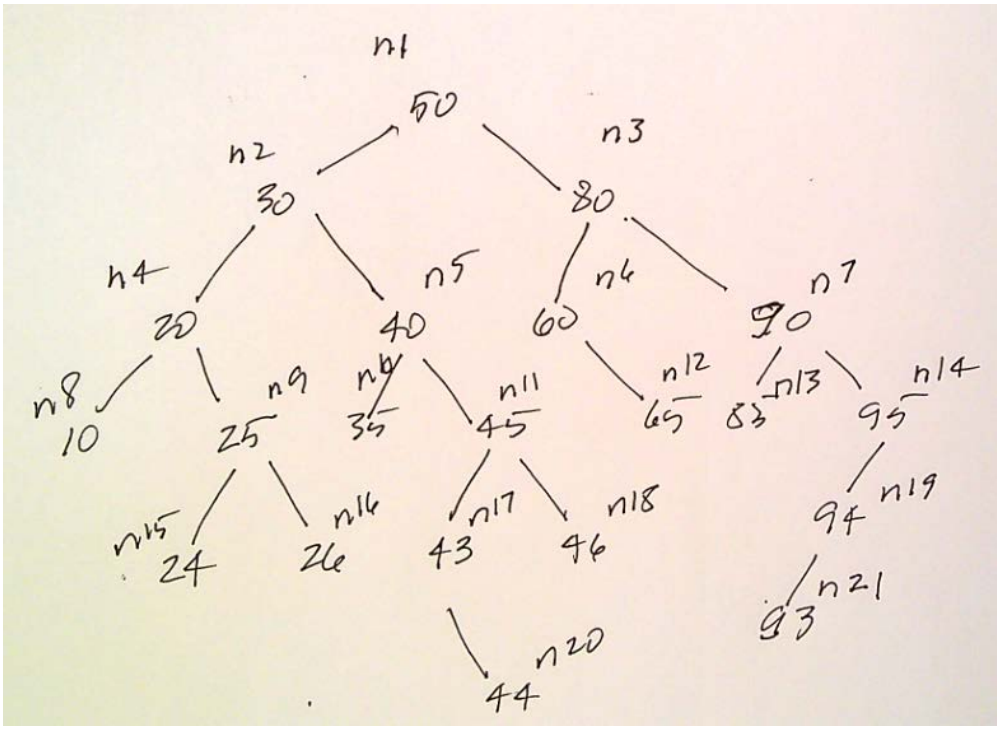
BST Delete in Racket

The following is a Racket source to help organize your code to implement the deletion operation. A few of the helper functions are implemented.

We can use a manually constructed tree to help test and debug. Here is a snapshot of the tree. Each node is bound to a different symbol. The tree is then built without using the insert or add value function. So the tree can be used to test other operations even if insert is not implemented.



The following Racket source defines a few of the required helper functions for BST delete as well as a starter version of delete. The tree pictured here is then constructed with define statements that simply link all the nodes together. You should confirm that your insertion method can build the tree, but that is not covered here.

