6. Search and Decision Trees

6.1. Binary Tree.

DEFINITION 6.1.1. *A* **binary tree** *is a rooted 2-ary tree. In a binary tree a child of a parent may be designated as a* **left child** *or a* **right child**, *but each parent has at most one of each.*

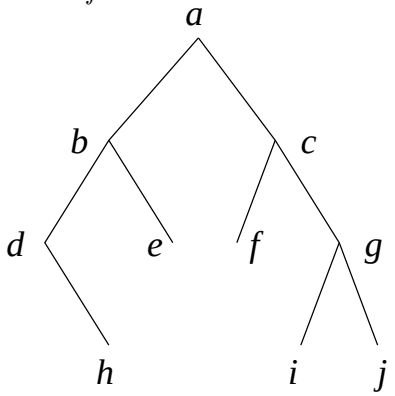
The rooted subtree consisting of the right (left) child of a vertex and all of its descendants is the **right** *(***left***)* **subtree** *at that vertex.*

A binary tree is often called a **binary search tree**.

Discussion

A binary tree is a rooted tree in which we may impose additional structure; namely, the designation of each child of a vertex as either a left or right child. This designation may be fairly arbitrary in general, although it may be natural in a particular application. A tree could be drawn with the left and right subtrees of a vertex reversed, so it may be unclear which is the left and right without a sketch of the tree or a clear designation from the beginning. If a tree has been sketched in the plane with the root at the top and the children of a vertex below the vertex, then the designation of left or right child should be inferred naturally from the sketch.

EXERCISE 6.1.1. *For the binary tree below, identify left and right children and sketch the left and right subtrees of each vertex other than the leaves.*



The tree shown above could have been drawn with the vertices b and c (and their subtrees) reversed. The resulting tree would be isomorphic to the original as rooted trees, but the isomorphism would not preserve the additional structure. The point is that the left and right descendents are not preserved under an isomorphism of rooted trees.

Searching items in a list can often be accomplished with the aid of a binary search tree. For example, suppose we are given a list of elements, $X_1, X_2, ..., X_n$, from an ordered set $(S, <)$, but the elements are not necessarily listed according to their preferred ordering. We can establish a recursive procedure for constructing a binary search tree with vertices labeled or *keyed* by the elements of the list as demonstrated by Example 6.2.1. This tree will allow us to search efficiently for any particular item in the list.

6.2. Example 6.2.1.

EXAMPLE 6.2.1. *Suppose $X_1, X_2, ..., X_n$ are elements from an ordered set $(S, <)$. Form a binary search tree recursively as follows.*

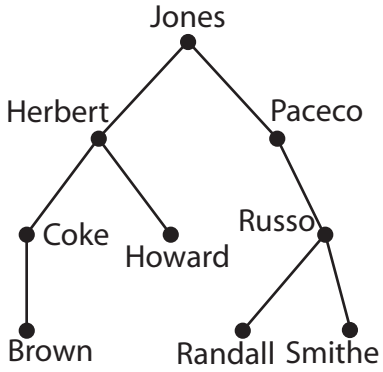
(1) Basis: Let X_1 be the label or key for the root.
(2) Recursion: Assume we have constructed a binary search tree with vertices keyed by $X_1, ..., X_i$, $1 \leq i < n$. Starting with the root, keyed X_1, compare X_{i+1} with the keys of the vertices already in the tree, moving to the left if the vertex key is greater than X_{i+1} and to the right otherwise. We eventually reach a leaf with key X_j for some j between 1 and i. We add a vertex with key X_{i+1} and edge (X_j, X_{i+1}) and designate X_{i+1} to be either a left child of X_j if $X_{i+1} < X_j$ or a right child of X_j if $X_j < X_{i+1}$.

Discussion

The main characteristic of the binary search tree constructed is that the key of any vertex is greater than the key of any vertex in its left subtree and is less than the key of any vertex in its right subtree.

