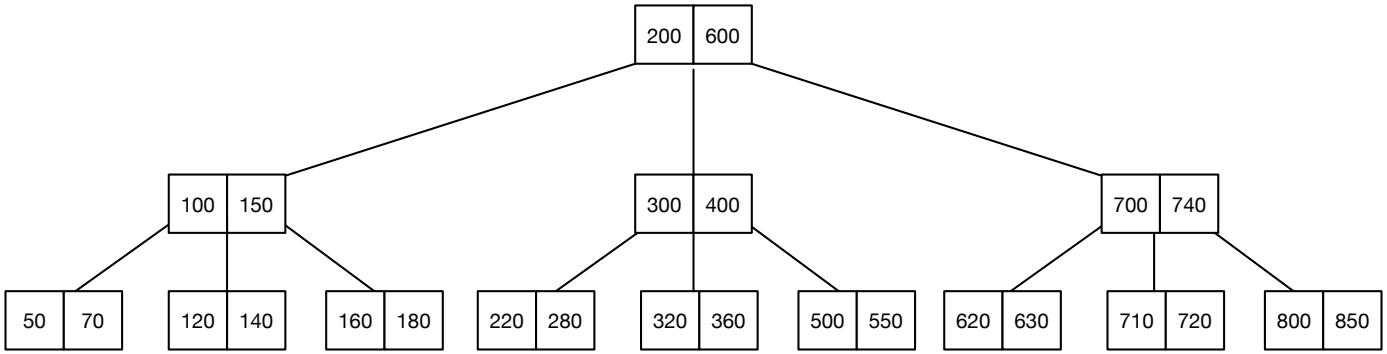
Laboratório/Trabalho: Implementação de árvores 2-3

- árvores 2-3: árvores B de ordem 3
- cada nó tem no máximo 2 chaves e no mínimo 1 chave

Exemplo de árvore 2-3



árvore 2-3 - implementação

```
struct smapa
{
        int  kp, kg;    /* chaves: kp<kg, se kg existir. Se kg=-1,
                           significa que ele não existe. */
        Mapa  *pai;
        Mapa  *esq;
        Mapa  *meio;
        Mapa  *dir;
};
```

árvore 2-3 - implementação - idéia inserção

- se tem espaço no nó, coloca chave nele
  - pode ser necessário "empurrar" a chave existente para a direita
- caso contrário, *quebra* (*overflowquebra*) em dois
  - valor mediano e ponteiro para novo nó retornados na recursão

árvore 2-3 - implementação

```
int insere2 (Mapa* m, int chave, int* valorainserir, Mapa** novofilho) {

  int inseriraqui = 0; /* indica se deve inserir neste nó */

  if (m==NULL) {
    printf("erro! subárvore nula! \n"); exit (1);
  }

  if (m->esq != NULL) {
    if (chave < m->kp) {
      ...
    }
    else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1)) {
      ...
    }
    else { /* chave > m->kg */
      ...
    }
  }
  else { /* este nó é folha, tem que inserir nele de qq jeito */
    *valorainserir = chave;
    inseriraqui = 1;
    *novofilho = NULL;
  }

  if (!inseriraqui) return 0; /* inserção já está completa */
  ...
```

