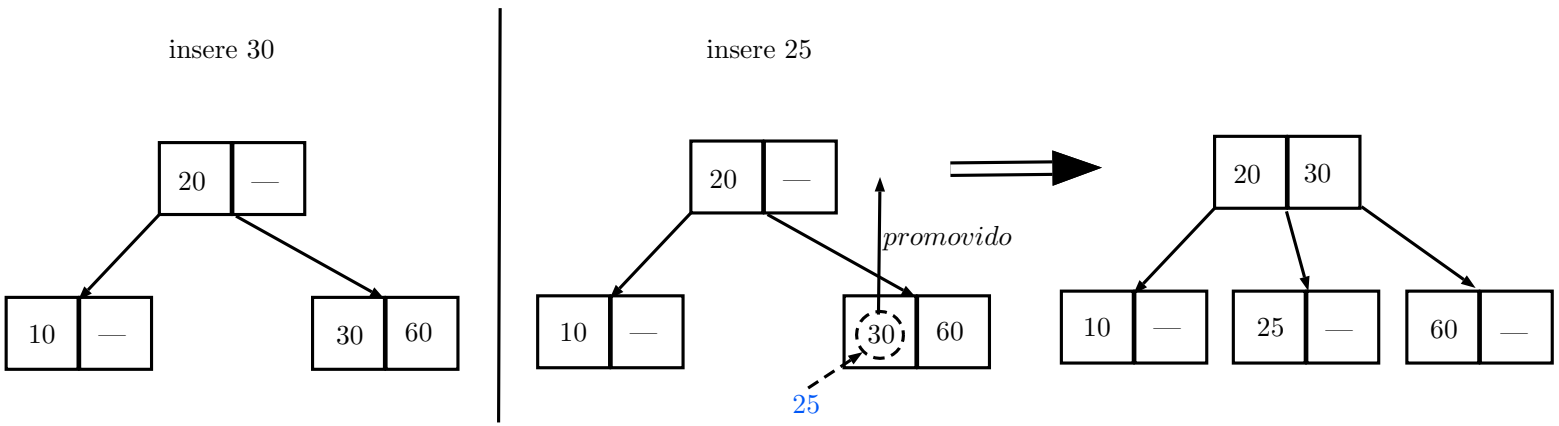
A inserção em uma árvore 2-3 tem os seguintes casos:

1. árvore é vazia: cria-se um nó com a chave sendo inserida
2. árvore não vazia: desce-se até a folha que deveria conter a chave
   - folha tem só uma chave: insira a chave na folha
   - folha tem 2 chaves: fazer *split* do nó tal que o valor mediano é "promovido" para o nó pai com o menor e o maior valor adicionados como filhos do pai. O processo de *split* é repetido recursivamente nos níveis superiores se necessário.

