CSc 3102: Deleting from a Binary Search Tree

Supplementary Notes

- Kinds of Node Deletion
- Deleting A Leaf Node
- Deleting A Node With A Single Child
- Deleting A Node With Two Children

1 Kinds of Node Deletion

The deletion operation is more involved than other binary search tree operations. The deletion operation involves three cases:

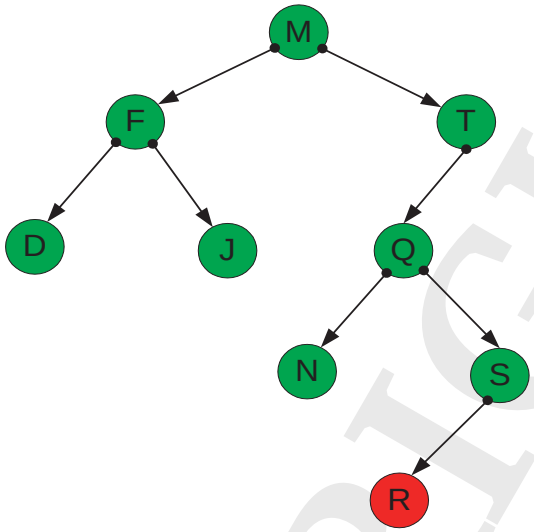
- Deleting a leaf node - pretty straightforward.
- Deleting a node that has one child - child is promoted.
- Deleting a node with two children - well, this involves a bit more work.

2 Deleting A Leaf Node

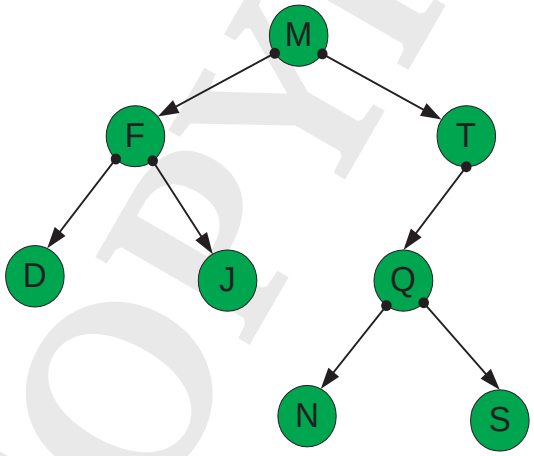
Deletion of a leaf node, a node without children, is pretty straightforward. We simply free the node: that is, once there is a pointer to the node, free memory associated with the node. See Figure 1.

COPYRIGHT

Figure 1: Leaf Node Deletion in a Binary Search Tree



Before Deleting Leaf Node R



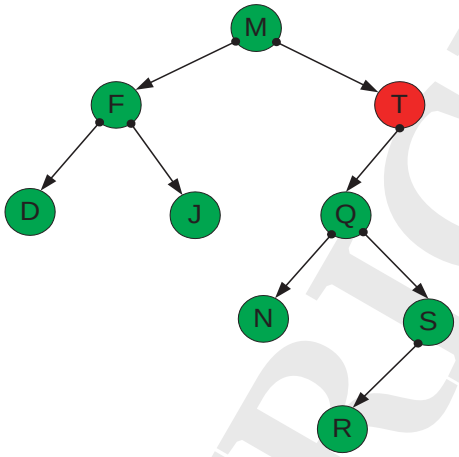
After Deleting Leaf Node R

COPYRIGHT

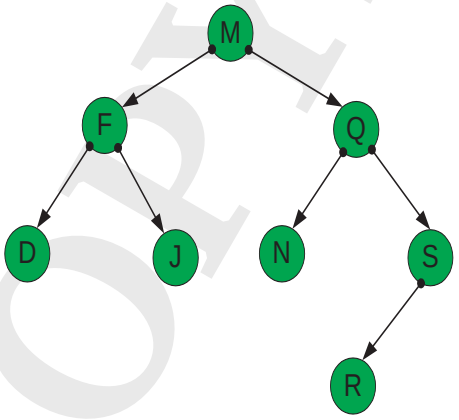
3 Deleting A Half Node

Deletion of a half node, a node with one child, simply involves promoting the child of the node to its position. A temporary pointer is set to the node and the pointer to the node is set to point to the child. The memory associated with the node is then deallocated via the temporary pointer. See figure 2.