Functions for inorder traversal, path from root to node, and a more detailed function to display nodes with some structure info per node, are provided.

```
#lang racket

(struct bst-node (value left right count) #:mutable #:transparent)

(define (display-node node)
  (display (format "[~a(" (bst-node-value node)))
  (cond
    ((empty? (bst-node-left node)) (display "N,"))
    (else (display (format "~a," (bst-node-value (bst-node-left node))))))
  (cond
    ((empty? (bst-node-right node)) (display "N)]"))
    (else (display (format "~a)]" (bst-node-value (bst-node-right node))))))
  )

(define (display-nodes-inorder n)
  (cond
    ((empty? n) (void))
    ((and (empty? (bst-node-left n)) (empty? (bst-node-right n))) (display-node n))
    ((empty? (bst-node-left n)) (display-node n) (display-nodes-inorder (bst-node-right n)))
    ((empty? (bst-node-right n)) (display-nodes-inorder (bst-node-left n)) (display-node n))
    (else (display-nodes-inorder (bst-node-left n)) (display-node n) (display-nodes-inorder (bst-
node-right n)))
    ))

(define (inorder n)
  (cond
    ((empty? n) (void))
    ((and (empty? (bst-node-left n)) (empty? (bst-node-right n))) (list (bst-node-value n)))
    ((empty? (bst-node-left n)) (append (list (bst-node-value n)) (inorder (bst-node-right n))))
    ((empty? (bst-node-right n)) (append (inorder (bst-node-left n)) (list (bst-node-value n))))
    (else (append (inorder (bst-node-left n)) (list (bst-node-value n)) (inorder (bst-node-right
n))))
    ))

(define (find-path v n) (find-path-1 v n empty))

(define (find-path-1 v n p)
  (cond
    ((null? n) p)
    ((= v (bst-node-value n)) (cons n p))
    ((< v (bst-node-value n)) (cons n (find-path-1 v (bst-node-left n) p)))
    ((> v (bst-node-value n)) (cons n (find-path-1 v (bst-node-right n) p)))
    ))

(define (get-path-values p)
  (cond
    ((null? p) empty)
    (else (cons (bst-node-value (car p)) (get-path-values (cdr p))))
    ))

(define (get-child-count tree)
  (cond
    ((null? tree) 0)
    ((and (null? (bst-node-left tree)) (null? (bst-node-right tree))) 0)
    ((null? (bst-node-left tree)) 1)
    ((null? (bst-node-right tree)) 1)
    (else 2)
    ))
```

```
; entry point for delete
; calls helper function
; uses find-path to find path from root to node to be deleted
; reverses path so that path is nl, node to delete is (car nl), parent is (car (cdr nl))

(define (delete-value-from-bst v tree) (delete-node-from-bst (reverse (find-path v tree)) tree))

; helper function for delete
(define (delete-node-from-bst nl tree)
  (cond
    ((empty? nl) tree)
    ((> (bst-node-count (car nl)) 1) (set-bst-node-count! (car nl) (- (bst-node-count (car nl))
1)) tree)
    ((= (get-child-count (car nl)) 0)
     (cond
       ((equal? (car nl) tree) empty)
       ((equal? (car nl) (bst-node-left (car (cdr nl)))) (set-bst-node-left! (car (cdr nl))
empty) tree)
       (else (set-bst-node-right! (car (cdr nl)) empty) tree)
       ))
    (else tree)))
```

```
; ===================================================
; test cases
; ===================================================

(define n21 (bst-node 93 empty empty 1))
(define n20 (bst-node 44 empty empty 1))
(define n19 (bst-node 94 n21 empty 1))
(define n18 (bst-node 46 empty empty 1))
(define n17 (bst-node 43 empty n20 1))
(define n16 (bst-node 26 empty empty 1))
(define n15 (bst-node 24 empty empty 1))
(define n14 (bst-node 95 n19 empty 1))
(define n13 (bst-node 83 empty empty 1))
(define n12 (bst-node 65 empty empty 1))
(define n11 (bst-node 45 n17 n18 1))
(define n10 (bst-node 35 empty empty 1))
(define n9  (bst-node 25 n15 n16 1))
(define n8  (bst-node 10 empty empty 1))
(define n7  (bst-node 90 n13 n14 1))
(define n6  (bst-node 60 empty n12 1))
(define n5  (bst-node 40 n10 n11 1))
(define n4  (bst-node 20 n8 n9 1))
(define n3  (bst-node 80 n6 n7 1))
(define n2  (bst-node 30 n4 n5 1))
(define n1  (bst-node 50 n2 n3 1))

(inorder n1) ; show list of values of nodes in tree rooted at n1
(display-nodes-inorder n1) ; like inorder, but for each node display value of left and right
child nodes
(displayln "") ; extra newline

; use find-path to get path of nodes from root to node with value 44 starting from n1
(define p44 (find-path 44 n1))
(get-path-values p44)

(define n1-1 (delete-value-from-bst 44 n1))
(inorder n1-1)
(display-nodes-inorder n1-1)
(displayln "")
```