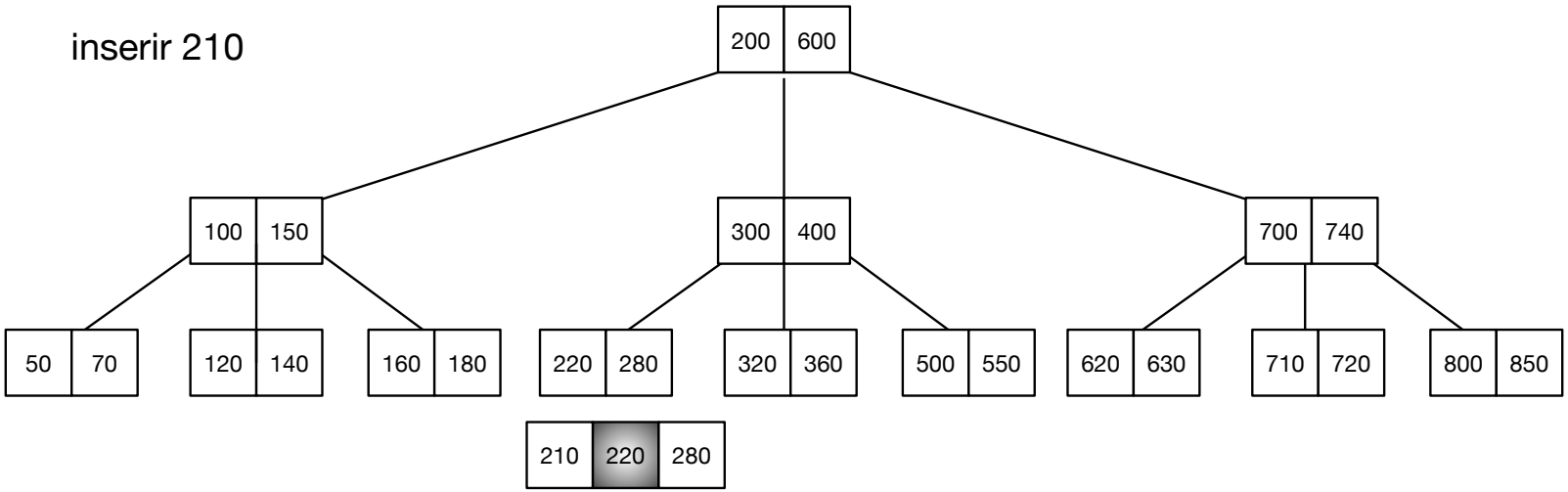
árvore 2-3 - implementação

```
int insere2 (Mapa* m, int chave, int* valorainserir, Mapa** novofilho) {

  int inseriraqui = 0; /* indica se deve inserir neste nó */

  if (m==NULL) {
    printf("erro! subárvore nula! \n"); exit (1);
  }

  if (m->esq != NULL) { /* não é folha, só insere neste nó se subir um valor  */
    if (chave < m->kp) {
      inseriraqui = insere2(m->esq, chave, valorainserir, novofilho);
    }
    else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1)) {
      /* ou está entre as duas chaves ou só tem uma chave no nó */
      inseriraqui = insere2(m->meio, chave, valorainserir, novofilho);
    }
    else { /* chave > m->kg */
      inseriraqui = insere2(m->dir, chave, valorainserir, novofilho);
    }
  }
  else { /* este nó é folha, tem que inserir nele de qq jeito */
    *valorainserir = chave;
    inseriraqui = 1;
    *novofilho = NULL;
  }

  if (!inseriraqui) return 0; /* inserção já está completa */
  ...
```

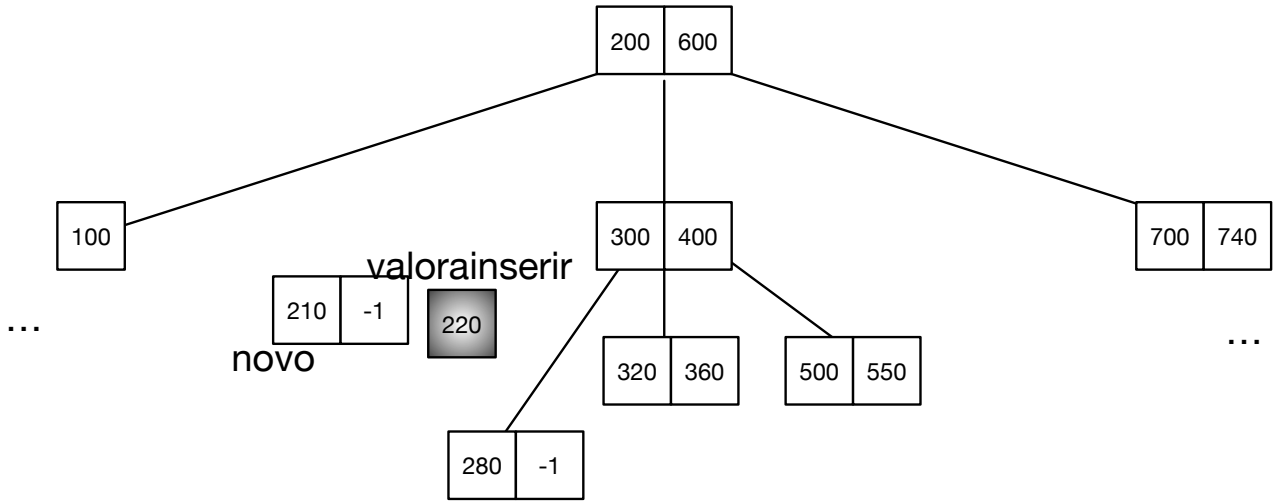
árvore 2-3 - implementação

```
int insere2 (Mapa* m, int chave, int* valorainserir, Mapa** novofilho) {

  int inseriraqui = 0; /* indica se deve inserir neste nó */

  ...

  if (!inseriraqui) return 0; /* inserção já está completa */

  if (m->kg==-1) {
    /* tem  espaço no nó */
    /* COMPLETAR */

     return 0; /* como havia espaço, não sobem valores a serem inseridos */
  }

  *novofilho = overflowQuebra (m, valorainserir, *novofilho);

  return 1; /* quando há quebra sempre sobe a mediana para nova inserção */
}
```

