B-Trees

- B-tree of ***order d*** is a tree with these properties:
  - Internal nodes have one more child (pointer) than data elements (keys). Leaf nodes have no children.
  - Root has between 1 and 2d data elements.
  - Non-root nodes have between d and 2d elements.
  - All leaves are at the same depth in the tree.
  - Has ***extended search property*** (binary search tree property extended to multiway tree)

- Search algorithm
  - Extrapolated from binary tree search algorithm.

- Insert algorithm
  - First, find ***leaf*** node where data would go.
  - Insert(data, node):
  - If data can fit in node, add it to the node.
  - If causes overflow:
  - split node at the median value.
  - Everything less than median becomes new leaf node.
  - Everything greater than median becomes new leaf node.
  - Promote median to parent node; call insert(median, parent) [*may create new parent node if there is no parent*]

- Search for item to delete
  - If at leaf node, delete the item
  - Rebalance up from leaf if necessary
  - If at internal node, swap with largest child in left sub-tree (analogous to BST deletion swap)
  - Rebalance if necessary

Deletion in a B-tree

Deletion from a leaf node

1. Search for the value to delete.
2. If the value is in a leaf node, simply delete it from the node.
3. If underflow happens, rebalance the tree as described in section "Rebalancing after deletion" below.

Deletion from an internal node

1. Choose a new separator (the *largest element in the left subtree*), remove it from the leaf node it is in, and replace the element to be deleted with the new separator.
2. The previous step deleted an element (the new separator) from a leaf node. If that leaf node is now deficient (has fewer than the required number of nodes), then rebalance the tree starting from the leaf node.

Rebalancing after deletion

Rebalancing starts from a leaf and proceeds toward the root until the tree is balanced. If deleting an element from a node has brought it under the minimum size, then some elements must be redistributed to bring all nodes up to the minimum. Usually, the redistribution involves moving an element from a sibling node that has more than the minimum number of nodes. That redistribution operation is called a **rotation**. If no sibling can spare an element, then the deficient node must be **merged** with a sibling. The merge causes the parent to lose a separator element, so the parent may become deficient and need rebalancing. The merging and rebalancing may continue all the way to the root. Since the minimum element count doesn't apply to the root, making the root be the only deficient node is not a problem. The algorithm to rebalance the tree is as follows:

- If the deficient node's right sibling exists and has more than the minimum number of elements, then ***rotate left****:*
    1. Copy the separator from the parent to the end of the deficient node (the separator moves down; the deficient node now has the minimum number of elements)
    2. Replace the separator in the parent with the first element of the right sibling (right sibling loses one node but still has at least the minimum number of elements)
    3. The tree is now balanced

- Otherwise, if the deficient node's left sibling exists and has more than the minimum number of elements, then ***rotate right****:*
    1. Copy the separator from the parent to the start of the deficient node (the separator moves down; deficient node now has the minimum number of elements)
    2. Replace the separator in the parent with the last element of the left sibling (left sibling loses one node but still has at least the minimum number of elements)
    3. The tree is now balanced

- Otherwise, if both immediate siblings have only the minimum number of elements, then ***merge*** with a sibling (left or right). A new node will be created that combines the items from the deficient node, the sibling node, and the "separator" item from the parent node that is between the two siblings.
    1. Copy the separator to the end of the left node (the left node may be the deficient node or it may be the sibling with the minimum number of elements)
    2. Move all elements from the right node to the left node (the left node now has the maximum number of elements, and the right node is empty)
    3. Remove the separator from the parent along with its empty right child (the parent loses an element)
        - If the parent is the root and now has no elements, then free it and make the merged node the new root (tree becomes shallower)
        - Otherwise, if the parent has fewer than the required number of elements, then rebalance the parent