Árvores B

1 Struct

```
struct smapa
{
  int kp, kg;
  Mapa *pai;
  Mapa *esq;
  Mapa *meio;
  Mapa *dir;
};
```

2 Headers

```
typedef struct smapa Mapa;

Mapa* cria (void);
Mapa* insere (Mapa* m, int chave);
int busca (Mapa *m, int chave);
Mapa* retira (Mapa *m, int chave);
void destroi (Mapa *m);
```

3 Implementation

3.1 Initializing

3.1.1 Create

```
Mapa* cria (void) {
  return NULL;
}

static Mapa* novoNo (int chave) {
  Mapa *m = (Mapa*) malloc(sizeof(struct smapa));
  if (m==NULL) { printf ("erro no malloc! \n"); exit(1);}
  m->pai = NULL;
  m->kp = chave;
  m->kg = -1;
  m->esq = m->meio = m->dir = NULL;
  return m;
}
```

3.2 Insertion

3.2.1 Insere Wrapper

```
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)) {
      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;
}
```

3.2.2 Insere

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

  int inseriraqui = 0;

  if (m==NULL) {
    printf("Erro\n"); exit (1);
  }

  if (m->esq != NULL) {
    if (chave < m->kp) {
      inseriraqui = insere2(m->esq, chave, valorainserir, novofilho);
    }
    else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1)) {
      inseriraqui = insere2(m->meio, chave, valorainserir, novofilho);
    }
    else {
      inseriraqui = insere2(m->dir, chave, valorainserir, novofilho);
    }
  }
  else {
    *valorainserir = chave;
    inseriraqui = 1;
    *novofilho = NULL;
  }

  if (!inseriraqui) return 0;


  if (m->kg==-1) {
    if (*valorainserir < m->kp) {
      m->kg = m->kp;
      m->kp = *valorainserir;
      m->dir = m->meio;
      m->meio = m->esq;
      m->esq = *novofilho;
      if(m->esq) m->esq->pai = m;

    }
    else {
      m->kg = *valorainserir;
      m->dir = m->meio;
```

```
      m->meio = *novofilho;
      if (m->meio) m->meio->pai = m;
    }
    return 0;
  }

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

  return 1;
}
```

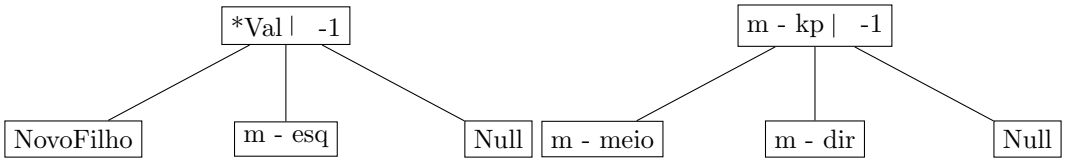
3.2.3 OverFlow

```
static Mapa* overflowQuebra (Mapa *m, int *valorainserir, Mapa* novofilho) {
  Mapa* novo;

  novo = (Mapa*) malloc(sizeof(struct smapa));

  if (novo==NULL) { printf ("erro em malloc! \n"); exit(1);}

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


  }
  m->meio = m->dir;
  m->kg = -1;
  m->dir = NULL;
  return novo;
```

```
}
```

3.3 1ºCaso Overflow

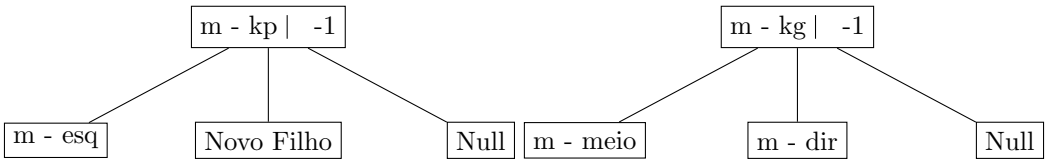
3.3.1 Novo/M



Valor a inserir = m - kp;

3.4 2ºCaso Overflow

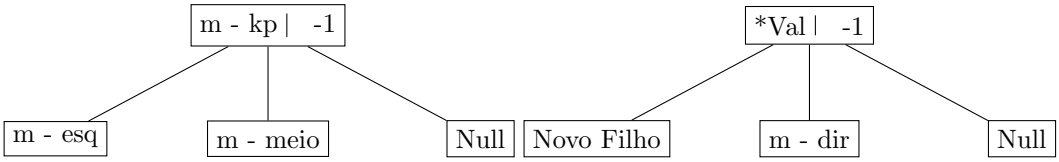
3.4.1 Novo/M



Valor a inserir = Valor a inserir

3.5 3ºCaso Overflow

3.5.1 Novo/M



Valor a inserir = m - kg;

4 Remover

4.1 Retira Wrapper

