RHION,RHU,RHEYA,SHYLO,MYRON,INES,HEATH,SYBIL,SHYAN,OKKES,YAZEN

Diagram

Description automatically generated

Adding last word (YAZEN):

Let ptr = Root node (‘R’). ‘Y’>’R’ and ‘R’.right != None. Point ptr to ‘R’.right (‘S’)

‘Y’>’S’ but ‘S’.right’ is None. Create a node ‘Y’ with mult 0 and point ‘S’.right to it.

‘Y’.mid is None. Just create node ‘A’ with mult 0 and point ‘Y’.mid to it

‘A’.mid is None. Just create node ‘Z’ with mult 0 and point ‘A’.mid to it

‘Z’.mid is None. Just create node ‘E’ with mult 0 and point ‘Z’.mid to it

‘E’.mid is None. Just create node ‘N’ and point ‘E’.mid to it. Because this is the final char of this word, mult = 1

class Node:

def \_\_init\_\_(self, d, n):

self.data = d

self.next = n

class LinkedList:

def \_\_init\_\_(self):

self.head = None

self.length = 0

def \_\_str\_\_(self):

if self.head == None:

return "empty"

st = "-"

ptr = self.head

while ptr != None:

st += "-> "+str(ptr.data)+" "

ptr = ptr.next

return st+"|"

def search(self, d):

i = 0

ptr = self.head

while ptr != None:

if ptr.data == d:

return i

ptr = ptr.next

i += 1

return -1

def search2(self, d): # recursive implementation

def searchRec(ptr,d,i):

if ptr == None: return -1

if prt.data == d: return i

return searchRec(ptr.next,d,i+1)

return searchRec(self.head,d,0)

def append(self, d):

if self.head == None:

self.head = Node(d,None)

else:

ptr = self.head

while ptr.next != None:

ptr = ptr.next

ptr.next = Node(d,None)

self.length += 1

def insert(self, i, d):

if self.head == None or i == 0:

self.head = Node(d,self.head)

else:

ptr = self.head

while i>1 and ptr.next != None:

ptr = ptr.next

i -= 1

ptr.next = Node(d,ptr.next)

self.length += 1

def remove(self, i): # removes i-th element and returns it

if self.head == None:

return None

if i == 0:

val = self.head.data

self.head = self.head.next

self.length -= 1

return val

ptr = self.head

while ptr.next != None:

if i == 1:

val = ptr.next.data

ptr.next = ptr.next.next

self.length -= 1

return val

ptr = ptr.next

i -= 1

def removeVal(self, d):

if self.head == None:

return -1

if self.head.data == d:

self.head = self.head.next

self.length -= 1

return 0

else:

i = 0

ptr = self.head

while ptr.next != None:

if ptr.next.data == d:

ptr.next = ptr.next.next

self.length -= 1

return i+1

ptr = ptr.next

i += 1

return -1

def sublist(self, i):

ptr = self.head

ls = LinkedList()

ls.length = self.length

while ptr != None and i>0:

ptr = ptr.next

i -= 1

ls.length -= 1

ls.head = ptr

return ls

def get(self, i):

ptr = self.head

while i > 0:

ptr = ptr.next

i -= 1

return ptr.data

def set(self, i, d):

ptr = self.head

while i > 0:

ptr = ptr.next

i -= 1

ptr.data = d

class Stack:

def \_\_init\_\_(self):

self.inList = LinkedList()

def \_\_str\_\_(self):

return str(self.inList)

def \_\_repr\_\_(self):

return f'{"|".join(self.inList.get(i)[0].data for i in range(self.size()))}'

def size(self):

return self.inList.length

def push(self, e):

self.inList.insert(0,e)

def pop(self):

return self.inList.remove(0)

class BgNode:

def \_\_init\_\_(self,d,l,r,n):

self.data = d

self.left = l

self.right = r

self.mid = n

self.mult = 0

# prints the node and all its children in a string

def \_\_str\_\_(self):

st = "("+str(self.data)+", "+str(self.mult)+") -> ["

if self.left != None:

st += str(self.left)

else: st += "None"

if self.mid != None:

st += ", "+str(self.mid)

else: st += ", None"

if self.right != None:

st += ", "+str(self.right)

else: st += ", None"

return st + "]"

def \_\_repr\_\_(self):

return f"({self.data}, {self.mult}) -> [{self.left.data if self.left else 'None'}, {self.mid.data if self.mid else 'None'}, {self.right.data if self.right else 'None'}]"

class BigTree:

def \_\_init\_\_(self):