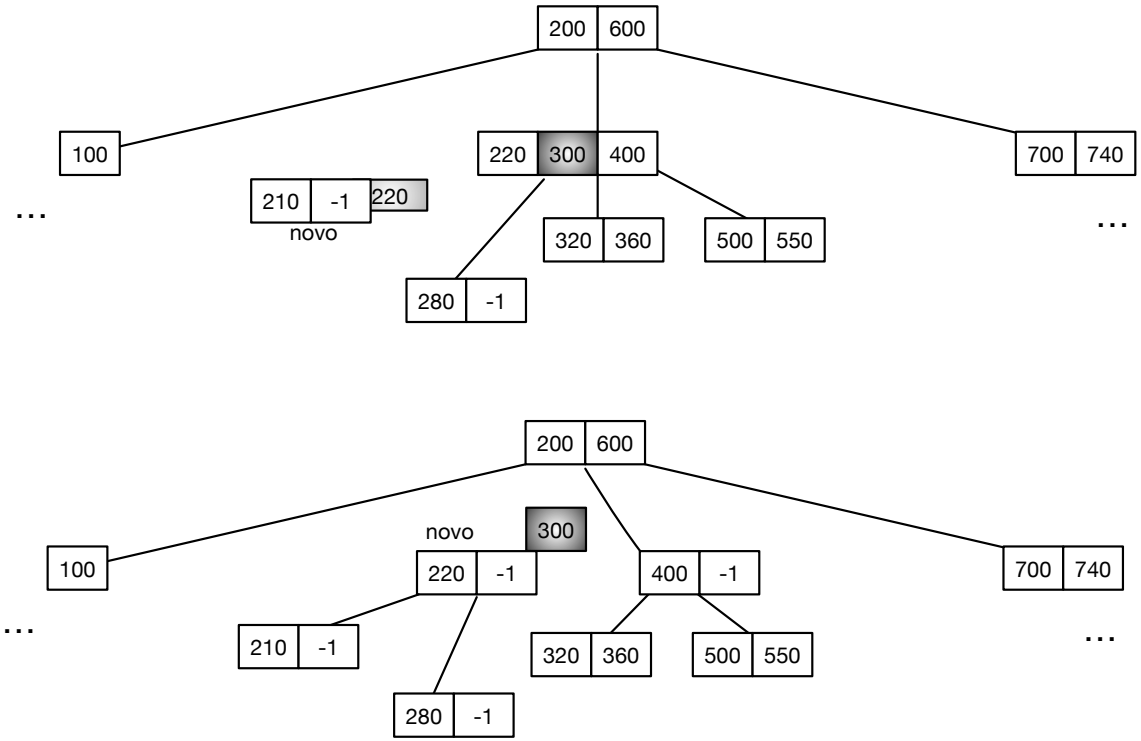
Exemplo de árvore 2-3



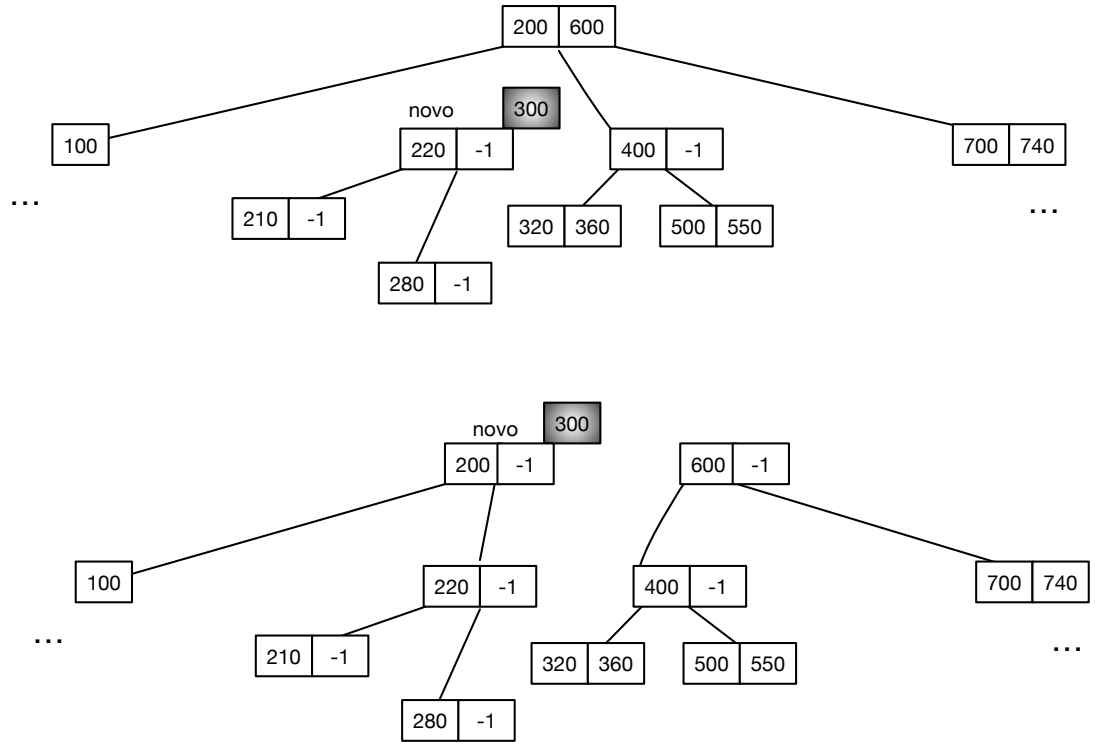
Exemplo de árvore 2-3



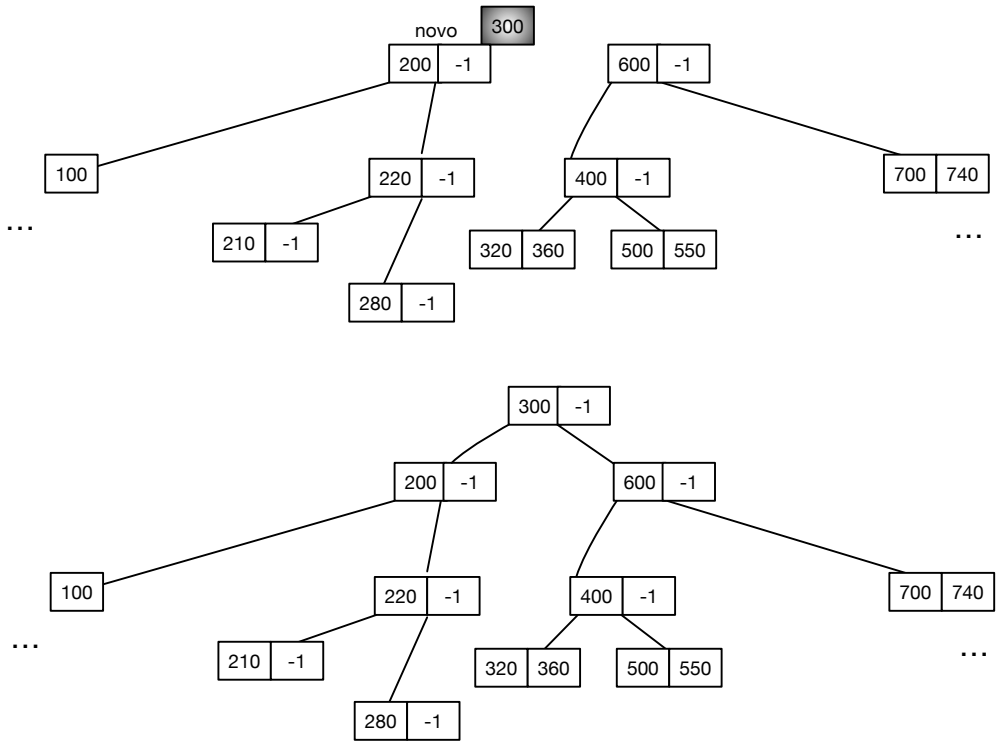
Exemplo de árvore 2-3



Exemplo de árvore 2-3



Exemplo de árvore 2-3



árvore 2-3 - implementação

```
static Mapa* overflowQuebra (Mapa *m, int *valorainserir, Mapa* novofil
  Mapa* novo;

  novo = (Mapa*) malloc(sizeof(struct smapa));
  ...
  if (*valorainserir < m->kp) {
    /* FEITO */
  else if (*valorainserir < m->kg) {
    /* completar */
  }
  else {
    /* completar */
  }
  return novo;
}
```

árvore 2-3 - implementação

```
static Mapa* overflowQuebra (Mapa *m, int *valorainserir, Mapa* novofil
  Mapa* novo;
  novo = (Mapa*) malloc(sizeof(struct smapa));

  if (*valorainserir < m->kp) {
    novo->esq = novofilho;
    if (novo->esq) novo->esq->pai = novo;
    novo->kp = *valorainserir;
    novo->meio = m->esq;
    if (novo->meio) novo->meio->pai = novo;
    novo->kg = -1;
    novo->dir = NULL;
    *valorainserir = m->kp;
    m->esq = m->meio;
    m->kp = m->kg;
  }
  else if (*valorainserir < m->kg) {
    /* completar */
  }
  else {
    /* completar */
```

