1 \hspace{1em} \textbf{B-Trees, Definition}

- Definition. Suppose \( m \) is a positive integer. Then a *B-tree of order \( m \)* is a tree with the following properties.

1. Each nonempty node other than the root has between \( \lceil \frac{m}{2} \rceil \) and \( m \) children.
2. If the tree is nonempty, the root has between 2 and \( m \) children.
3. If a node has \( c \) children then it stores \( c - 1 \) keys in increasing order.
4. If the keys stored in a node are
   \[ k_1 < k_2 < \cdots < k_s, \]
   then
   (a) all keys stored in leftmost or 0th subtree are less than \( k_1 \),
   (b) all keys stored in the \( i \)th subtree, \( 1 \leq i < s \), lie between \( k_i \) and \( k_{i+1} \),
   (c) and all keys stored in the last subtree are greater than than \( k_s \).
5. All the leaves have the same depth.
/ insert
search current node for key;
if (found) {
    print message;
    return null;
} else if (current node is a leaf) {
    insert key into current node;
} else {
    recursively insert into appropriate child of current node, possibly returning a new key and a new child;
    if (return is not null) {
        insert new key into current node;
        set reference to new child;
    }
}

if (current node has too many children) {
    split the current node, creating a new key and a new child;
    return the key and child;
} else {
    return null;
}
// remove
search current node for key;
if (found) {
    if (current node is a leaf) {
        delete key from current node;
    } else {
        temp == curr node;
        find immediate predecessor of key (changing current node);
        replace key in temp with predecessor;
        delete immediate predecessor from new current node;
    }
} else if (current node is a leaf) {
    print message;
    return noMerge;
} else {
    recursively delete key from appropriate child of current node,
    passing in "split key" and returning merge info;
    if (children were merged)
        delete split key;
}

if (current node has too few keys) {
    if (left sibling has extra key) {
        shuffle keys;
        return noMerge;
    } else if (right sibling has extra key) {
        shuffle keys;
        return noMerge;
    } else {
        merge current node, split key, and sibling;
        return merge;
    }
else
    return noMerge;