# Binary Search Tree Description

A binary search tree is organized, as the name suggests, in a binary tree. We can represent such a tree by a linked data structure in which each node is an object. In addition to a key and satellite data, each node contains attributes left, right ,and p that point to the nodes corresponding to its left child, its right child, and its parent, respectively. If a child or the parent is missing, the appropriate attribute contains the value NIL. The root node is the only node in the tree whose parent is NIL.

The keys in a binary search tree are always stored in such a way as to satisfy the

binary-search-tree property:

Let x be a node in a binary search tree. If y is a node in the left subtree of x, then y.key <= x.key. If y is a node in the right subtree of x, then y.key => x.key.

Searching

We use the following procedure to search for a node w ith a given key in a binary

search tree. Given a pointer to the root of the tree and a key k, TREE-SEARCH returns a pointer to a node with key k if one exists; otherwise, it returns NIL.

The procedure begins its search at the root and traces a simple path downward in

the tree, as shown in Figure 12.2. For each node x it encounters, it compares the key

k with x.key .If the two keys are equal, the search terminates. If k is smaller than x.key , the search continues in the left subtree of x, since the binary-search-tree property implies that k

could not be stored in the right subtree. Symmetrically, if k is larger than x.key , the search continues in the right subtree. The nodes encountered during the recursion form a simple path downward from the root of the tree, and thus the running time of TREE-SEARCH is O(h)

, where h is the height of the tree.

We can rew rite this procedure in an iterative fashion by “unrolling” the recursion into a

While loop. On most computers, the iterative version is more efficient.

Minimum and maximum

We can always find an element in a binary search tree whose key is a minimum by

Following left child pointers from the root until we encounter a NIL. If a node x has no left subtree, then since every key in the right subtree of x is at least as large as x.key, the minimum key in the subtree rooted at x is x.key. If node x has a left subtree, then since no key in the right subtree is smaller than x.key and every key in the left subtree is not larger than x.key, the minimum key in the subtree rooted at x resides in the subtree rooted at x.left. Similar logic follows for maximum and the right most child.