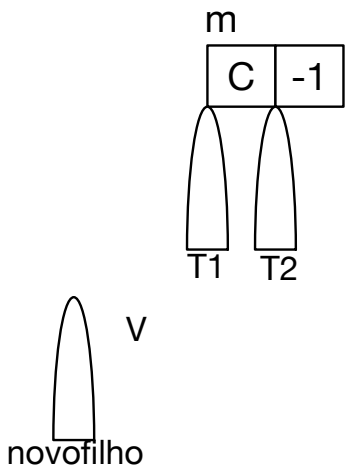
árvore 2-3 - implementação

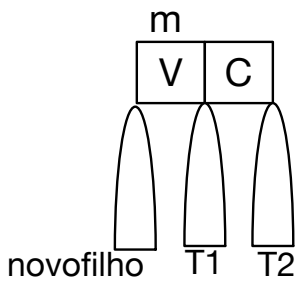
```
static Mapa* overflowQuebra (Mapa *m, int *valorainserir, Mapa* novofil
  Mapa* novo;
  novo = (Mapa*) malloc(sizeof(struct smapa));

  if (*valorainserir < m->kp) {
    novo->esq = novofilho;
    if (novo->esq) novo->esq->pai = novo;
    novo->kp = *valorainserir; novo->meio = m->esq;
    if (novo->meio) novo->meio->pai = novo;
    novo->kg = -1; novo->dir = NULL;
    *valorainserir = m->kp;
    m->esq = m->meio; m->kp = m->kg;
  }
  else if (*valorainserir < m->kg) { /* completar */ }
  else { /* completar */ }
  m->meio = m->dir;
  m->kg = -1;
  m->dir = NULL;
  return novo;
}
```

árvore 2-3 - implementação

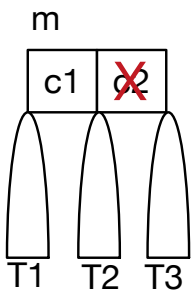
```
int insere2 (Mapa* m, int chave, int* valorainserir, Mapa** novofilho);
Mapa* insere (Mapa* m, int chave) {
  int valorquesubiu; Mapa* novofilho; Mapa* novaraiz;
  if (m==NULL) {
    m = novoNo (chave);
    m->pai = novoNo (-1);
  }
  else {
    if (insere2 (m, chave, &valorquesubiu, &novofilho)) {
      /* cria nova raiz */
      novaraiz = novoNo (valorquesubiu);
      novaraiz->pai = m->pai;
      novaraiz->esq = novofilho;
      novaraiz->esq->pai = novaraiz;
      novaraiz->meio = m;
      novaraiz->meio->pai = novaraiz;
      m = novaraiz;
    }
  }

  return m;
```