Árvore 2-3: Inserção - exemplo

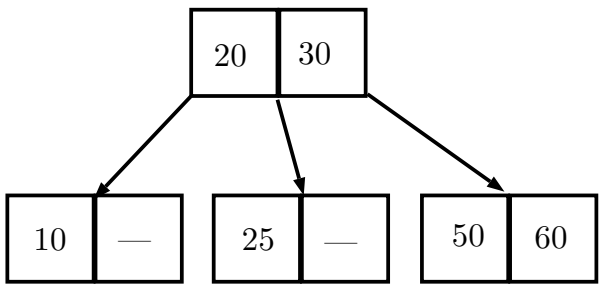


Árvore 2-3: Inserção - exemplo

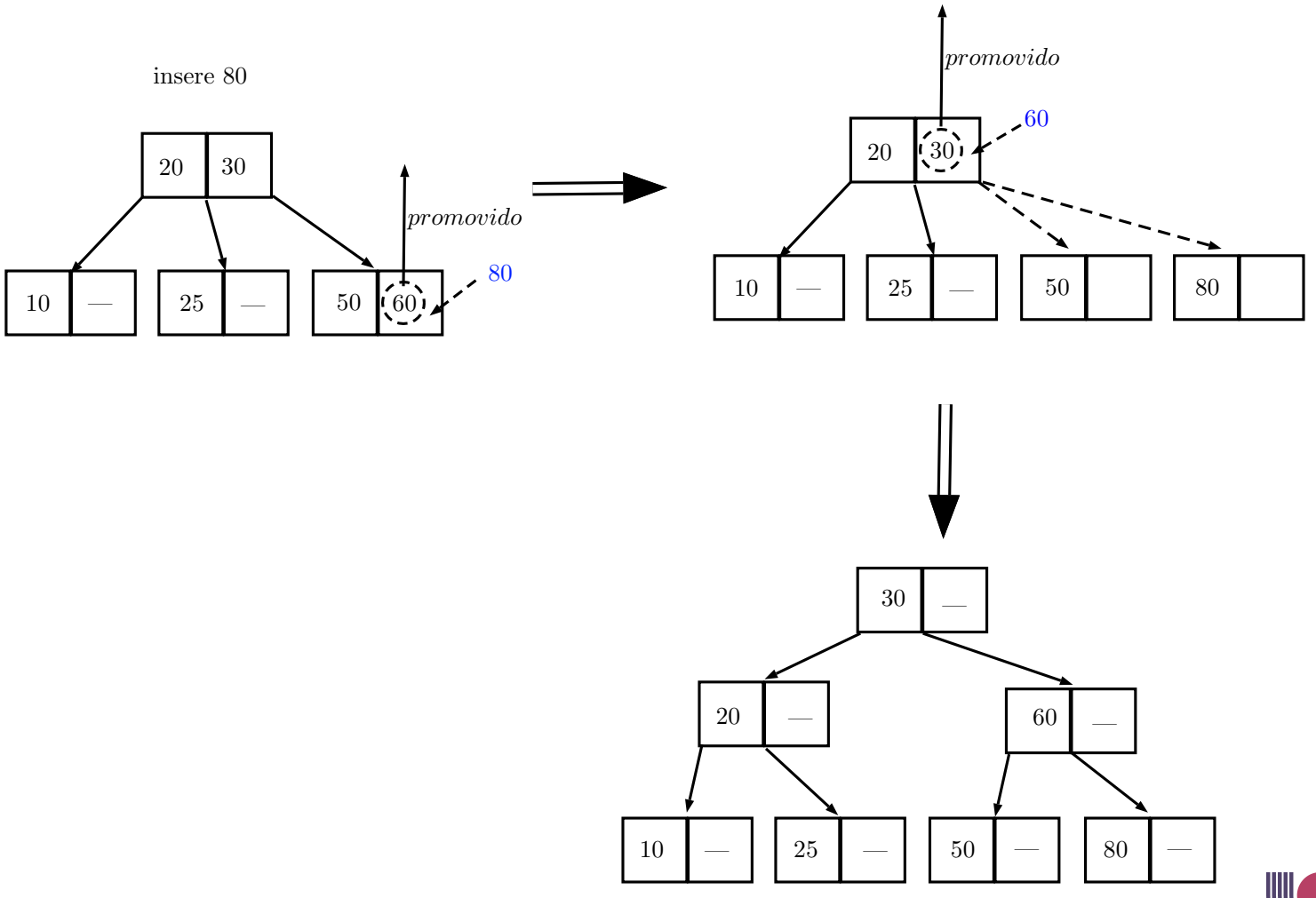


Árvore 2-3: Inserção - exemplo

insere 50



Árvore 2-3: Inserção - exemplo



Árvore 2-3: inserção

A implementação da árvore 2-3 aqui apresentada tem várias funções auxiliares:

- `arvore23 inserir_aux(arvore23 r, int chave, int* chave_promovida)`: insere a chave na árvore 2-3 retornando o nó gerado na operação de split e a chave a ser promovida se for o caso.
- `arvore23 split(arvore23 p, int chave, arvore23 subarvore,int *chave_promovida)`: quebra o nó com 2 chaves em 2 nós com 1 chave cada e informa a chave a ser promovida para o pai.

