```
vazio insercaoEmArvoreB (Arvore *arvore, TipoChave K)
inicio
   encontre um nó folha para inserir K;
   enquanto (1)
      encontre uma posição apropriada no vetor de chaves para K;
      se (nó não está cheio) entao
         insira K e aumente a quantidade de chaves no nó;
         retorne;
      senao
         divida o nó em nó1 e nó2; //nó1 == nó, nó2 é novo
         distribua as chaves e os ponteiros igualmente entre nó1 e nó2
    e inicialize apropriadamente sua quantidades de chaves
         K = última chave de nó1; //a qual é retirada de nó1
         se (nó era a raiz) entao
            crie uma nova raiz como ascendente de nó1 e nó2;
            coloque K e ponteiros para nó1 e nó2 na raiz e ajuste sua
    quantidade de chaves para 1;
            retorne;
         senao
            nó = seu ascendente; //e agora processe o ascendente de nó
         fimse
      fimse
   fimenquanto
fim
```

Árvores B

Com base no algoritmo apresentado elabore uma função em C que execute a operação em questão.

Árvores B

Analisaremos agora a implementação na linguagem C de uma função que efetua a inserção de uma chave em uma árvore B.

Visando possibilitar uma maior flexibilidade à árvore B trabalhada redefiniremos as estruturas de dados capazes de representar a árvore.

```
#define ordem 5
#define maximoDeChaves ordem-1
/*#define minimoDeChaves (int)ceil(ordem/2.0)-1*/
#define minimoDeChaves 2
typedef int TipoChave;
typedef struct
{    TipoChave chave;
     /* ... */
} Registro;
```

Árvores B

```
typedef struct _No
{
    int numChaves;
    Registro registros[maximoDeChaves];
    struct _No *filhos[ordem];
} No;
typedef No *ArvoreB;
```

A maior complexidade do processo de inserção reside no ajuste dos ponteiros da estrutura.

Visando proporcionar uma manipulação adequada dos mesmos, o processo de inserção pode ser dividido em três funções, sendo uma recursiva. Analisaremos agora estas funções.

```
void inserir(Registro reg, ArvoreB *arvore)
{
  int cresceu;
  Registro regRetorno;
  No *filhoDir;
  No *novaRaiz;
  inserirAux(reg, *arvore, &cresceu, &regRetorno, &filhoDir);
  if (cresceu) {
    novaRaiz = (No *) malloc(sizeof(No));
    novaRaiz->numChaves = 1;
    novaRaiz->registros[0] = regRetorno;
    novaRaiz->filhos[1] = filhoDir;
    novaRaiz->filhos[0] = *arvore;
    *arvore = novaRaiz;
  }
}
```

```
void inserirAux(Registro reg, ArvoreB arvore, int *cresceu, Registro
    *regRetorno, ArvoreB *arvRetorno)
{
  No *temp;
  int i, j;
  if (arvore == NULL)
  {
    *cresceu = 1;
    *regRetorno = reg;
    *arvRetorno = NULL;
    return;
  }

  for (i=1; i < arvore->numChaves &&
  reg.chave > arvore->registros[i - 1].chave; i++);
  if (reg.chave == arvore->registros[i - 1].chave)
  {
    printf("\n Erro: Registro ja esta presente.\n");
    *cresceu = 0;
    return;
  }
```

```
if (reg.chave < arvore->registros[i - 1].chave)
  inserirAux(reg, arvore->filhos[i - 1], cresceu, regRetorno,
   arvRetorno);
else
  inserirAux(reg, arvore->filhos[i], cresceu, regRetorno,
   arvRetorno);
if (!*cresceu)
  return;
if (arvore->numChaves < maximoDeChaves)
{
  inserirChaveNoNo(arvore, *regRetorno, *arvRetorno);
  *cresceu = 0;
  return;
}

temp = (No *) malloc(sizeof(No));
temp->numChaves = 0;
temp->filhos[0] = NULL;
if (i <= minimoDeChaves + 1)
{
  inserirChaveNoNo(temp, arvore->registros[maximoDeChaves - 1],
   arvore->filhos[maximoDeChaves]);
  arvore->numChaves--;
  inserirChaveNoNo(arvore, *regRetorno, *arvRetorno);
}
```

```
  else
    inserirChaveNoNo(temp, *regRetorno, *arvRetorno);
  for (j = minimoDeChaves + 2; j <= maximoDeChaves; j++)
    inserirChaveNoNo(temp, arvore->registros[j - 1], arvore->filhos[j]);
  arvore->numChaves = minimoDeChaves;
  temp->filhos[0] = arvore->filhos[minimoDeChaves + 1];
  *regRetorno = arvore->registros[minimoDeChaves];
  *arvRetorno = temp;
  /*Perceba que a execução da função acaba e cresceu continua
  com o valor recebido*/
}
```

```
void inserirChaveNoNo(ArvoreB arvore, Registro reg, No *
    subArvDir)
{
  int k;
  int naoAchouPosicao;
  k = arvore->numChaves;
  naoAchouPosicao = k > 0;
  while (naoAchouPosicao)
  {
    if (reg.chave >= arvore->registros[k - 1].chave)
    {
      naoAchouPosicao = 0;
      break;
    }
    arvore->registros[k] = arvore->registros[k - 1];
    arvore->filhos[k + 1] = arvore->filhos[k];
    k--;
    if (k < 1)
      naoAchouPosicao = 0;
  }
  arvore->registros[k] = reg;
  arvore->filhos[k + 1] = subArvDir;
  arvore->numChaves++;
}
```