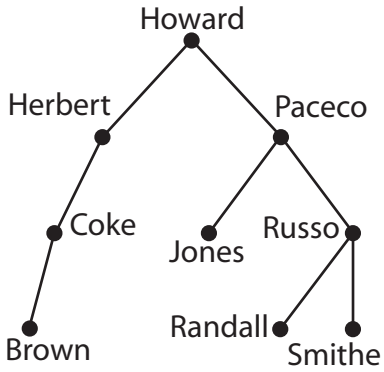
EXAMPLE 6.2.2. *Construct a binary search tree with vertices keyed by the names in the list {Jones, Paceco, Hebert, Howard, Russo, Coke, Brown, Smithe, Randall} using alphabetical order.*

Solution:



Discussion

Example 6.2.2 is one in which we have assigned the vertices keys from a list of names, ordered alphabetically. Notice that if we rearrange the list of names, say {Howard, Paceco, Jones, Hebert, Russo, Coke, Brown, Randall, Smithe}, we might get a different tree:



6.3. Decision Tree.

Definition 6.3.1. *A* **decision tree** *is a rooted tree in which each internal vertex corresponds to a decision and the leaves correspond to the possible outcomes determined by a sequence of decisions (a path).*

Discussion

Decision trees may be used to determine the complexity of a problem or an algorithm. Notice that a decision tree need *not* be a binary tree.

6.4. Example 6.4.1.

Example 6.4.1 (Has been featured as a puzzler on *Car Talk*). *Suppose you are given a collection of identical looking coins and told one of them is counterfeit. The counterfeit coin does not weigh the same as the real coins. You may or may not be told the counterfeit coin is heavier or lighter. The only tool you have to determine which is counterfeit is a balance scale. What is the fewest number of weighings required to find the counterfeit coin? Your answer does depend on the number of coins you are given and whether or not you are told the counterfeit is heavier or lighter than the rest.*

Solution

The solution will be obtained by a sequence of weighings as follows:

- Choose two subsets of the coins of equal number and compare them on the balance.
- There are three possibilities: one of the two sets of coins weighs more than, less than, or is the same as the other set of coins.
- Depending on the outcome of the first weighing, choose another two subsets of the coins and compare them.
- This problem can be modeled with a ternary (3-ary) tree where an internal vertex corresponds to a weighing, and edge corresponds to an outcome of that weighing, and a leaf corresponds to a coin found to be counterfeit.

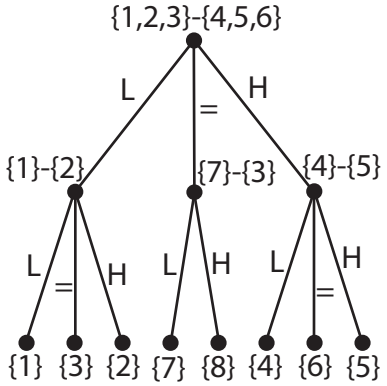
In order to determine the minimal number of weighings for a given problem, you must be clever in choosing the sets of coins to weigh in order not to require any redundant comparisons.

Discussion

Just think, you could have won a t-shirt from *Car Talk* if you had known about this!

EXAMPLE 6.4.2. *Suppose there are 8 coins,* $\{1, 2, 3, 4, 5, 6, 7, 8\}$*, and you know the counterfeit coin is lighter than the rest.*

Solution: Use a ternary tree to indicate the possible weighings and their outcomes. A vertex labeled with the notation $\{a, ..., b\} - \{x, ..., y\}$ *stands for the act of comparing the set of coins* $\{a, ..., b\}$ *to* $\{x, ..., y\}$*. An edge from that vertex will have one of the labels* L*,* $=$*, or* H*, depending on whether the first of the two sets is lighter than, equal in weight to, or heavier than the second set.*



With careful choices of the sets we see that only two weighings are necessary.

Notice there is a leaf for every possible outcome, that is, one for each coin. It is not difficult, but perhaps tedious, to see that we cannot get by with only one weighing. You would have to argue cases. This problem was made a bit easier since we knew the counterfeit coin is lighter.

EXERCISE 6.4.1. *In the example above, how many weighings are necessary if you don't know whether the counterfeit is lighter or heaver? [Notice that in this case there would be 16 leaves.]*