Árvore 2-3: inserção

funções auxiliares:

- `void adicionaChave(arvore23 r, int chave, arvore23 p)`: adiciona a chave em um nó que tem só 1 chave.
- `int eh_folha(arvore23 p)`: verifica se um nó é folha
- `arvore23 criaNo23(int chave_esq, int chave_dir, arvore23 esq, arvore23 meio, arvore23 dir, int n)`: cria um nó com os parâmetros indicados.

Árvore 2-3: Inserção

Função principal:

```
 // Insere a chave na árvore 2-3
arvore23 inserir(arvore23 r, int chave){
  if(vazia(r)) // caso base especial: a árvore é vazia
    return criaNo23(chave,0,NULL,NULL,NULL,1);
  else {
    int chave_promovida;
    arvore23 aux = inserir_aux(r,chave,&chave_promovida);
    if(!vazia(aux))  // cria nova raiz
      return criaNo23(chave_promovida, 0,r,aux,NULL,1);
    else // raiz não se altera
      return r;
  }
}
```

Árvore 2-3: Inserção

```
//Insere uma chave em uma árvore 2-3 retornando nó gerado pelo split e
// a chave a ser promovida
// Pré-condição: raiz não vazia
// Pós-condição: chave inserida em alguma subárvore de r
// Retorno: nó gerado no split e chave promovida ou NULL caso não tenha
// ocorrido split

arvore23 inserir_aux(arvore23 r, int chave, int* chave_promovida) {
  if(eh_folha(r)){ // caso base: está em uma folha
    if(r->n == 1) {
      adicionaChave(r,chave,NULL);
      return NULL;
    }
    else return split(r, chave, NULL, chave_promovida);
  }
  else { // precisa descer
```

Árvore 2-3: Inserção

```
// continuação de insercao_aux
  else { // precisa descer
    arvore23 paux;
    int ch_aux;
    if (chave < r->chave_esq)
      paux = inserir_aux(r->esq, chave, &ch_aux);
    else if ((r->n == 1) || (chave < r->chave_dir))
      paux = inserir_aux( r->meio, chave, &ch_aux);
    else
      paux = inserir_aux(r->dir, chave, &ch_aux);
    if (paux == NULL) // nao promoveu
       return NULL;
    if (r->n == 1){
       adicionaChave(r, ch_aux, paux);
       return NULL;
    }
    else // precisa fazer split
       return split(r, ch_aux, paux,chave_promovida);
  }
}
```

Árvore 2-3: Inserção

```
arvore23 split(arvore23 p, int chave, arvore23 subarvore,
               int *chave_promovida){
 arvore23 paux;
 if (chave > p->chave_dir) { // chave ficará mais a direita
   *chave_promovida = p->chave_dir; // promove a antiga maior
   paux = p->dir;
   p->dir = NULL; // elimina o terceiro filho
   p->n = 1; // atualiza o número de chaves
   return criaNo23(chave, 0, paux, subarvore, NULL, 1);
 }
 if (chave >= p->chave_esq){ // chave está no meio
        *chave_promovida = chave; // continua sendo promovida
        paux = p->dir;
        p->dir = NULL;
        p->n = 1;
        return criaNo23(p->chave_dir, 0, subarvore, paux, NULL,1);
 }
 // chave ficará mais à esquerda
```

Árvore 2-3: Inserção

```
  //continuacao de split
  // chave ficará mais à esquerda
  *chave_promovida = p->chave_esq;
  // primeiro cria o nó à direita
  paux = criaNo23(p->chave_dir, 0, p->meio, p->dir, NULL,1);
  p->chave_esq = chave; // atualiza o nó à esquerda
  p->n = 1;
  p->dir = NULL;
  p->meio = subarvore;
  return paux;
}
```