árvore 2-3 - inserção

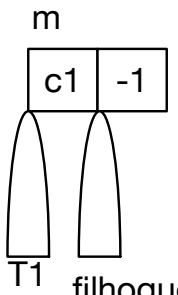




árvore 2-3 - inserção

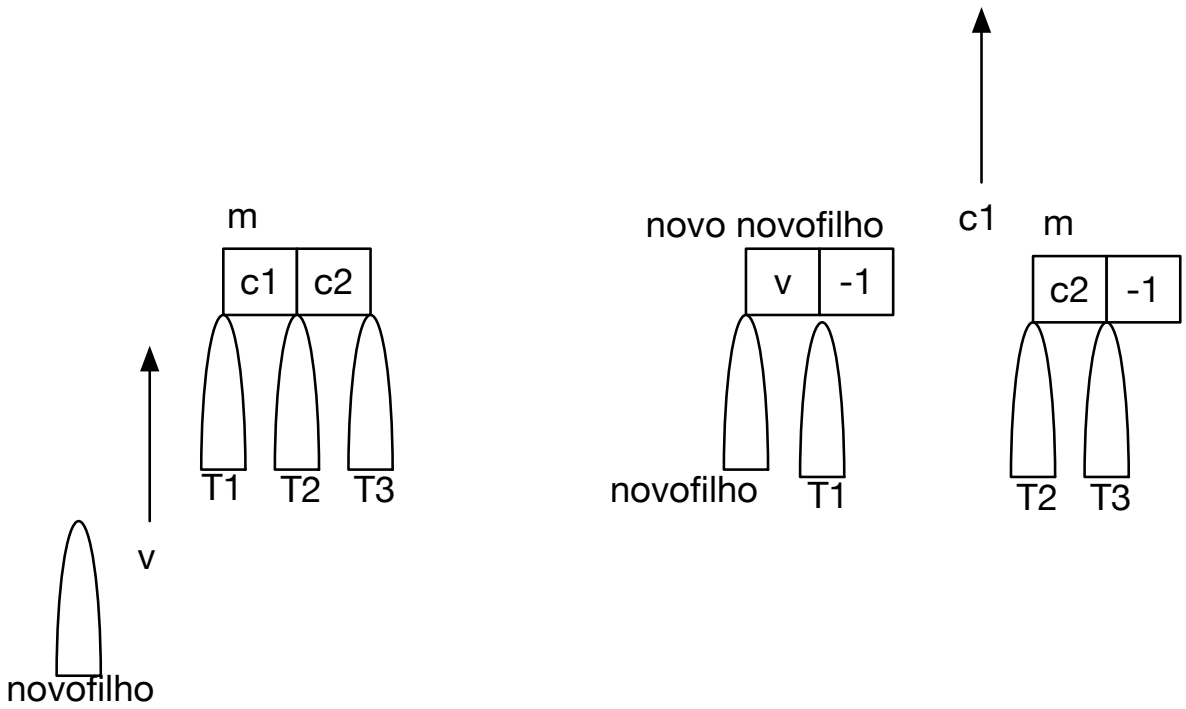


T2 ou T3 “acabaram” - sobrou filhoqueficou

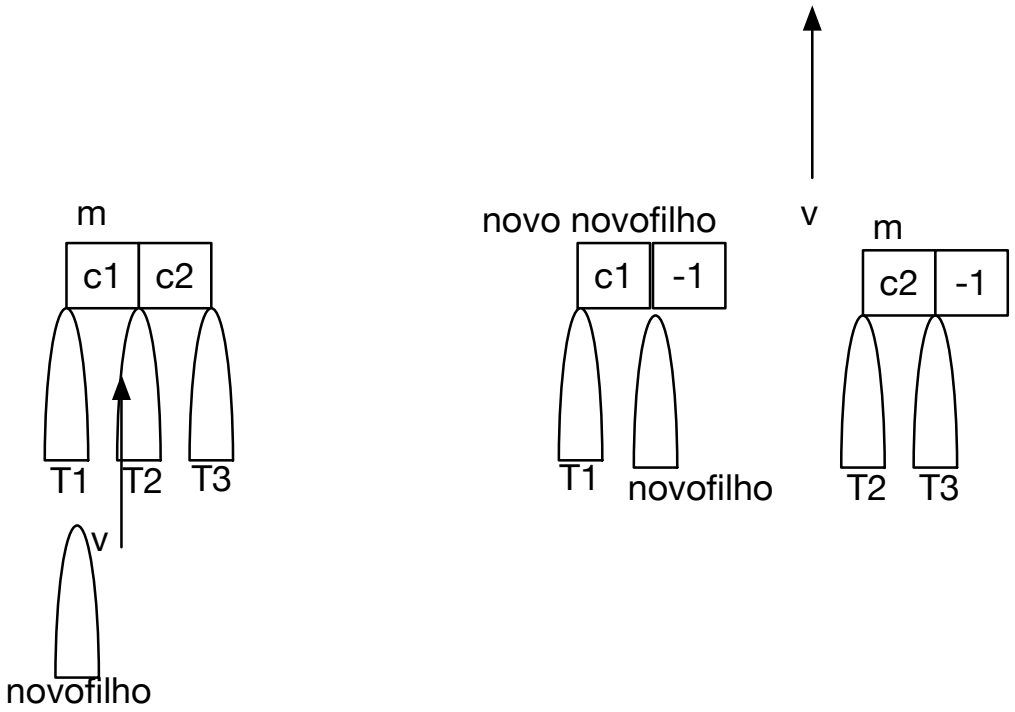


filhoqueficou

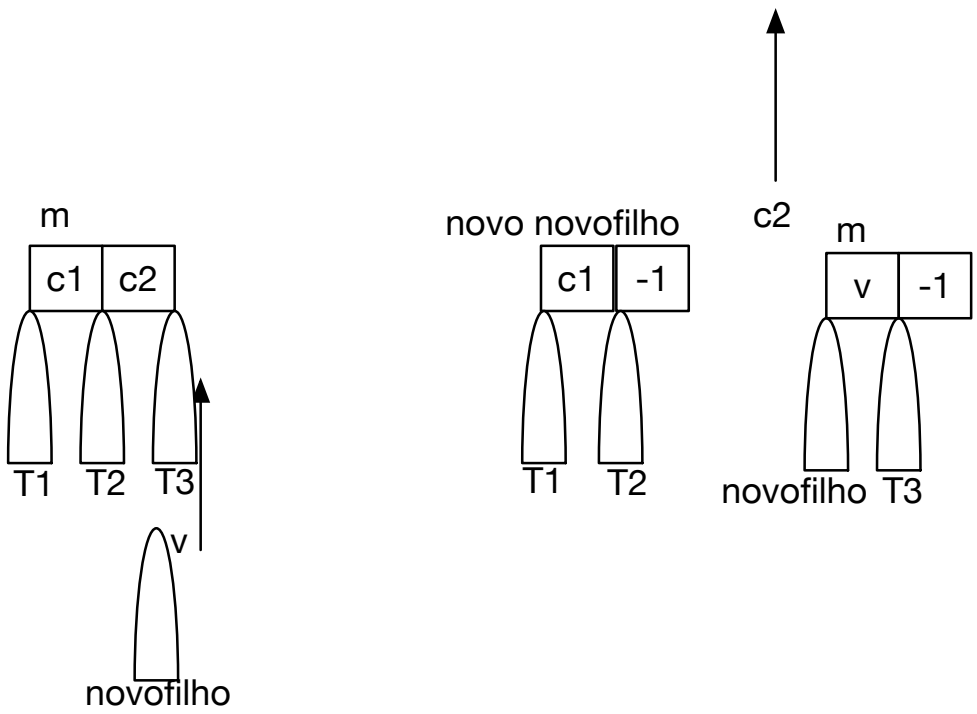
árvore 2-3 - inserção



árvore 2-3 - inserção



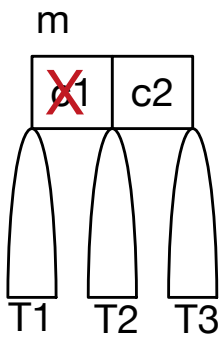
árvore 2-3 - inserção



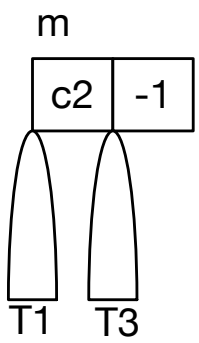
árvore 2-3 - retirada

- árvore sempre deve ficar cheia

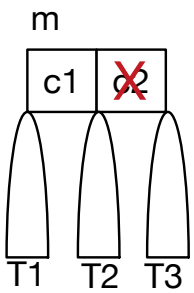
árvore 2-3 - retirada – casos simples



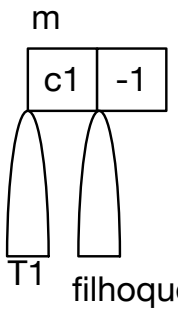
T1 ou T2
“acabaram”



