|  |
| --- |
| Introduction to B-Trees |
| A B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. Unlike self-balancing binary search trees, it is optimized for systems that read and write large blocks of data. It is most commonly used in database and file systems. |
| The B-Tree Rules |
| Important properties of a B-tree:   * B-tree nodes have many more than two children. * A B-tree node may contain more than just a single element.   The set formulation of the B-tree rules: Every B-tree depends on a positive constant integer called MINIMUM, which is used to determine how many elements are held in a single node.   * Rule 1: The root can have as few as one element (or even no elements if it also has no children); every other node has at least MINIMUM elements. * Rule 2: The maximum number of elements in a node is twice the value of MINIMUM. * Rule 3: The elements of each B-tree node are stored in a partially filled array, sorted from the smallest element (at index 0) to the largest element (at the final used position of the array). * Rule 4: The number of subtrees below a nonleaf node is always one more than the number of elements in the node.   + Subtree 0, subtree 1, ... * Rule 5: For any nonleaf node:   + An element at index *i* is greater than all the elements in subtree number *i* of the node, and   + An element at index *i* is less than all the elements in subtree number *i* + 1 of the node. * Rule 6: Every leaf in a B-tree has the same depth. Thus it ensures that a B-tree avoids  the problem of a unbalanced tree.  |  |  | | --- | --- | | https://www.cpp.edu/~ftang/courses/CS241/notes/images/trees/b-tree1.bmp   A binary search tree has *one* value in each node and *two* subtrees. This notion easily generalizes to an M-way search tree, which has (M-1) values per node and M subtrees. M is called the *degree* of the tree. A binary search tree, therefore, has degree 2.  In fact, it is not necessary for every node to contain exactly (M-1) values and have exactly M subtrees. In an M-way subtree a node can have anywhere from 1 to (M-1) values, and the number of (non-empty) subtrees can range from 0 (for a leaf) to 1+(the number of values). M is thus a *fixed upper limit* on how much data can be stored in a node.  The values in a node are stored in ascending order, V1 < V2 < ... Vk (k <= M-1) and the subtrees are placed between adjacent values, with one additional subtree at each end. We can thus associate with each value a `left' and `right' subtree, with the right subtree of Vi being the same as the left subtree of V(i+1). All the values in V1's left subtree are less than V1 ; all the values in Vk's subtree are greater than Vk; and all the values in the subtree between V(i) and V(i+1) are greater than V(i) and less than V(i+1).  For example, here is a 3-way search tree:  https://webdocs.cs.ualberta.ca/~holte/T26/Lecture10Fig1.gif  In our examples it will be convenient to illustrate M-way trees using a small value of M. But bear in mind that, in practice, M is usually very large. Each node corresponds to a physical block on disk, and M represents the maximum number of data items that can be stored in a single block. M is maximized in order to speedup processing: to move from one node to another involves reading a block from disk - a very slow operation compared to moving around a data structure stored in memory.  The algorithm for searching for a value in an M-way search tree is the obvious generalization of the algorithm for searching in a binary search tree. If we are searching for value X are and currently at node consisting of values V1...Vk, there are four possible cases that can arise:   1. If X < V1, recursively search for X in V1's left subtree. 2. If X > Vk, recursively search for X in Vk's right subtree. 3. If X=Vi, for some i, then we are done (X has been found). 4. the only remaining possibility is that, for some i, Vi < X < V(i+1). In this case recursively search for X in the subtree that is in between Vi and V(i+1).   A B-tree is an M-way search tree with two special properties:   1. It is perfectly balanced: every leaf node is at the same depth. 2. Every node, except perhaps the root, is at least half-full, i.e. contains M/2 or more values (of course, it cannot contain more than M-1 values). The root may have any number of values (1 to M-1).   [The 3-way search tree above](https://webdocs.cs.ualberta.ca/~holte/T26/m-way-trees.html#l10f1) is clearly *not* a B-tree. Here is a 3-way B-tree containing the same values:  https://webdocs.cs.ualberta.ca/~holte/T26/Lecture10Fig2.gif  And here is a 5-way B-tree (each node other than the root must contain between 2 and 4 values):  https://webdocs.cs.ualberta.ca/~holte/T26/Lecture10Fig3.gif  In the descriptions of our algorithms, we assume that M is odd; therefore each node (other than the root) must contains between (M-1)/2 and M-1 values. | https://www.cpp.edu/~ftang/courses/CS241/notes/images/trees/b-tree2.bmp | |