Árvore 2-3: Inserção

```
// Adiciona uma chave em um nó que tem 1 chave
// Pré-condição: nó r tem somente uma chave
// Pós-condição: insere chave no nó r com subárvore p
void adicionaChave(arvore23 r, int chave, arvore23 p) {
  if(r->chave_esq < chave) {
    r->chave_dir = chave;
    r->dir = p;
  }
  else {
    r->chave_dir = r->chave_esq;
    r->chave_esq = chave;
    r->dir = r->meio;
    r->meio = p;
  }
  r->n = 2;
}
```

Árvore 2-3: remoção

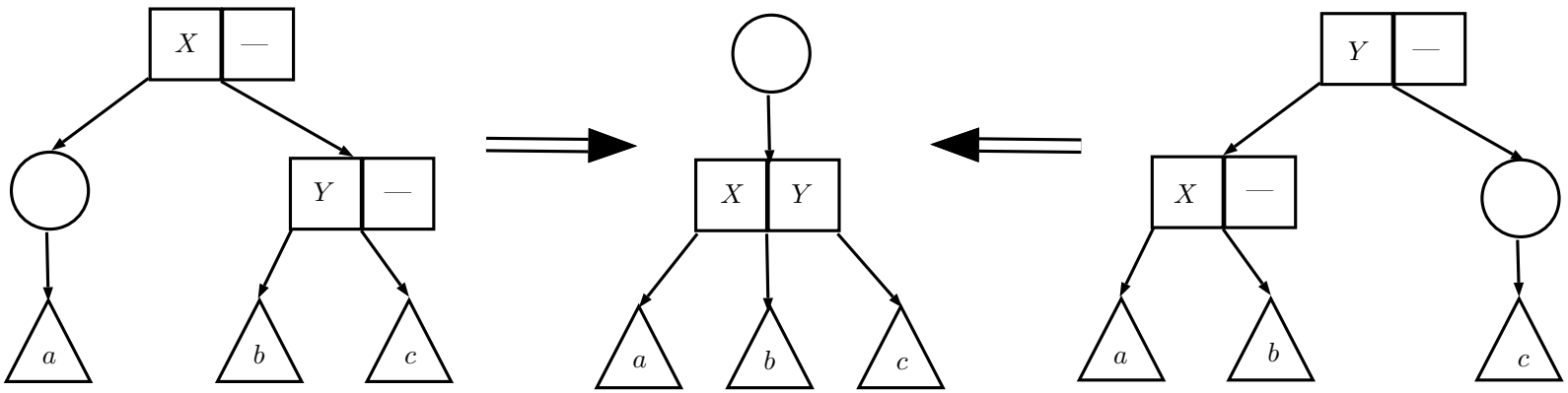
Remoção

A remoção em uma árvore 2-3 tem os seguintes casos:

1. a remoção é na folha:
   - o nó tem 2 chaves: elimina-se a chave desejada
   - o nó tem 1 chave: o nó ficará vazio e é feita uma redistribuição/*merge* com nós vizinhos. O processo de redistribuição/*merge* é repetido nos níveis superiores se necessário.
2. a remoção é em um nó interno: encontrar o sucessor *in-ordem* (será uma folha) e copiá-la por cima da chave a ser removida e remover recursivamente a chave copiada, semelhante ao esquema de uma árvore binária
3. se a remoção causar a raiz ficar vazia, ela é eliminada e o filho torna-se a nova raiz.

