```
proc insert (nodepointer, entry, newchildentry)
// Inserts entry into subtree with root '*nodepointer'; degree is d;
// 'newchildentry' is null initially, and null upon return unless child is split

if *nodepointer is a non-leaf node, say N,
    find i such that K_i ≤ entry's key value < K_{i+1};          // choose subtree
    insert(P_i, entry, newchildentry);                  // recursively, insert entry
    if newchildentry is null, return;                  // usual case; didn't split child
    else,                          // we split child, must insert *newchildentry in N
        if N has space,                                                  // usual case
            put *newchildentry on it, set newchildentry to null, return;
        else,                              // note difference wrt splitting of leaf page!
            split N:                  // 2d + 1 key values and 2d + 2 nodepointers
            first d key values and d + 1 nodepointers stay,
            last d keys and d + 1 pointers move to new node, N2;
            // *newchildentry set to guide searches between N and N2
            newchildentry = & (⟨smallest key value on N2, pointer to N2⟩);
            if N is the root,                                  // root node was just split
                create new node with ⟨pointer to N, *newchildentry⟩;
                make the tree's root-node pointer point to the new node;
            return;

if *nodepointer is a leaf node, say L,
    if L has space,                                                      // usual case
    put entry on it, set newchildentry to null, and return;
    else,                                          // once in a while, the leaf is full
        split L: first d entries stay, rest move to brand new node L2;
        newchildentry = & (⟨smallest key value on L2, pointer to L2⟩);
        set sibling pointers in L and L2;
        return;
endproc
```

Figure 9.11 Algorithm for Insertion into B+ Tree of Order d