Declaração da página de árvore-B (em C)(ordem da árvore é dada pela constante `MAXKEYS`)

```
Struct BTPAGE {
      short KEYCOUNT;
      char KEY[MAXKEYS];
      short CHILD[MAXKEYS + 1];
} PAGE;
```

Estrutura do algoritmo de busca por chave

FUNCTION: search (RRN, /* posição relativa (no arquivo da árvore-B) da página a ser pesquisada
KEY, /* chave sendo procurada
FOUND_RRN /* posição relativa(no arquivo da árvore-B) da página que contém a chave
FOUND_POS) /* posição da chave na página (no vetor KEYS)

if RRN == NIL **then**
return NOT FOUND
else
read página na posição relativa dada por RRN em PAGE
procura por KEY em PAGE, POS := posição no vetor de chaves em que KEY aparece (ou apareceria)
if KEY foi encontrada **then**
FOUND_RRN := RRN /* esse é o RRN da página corrente, que contém a chave
FOUND_POS := POS
return FOUND
else
return (search(PAGE.CHILD[POS], KEY, FOUND_RRN, FOUND_POS))
endif
endif
end FUNCTION

Estrutura do algoritmo de inserção de chave (sem redistribuição de chaves entre páginas irmãs)

Insere chave em KEY em árvore-B mantida em disco.
A tentativa de inserção começa verificando o conteúdo da página cuja posição relativa no arquivo é dada por CURRENT_RRN.
Se essa página não é uma página folha, a função chama a si própria recursivamente, até que encontre KEY em alguma página ou até que chegue a um nó folha.
Se KEY é encontrada, a função termina e retorna um valor de erro (ERROR).
Se há espaço para inserir KEY na página PAGE , ela é inserida. Senão, PAGE vai ser subdividida (chamada ao procedimento split).
O procedimento de subdivisão (split) vai colocar na variável PROMO_KEY a chave do meio, e a posição relativa no arquivo da nova página recém-criada na variável PROMO_R_CHILD, de modo que o processo de inserção continue nos níveis superiores da árvore, à medida em que a recursão vai `subindo´ de volta.
Quando ocorre promoção de uma chave para o nível de cima, isso é indicado pelo retorno de PROMOTION, caso contrário, isso é indicado pelo retorno de NO_PROMOTION

FUNCTION: insert (CURRENT_RRN, KEY, PROMO_R_CHILD, PROMO_KEY)

```
if CURRENT_RRN ==  NIL then
        PROMO_KEY  :=  KEY
        PROMO_R_CHILD := NIL
        return PROMOTION
else
        read página dada por CURRENT_RRN em PAGE
        procura por KEY em PAGE
        POS := posição no vetor de chaves em que KEY aparece (ou apareceria)
if KEY encontrada then
        emite mensagem de erro indicando ocorrência de chave duplicada
        return ERROR

RETURN_VALUE :=  insert(PAGE.CHILD[POS], KEY, P_B_RRN, P_B_KEY)

if RETURN_VALUE == NO_PROMOTION or ERROR then
        return RETURN_VALUE
elseif  tem espaço para P_B_KEY em PAGE then
        insere P_B_KEY e P_B_RRN nas posições devidas dos vetores KEY e CHILD em PAGE
        return NO_PROMOTION
else
        split(P_B_KEY, P_B_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)
        write PAGE no arquivo da árvore-B no RRN dado por CURRENT_RRN
        write NEWPAGE no arquivo da árvore-B no RRN dado por PROMO_R_CHILD
        return PROMOTION
endif
```

Estrutura do algoritmo de subdivisão de página

Um procedimento de inserção de I_KEY e I_RRN provocou overflow na página PAGE. A subdivisão cria uma nova página (NEWPAGE), redistribui as chaves entre PAGEe NEWPAGE, e determina qual chave e filho direito correspondente devem ser promovidos para o nível de cima. A chave promovida é retornada na variável PROMO_KEY, e seu filho direito é retornado na variável PROMO_R_CHILD.

PROCEDURE: split (I_KEY, I_RRN, PAGE, PROMO_KEY, PROMO_R_CHILD, NEWPAGE)

copia todas as chaves e ponteiros de PAGE em uma página de trabalho que consegue acomodar uma chave e um ponteiro extra

insere I_KEY e I_RRN nos seus locais apropriados na página de trabalho

aloca e inicializa uma nova página (registro)no arquivo da árvore-B para acomodar NEWPAGE

PROMO_KEY:= valor da chave do meio, que vai ser promovida após a subdivisão

PROMO_R_CHILD:= RRN de NEWPAGE

Copia as chaves e os respectivos ponteiros para filhos que precedem PROMO_KEY da página de trabalho para PAGE

Copia as chaves e os respectivos ponteiros para filhos que sucedem PROMO_KEY da página de trabalho para NEWPAGE

End PROCEDURE

Estrutura do programa que aciona os demais

MAIN PROCEDURE : driver

if arquivo da árvore- B já existe **then**

open B-tree file

else cria arquivo da árvore-B e insere a primeira chave na página raiz

KEY:= chave a ser inserida

ROOT := RRN da página raiz (obtém do arquivo)

while existem chaves a serem inseridas

if (insert (ROOT, KEY, PROMO_R_CHILD, PROMO_KEY) == PROMOTION) **then**

cria nova página raiz com key := PROMO_KEY, left child := ROOT e right child := PROMO_R_CHILD

ROOT:= RRN da nova página raiz

endif

KEY: = próxima chave a ser inserida

endwhile

atualiza RRN do nó raiz no arquivo da árvore-B

close arquivo da árvore-B

end MAIN PROCEDURE