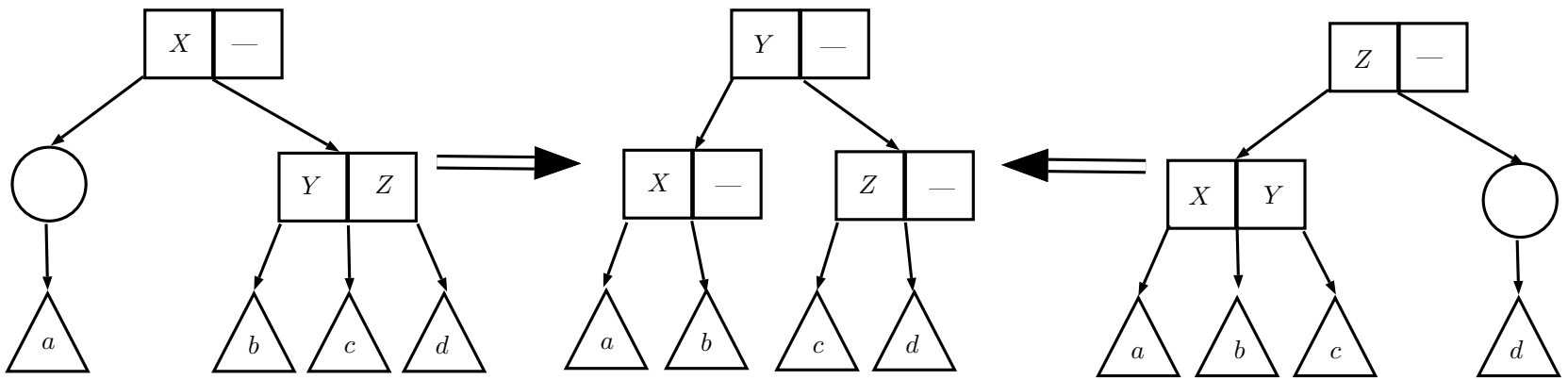
Árvore 2-3: Esquema da remoção

Nó vazio é representado pelo círculo. Irmão do nó vazio só tem uma chave. Operação de *merge* é necessária.



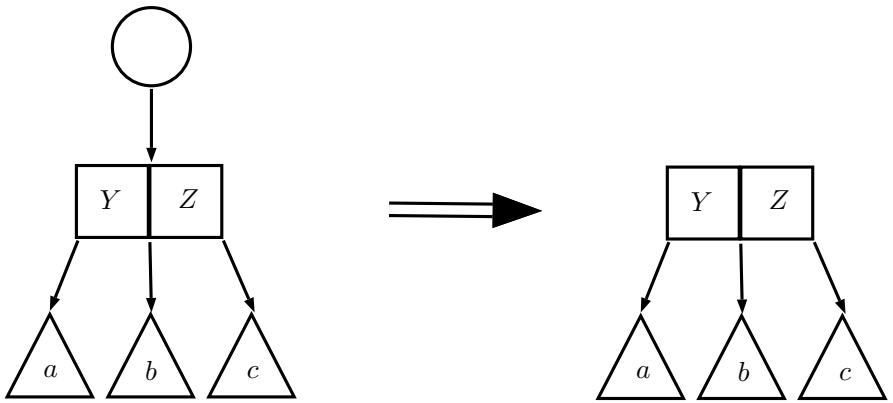
Árvore 2-3: Esquema da remoção

Nó vazio é representado pelo círculo. Irmão do nó vazio tem 2 chaves. Operação de redistribuição é necessária.



