درج:

برای درج ما به موقعیت مناسب گره نیاز داریم و نگه داری ساکسسور یا والد تاثیری برای ما ندارد و از رویه زیر استفاده میکنیم:

function insert(root, value):

if root is null:

return new Node(value)

current = root

while current is not null:

if value < current.value:

if current.left is null:

current.left = new Node(value)

return root

current = current.left

else:

if current.right is null:

current.right = new Node(value)

return root

current = current.right

سرچ:

در سرچ اگر گره مورد نظر ما نباشد در پارت بعدی جست و جو به سراغ succ میرویم و رویه آن به شکل زیر است:

function search(root, value):

current = root

while current is not null:

if value == current.value:

return current

else if value < current.value:

current = current.left

else:

current = current.right

return null // Not found

function find\_successor(node):

if node.right is not null:

return find\_min(node.right)

else:

current = node.parent

while current is not null and node == current.right:

node = current

current = current.parent

return current

function find\_min(node):

while node.left is not null:

node = node.left

return node

حذف:

برای حذف کردن نود ها اما با داشتن ساکسسور مجبور میشویم که در بدترین حالت کل درخت را پیمایش کنیم اما در صورتی که با استفاده از یک رویه میانی ابتدا والد هر گره را پیدا کنیم سپس میتوانیم با استفاده از والد در زمان o(h) عملیات حذف را هم انجام دهیم

پیدا کردن والد خود در زمان o(h) شدنی است

پس از ان حذف نیز با همین مرتبه زمانی قابل انجام است و مرتبه زمانی کل ما o(h) خواهد ماند

function delete(root, value):

parent = null

current = root

// Find the node to delete and its parent

while current is not null and current.value != value:

parent = current

if value < current.value:

current = current.left

else:

current = current.right

if current is null:

return root // Node not found

// Case 1: Node has no children (Leaf node)

if current.left is null and current.right is null:

if parent is null:

root = null // Deleting the root node

else if parent.left == current:

parent.left = null

else:

parent.right = null

// Case 2: Node has one child

else if current.left is null:

if parent is null:

root = current.right

else if parent.left == current:

parent.left = current.right

else:

parent.right = current.right

else if current.right is null:

if parent is null:

root = current.left

else if parent.left == current:

parent.left = current.left

else:

parent.right = current.left

// Case 3: Node has two children

else:

// Find the successor (smallest node in right subtree)

successor = find\_min(current.right)

// Replace current value with successor's value

current.value = successor.value

// Delete the successor node

if successor.parent.left == successor:

successor.parent.left = successor.right

else:

successor.parent.right = successor.right

return root