```
Mapa* retira (Mapa *m, int chave) {
  tresultret res;
  Mapa* novaraiz;


  res = retirarec (m, chave);

  if (res == RETIRA_MAIOR) {
    preenche (m, m->esq, m->kp, m->meio?m->meio:m- >dir, -1, NULL);
  }

  else if (res == RETIRA_MENOR) {
```

```
    if (m->kg != -1) {
      preenche (m, m->esq?m->esq:m->meio, m->kg, m- >dir, -1, NULL);
    }

    else {
      novaraiz = (m->esq)?m->esq:m->meio;
      novaraiz->pai = m->pai;
      free(m);
      m = novaraiz;
    }
  }

  return m;
}
```

4.2 Retira Rec

```
static tresultret retirarec (Mapa *m, int chave) {

  tresultret res;
  tpos minhapos;
  Mapa *filhoqueficou, *irmao;

  if (m==NULL) {
    printf("erro! \n"); exit (1);
  }
  if (m->esq != NULL) {
    if (chave < m->kp) {
      res = retirarec (m->esq, chave);
    }
    else if (m->kp == chave) {
      m->kp = maisaesquerda (m->meio);
      res = retirarec (m->meio, m->kp);
    }
    else if (((m->kg != -1) && (chave < m->kg)) || (m->kg == -1)) {
      res = retirarec(m->meio, chave);
    }
    else if (m->kg == chave) {
      m->kg = maisaesquerda (m->dir);
      res = retirarec (m->dir, m->kg);
    }
    else {
      res = retirarec(m->dir, chave);
    }
    if (res==OK) return OK;
  }
  else {
    if (chave==m->kp) res = RETIRA_MENOR;
    else if (chave == m->kg) res = RETIRA_MAIOR;
    else
      return OK;
  }
  if (res == RETIRA_MAIOR) {
    preenche (m, m->esq, m->kp, m->meio?m->meio:m- >dir, -1, NULL);
    return OK;
}

  if (m->kg != -1) {
    preenche (m, m->esq?m->esq:m->meio, m->kg, m- >dir, -1, NULL);
```

```
    return OK;
  }
  minhapos = minhaposnopai (m->pai, m);
  filhoqueficou = m->esq?m->esq:m->meio;
  if (minhapos == ESQ) {
    irmao = m->pai->meio;
    if (irmao->kg == -1) {
      preenche (irmao, filhoqueficou, m->pai->kp, irmao->esq, irmao->kp, irmao->meio);
      if (irmao->esq) irmao->esq->pai = irmao;
      m->pai->esq = NULL;
      free(m);
      res = RETIRA_MENOR;
    }
    else {
      preenche (m, filhoqueficou, m->pai->kp, irmao- >esq, -1, NULL);
      if (m->esq) m->esq->pai = m;
      if (m->meio) m->meio->pai = m;
      preenche (m->pai, m->pai->esq, irmao->kp, m- >pai->meio, m->pai->kg, m->pai->dir);
      preenche (irmao, irmao->meio, irmao->kg, irmao->dir, -1, NULL);
      res = OK;
    }
  }
  else if (minhapos == MEIO) {
    irmao = m->pai->esq;
    if (irmao->kg == -1) {
      preenche(irmao, irmao->esq, irmao->kp, irmao- >meio, m->pai->kp, filhoqueficou);
      if(irmao->dir) irmao->dir->pai = irmao;
      m->pai->meio = NULL;
      free(m);
      res = RETIRA_MENOR;

    }
    else {
preenche(m,irmao->dir,m->pai->kp,filhoqueficou, -1,NULL);
      if(m->esq) m->esq->pai = m;
      if(m->meio) m->meio->pai = m;
      preenche(m->pai,m->pai->esq, irmao->kg, m->pai->meio, m->pai->kg, m->pai->dir);
      preenche(irmao, irmao->esq, irmao->kp, irmao->meio, -1,NULL);
      res = OK;
    }
  }
  else if (minhapos == DIR) {
    irmao = m->pai->meio;
    if (irmao->kg == -1) {
      preenche(irmao,irmao->esq,irmao->kp,irmao->meio,m->pai->kg,filhoqueficou);
      if(irmao->dir) irmao->dir->pai = irmao;
      free(m);
      res = RETIRA_MAIOR;
    }
    else {
      preenche(m,irmao->dir,m->pai->kg, filhoqueficou,-1,NULL);
      if(m->esq) m->esq->pai=m;
      if(m->meio) m->meio->pai=m;
      preenche(m->pai,m->pai->esq,m->pai->kp,m->pai- >meio,irmao->kg,m->pai->dir);
      preenche(irmao,irmao->esq,irmao->kp,irmao->meio,-1,NULL);
      res = OK;
    }
  }
  return res;
}
```