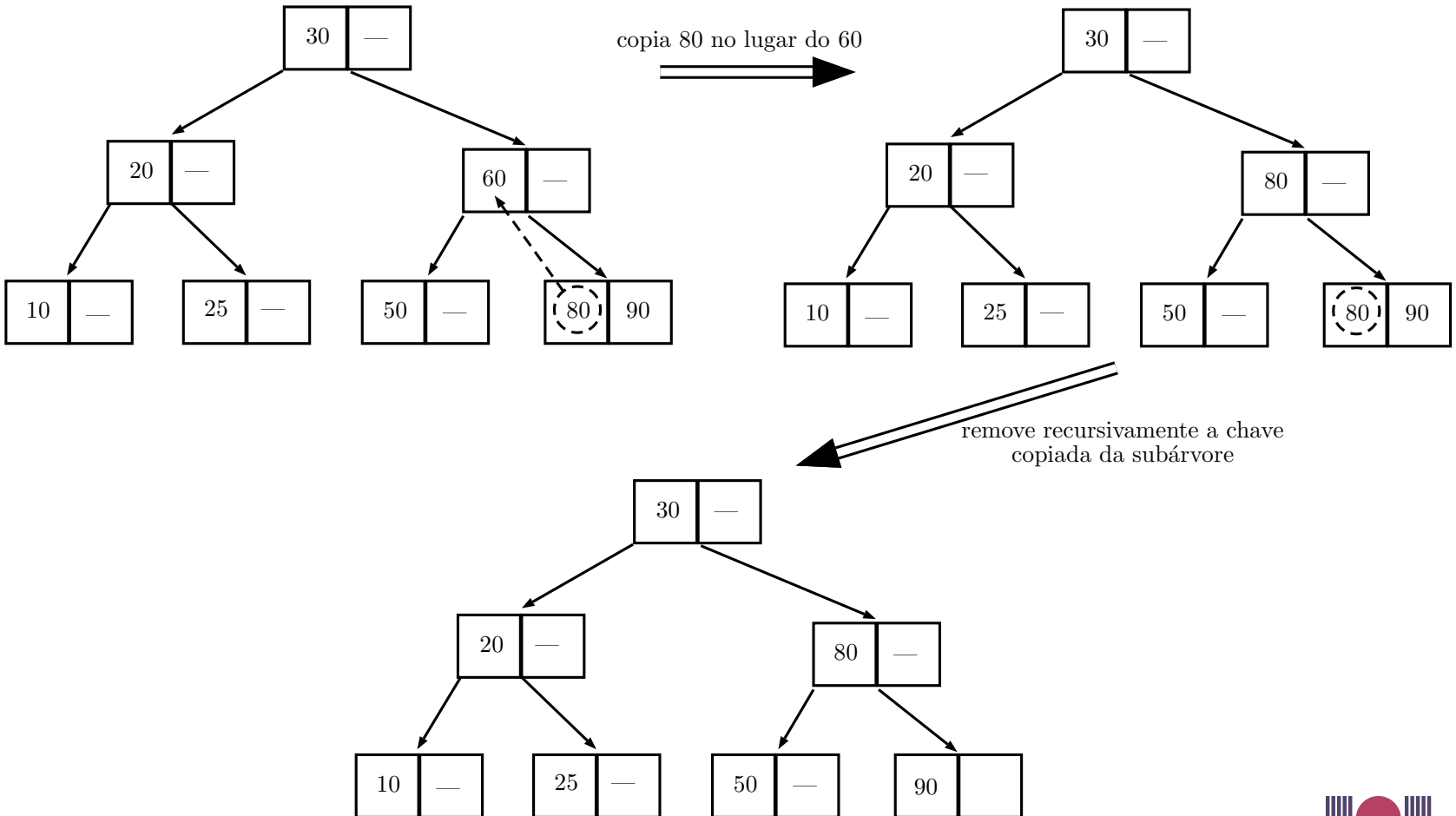
Árvore 2-3: Esquema da remoção

Raiz ficou vazia. O filho será a nova raiz.