Figure 2: Half Node Deletion in a Binary Search Tree



Before Deleting Half Node T



After Deleting Half Node T

COPYRIGHT

4 Deleting a Full Node

Deletion of a full node, a node with two children, X involves the following steps:

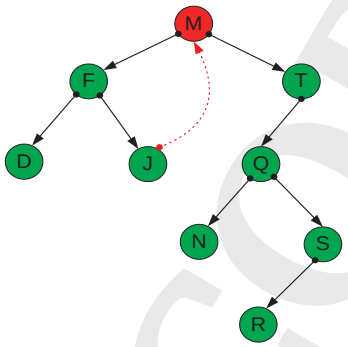
- Locate another node Y that is easier to delete from the tree than the node X.
- Copy the key and data of node Y into node X, thus effectively deleting node X.
- Deallocate memory previously associated with node Y.

Here you have two options to preserve the property of the binary search tree that requires the keys of every node to the right of a node come after the key of the node and the keys of every node to the left of a node come before the key of the node in the ordering used to construct the tree.

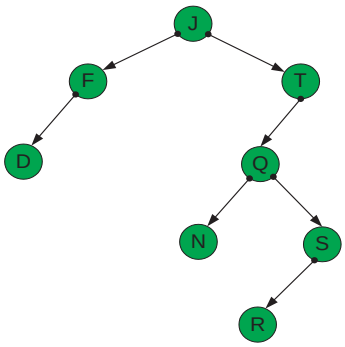
1. X may be replaced by the right-most node in its left-child's subtree - see Figure 3, or
2. X may be replaced by the left-most node in its right-child's subtree - see Figure 4.

See Figures 3 and 4 for illustrations of both approaches.

Figure 3: Full Node Deletion Using In-order Predecessor Replacement



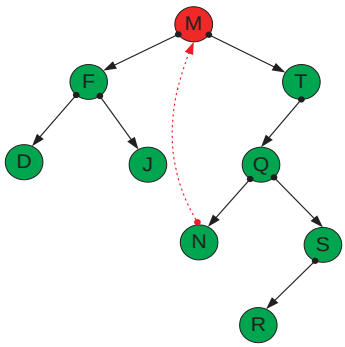
Before Deleting Full Node M



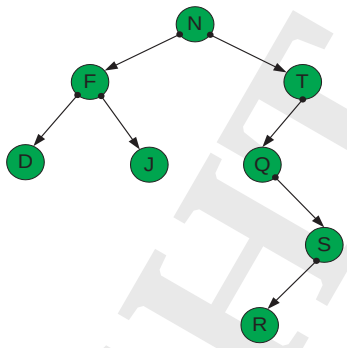
After Deleting Full Node M

COPYRIGHT

Figure 4: Full Node Deletion Using In-order Successor Replacement



Before Deleting Full Node M



After Deleting Full Node M

Problem 1. Show the contents of the initially empty binary search tree b before and after each deletion, assuming inorder predecessor replacement strategy.

1. b.insert(92);
2. b.insert(24);
3. b.insert(6);
4. b.insert(7);
5. b.insert(11);
6. b.insert(8);
7. b.insert(22);
8. b.delete(24);
9. b.insert(4);
10. b.insert(5);
11. b.insert(16);
12. b.delete(92);
13. b.insert(19);
14. b.insert(20);
15. b.insert(78);
16. b.delete(6);

Problem 2. Repeat Problem 1 assuming inorder successor replacement strategy.

COPYRIGHT