ou
T2

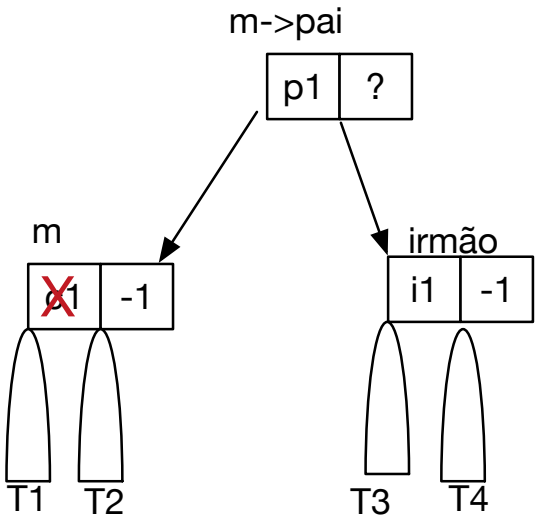


T2 ou T3
“acabaram” -
sobrou
filhoqueficou

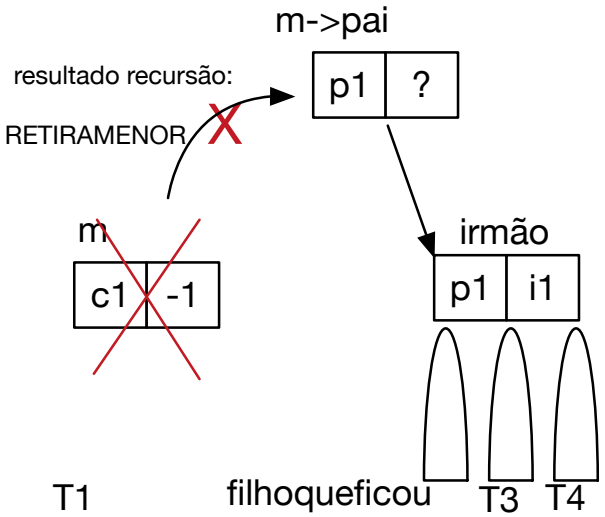


filhoqueficou

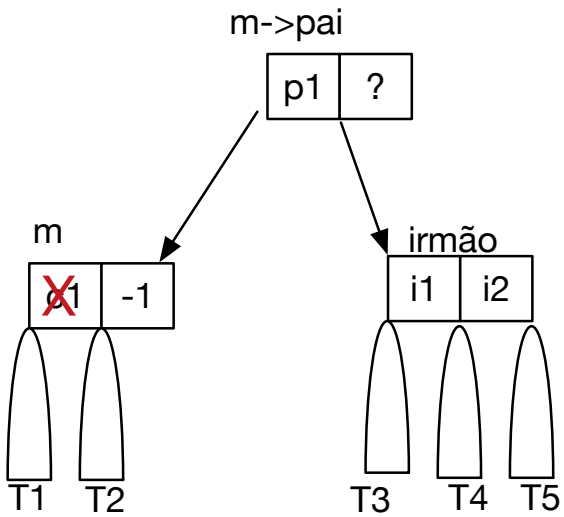
árvore 2-3 - retirada – caso combinação



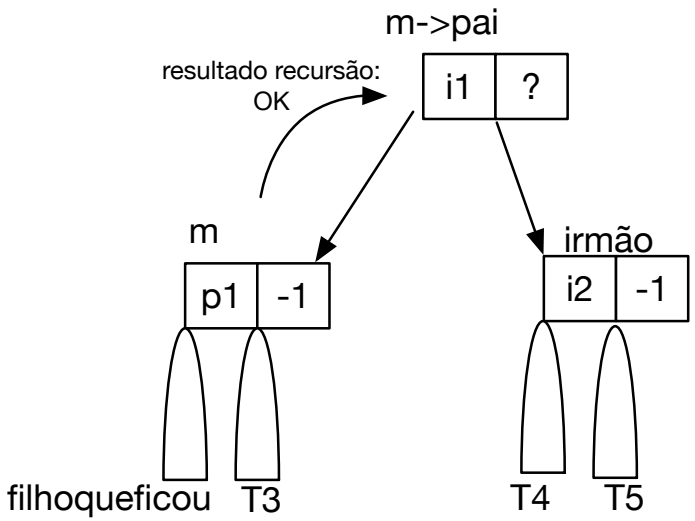
T1 ou T2 “acabaram” - sobrou filhoqueficou



árvore 2-3 - retirada – caso redistribuição



T1 ou T2 “acabaram” - sobrou filhoqueficou



árvore 2-3 - retirada

```
tresultret retirarec (Mapa *m, int chave) {
  ..
  if (m==NULL) { printf("erro! subárvore nula! \n"); exit (1); }

  if (m->esq != NULL) { /* não é folha */
    if (chave < m->kp) {
      res = retirarec (m->esq, chave);
    }
    else if (m->kp == chave) { /* achou - troca por succ */
      m->kp = maisaesquerda (m->meio);
      res = retirarec (m->meio, m->kp);
    }
    else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1)) {
      /* ou está entre as duas chaves ou só tem uma chave no nó */
      res = retirarec(m->meio, chave);
    }
    else if (m->kg == chave) { /* achou - troca por succ */
      m->kg = maisaesquerda (m->dir);
      res = retirarec (m->dir, m->kg);
    }
    else { /* chave > m->kg */
      res = retirarec(m->dir, chave);
    }
    if (res==OK) return OK;
  }
  else { /* este nó é folha, chave tem que estar nele de qq jeito */
    if (chave==m->kp) res = RETIRA_MENOR;
    else if (chave == m->kg) res = RETIRA_MAIOR;
    else /* chave não está na árvore!!! */
      return OK;
  }
  /* retirada */
  ...
```

árvore 2-3 - retirada

```
tresultret retirarec (Mapa *m, int chave) {
  ...
  /* retirada */
  /* pode ser porque estamos em uma folha ou porque "caiu" uma das chaves */
  if (res == RETIRA_MAIOR) { /* caso mais simples */
    preenche (m, m->esq, m->kp, m->meio?m->meio:m->dir, -1, NULL);
    return OK;
  }
  /* RETIRAMENOR */
  if (m->kg != -1) {
  /* ainda vai ficar um no nó, tb simples */
  }
  /* RETIRAMENOR: essa é a única chave! combinar ou distribuir */
  minhapos = minhaposnopai (m->pai, m);
  /* se ainda tiver algum filho pegá-lo para passar para outro */
  filhoqueficou = m->esq?m->esq:m->meio;
  if (minhapos == ESQ) {
    irmao = m->pai->meio;
    if (irmao->kg == -1) { /* combinar */
     ...
    }
    else ...
  }
  else ...
  ...
  return res;
}
```