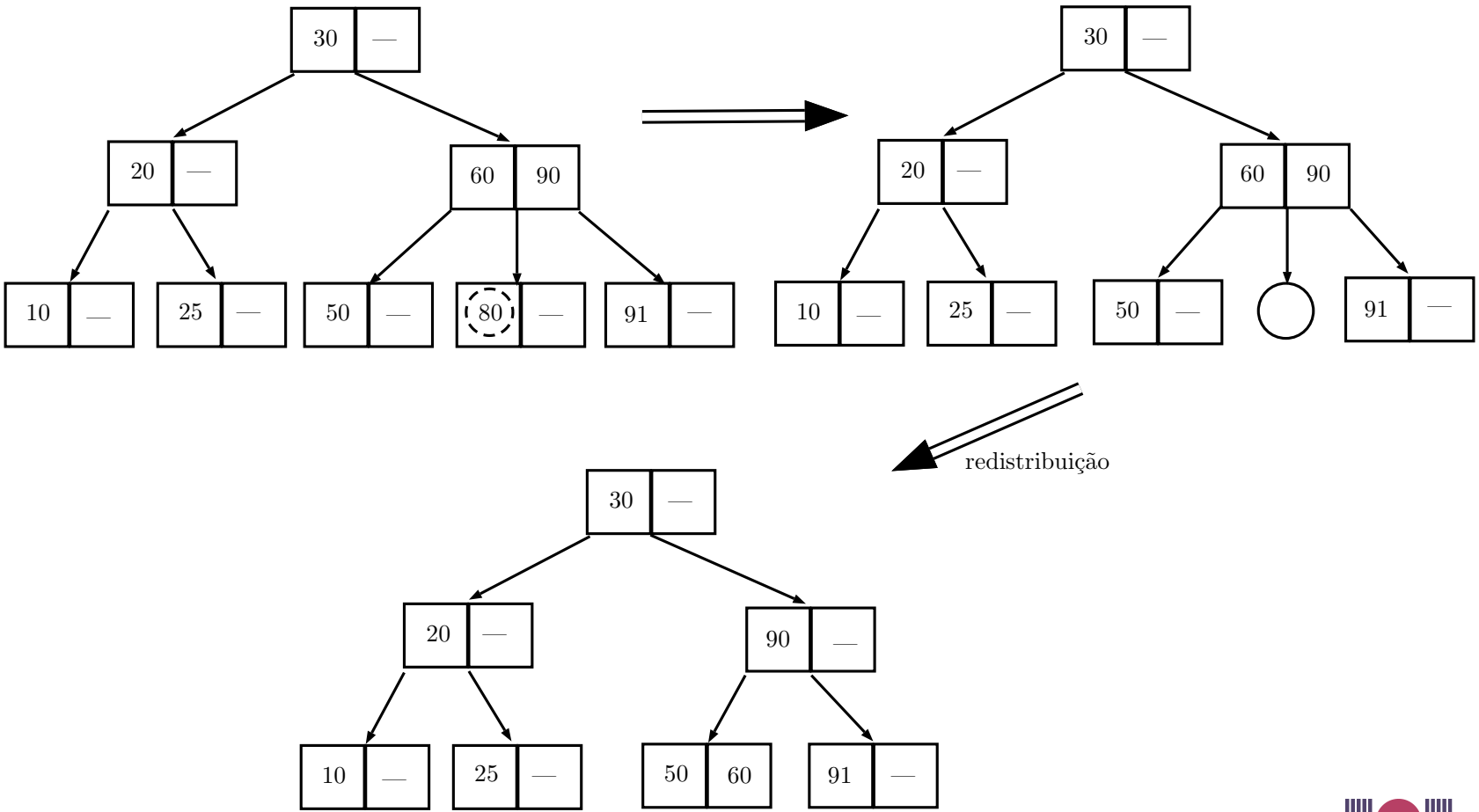
Árvore 2-3: Remoção - exemplo

Removendo a chave 60:



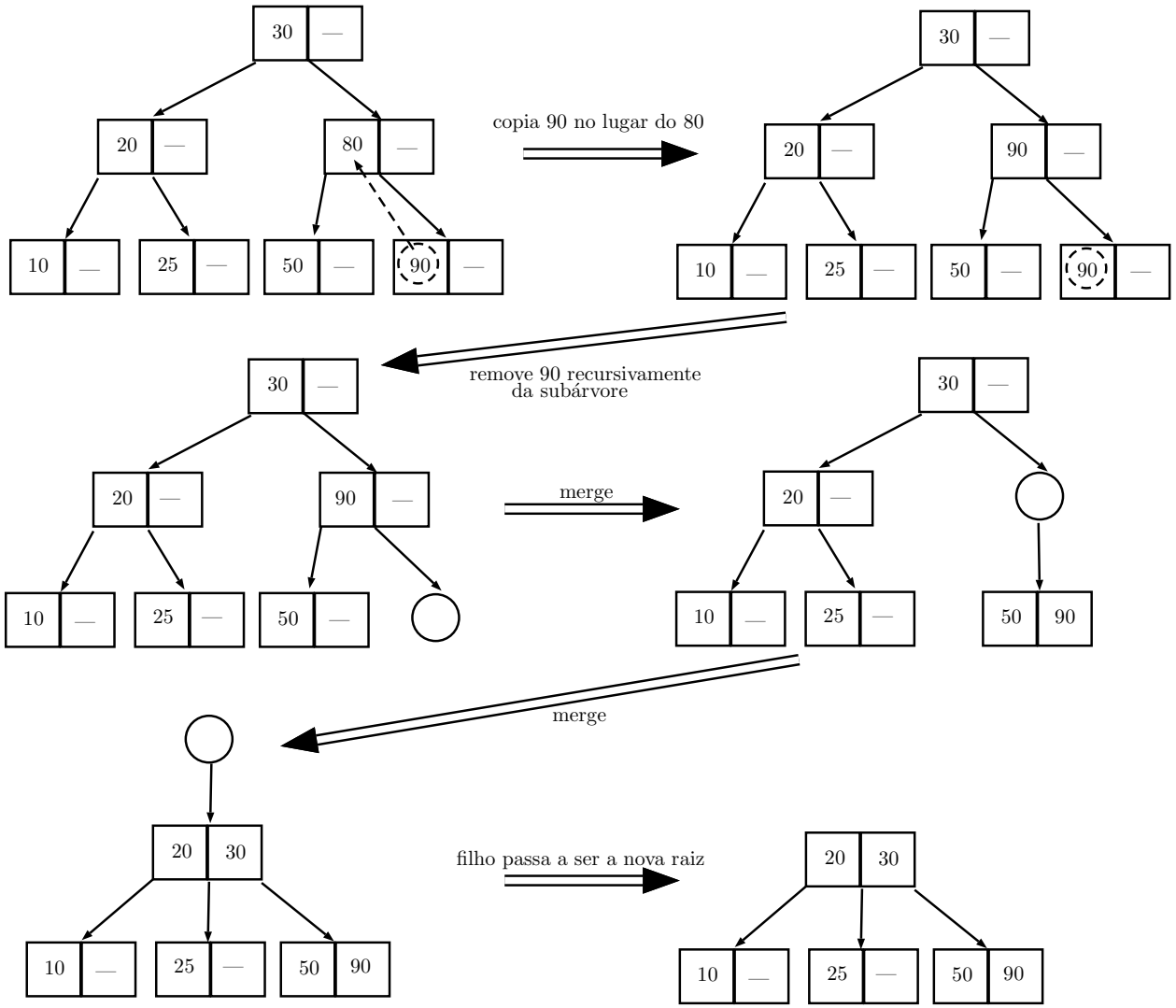
Árvore 2-3: Remoção - exemplo

Removendo a chave 80:



Árvore 2-3: Remoção - exemplo

Removendo a chave 80:



Bibliografia I

[Turbak 2004] Turbak, Lyn.
Data Structures: handout #26, Wellesley College, MA, 2004, disponível em `http://cs.wellesley.edu/~cs230/fall02/2-3-trees.pdf`

[Redekopp] Redekopp, Mark; kempe, David.
CSCI 104: 2-3 Trees, University of Southern California, disponível em `http://ee.usc.edu/~redekopp/cs104/slides/L19_BalancedBST_23.pdf`