self.root = None

self.size = 0

def \_\_str\_\_(self):

return str(self.root)

def add(self,st):

if st == "":

return None

if self.root == None:

self.root = BgNode(st[0],None,None,None)

ptr = self.root

for i in range(len(st)):

d = st[i]

while True:

if d == ptr.data:

break

elif d < ptr.data:

if ptr.left == None:

ptr.left = BgNode(d,None,None,None)

ptr = ptr.left

else:

if ptr.right == None:

ptr.right = BgNode(d,None,None,None)

ptr = ptr.right

if i < len(st)-1 and ptr.mid == None:

ptr.mid = BgNode(st[i+1],None,None,None)

if i < len(st)-1:

ptr = ptr.mid

ptr.mult += 1

self.size += 1

def addAll(self,A):

for x in A: self.add(x)

def printAll(self):

def printFrom(ptr,s):

if ptr == None: return

s0 = s + ptr.data

if ptr.left != None: printFrom(ptr.left,s)

for i in range(ptr.mult): print(s0, end=" ")

if ptr.mid != None: printFrom(ptr.mid,s+ptr.data)

if ptr.right != None: printFrom(ptr.right,s)

printFrom(self.root,""); print()

# ---- RHION,RHU,RHEYA,SHYLO,MYRON,INES,HEATH,SYBIL,SHYAN,OKKES,YAZEN

def count(self,st):

def search(ptr, char):

"""Returns the child node with data as 'char' from ptr node, if it exists."""

while ptr:

if ptr.data == char:

return ptr

if char < ptr.data and ptr.left:

ptr = ptr.left

elif char > ptr.data and ptr.right:

ptr = ptr.right

else:

return

if self.root and st:

ptr = self.root

i = 0

while i < len(st)-1:

ptr = search(ptr, st[i])

if ptr and ptr.mid:

i += 1

ptr = ptr.mid

else:

return 0

ptr = search(ptr, st[i])

if ptr:

return ptr.mult

return 0

def toIncArray(self):

if not self.root:

return []

a, i = [0 for \_ in range(self.size)], 0

s, w = Stack(), self.root.data

s.push([self.root, 0])

while s.size():

n, check = s.inList.get(0)

if check == 0 and n.left:

s.inList.get(0)[1] = 1

s.push([n.left, 0])

w = w[:-1] + n.left.data

continue

if check == 1:

w += n.data

if check <= 1:

for m in range(n.mult):

a[i] = w

i += 1

if n.mid:

s.inList.get(0)[1] = 2

s.push([n.mid, 0])

w += n.mid.data

continue

s.pop()

w = w[:-1]

if n.right:

s.push([n.right, 0])

w += n.right.data

return a

def remove(self,st):

def search(ptr, char, node, d):

"""Returns the child node with data as 'char' from ptr node, if it exists."""

while ptr:

i = 0

if ptr.mid:

if (ptr.mid.left or ptr.mid.right) and ptr.mid: # ptr.mid.mid

node = ptr

d = 1

if ptr.data == char:

return ptr, node, d

if char < ptr.data and ptr.left:

ptr = ptr.left

d = 0

elif char > ptr.data and ptr.right:

ptr = ptr.right

d = 2

else:

return None, node, d

if self.root and st:

ptr = self.root

branch\_node, branch\_dir = self.root, 1

i = 0

while i < len(st)-1:

ptr, branch\_node, branch\_dir = search(ptr, st[i], branch\_node, branch\_dir)

if ptr and ptr.mid:

i += 1

ptr = ptr.mid

else:

return 0

ptr, branch\_node, branch\_dir = search(ptr, st[i], branch\_node, branch\_dir)

if ptr:

if ptr.mult > 1:

ptr.mult -= 1

elif ptr.mid:#ptr.left or ptr.right:

if ptr.mult == 1:

ptr.mult = 0

else:

return False

else:

if branch\_dir == 0:

if branch\_node.left.left and not branch\_node.left.right:

branch\_node.left = branch\_node.left.left

elif branch\_node.left.right and not branch\_node.left.left:

branch\_node.left = branch\_node.left.right

elif branch\_dir == 1:

if branch\_node.mid.left and not branch\_node.mid.right:

branch\_node.mid = branch\_node.mid.left

elif branch\_node.mid.right and not branch\_node.mid.left:

branch\_node.mid = branch\_node.mid.right

else:

if branch\_node.right.left and not branch\_node.right.right:

branch\_node.right = branch\_node.right.left

elif branch\_node.mid.right and not branch\_node.mid.left:

branch\_node.right = branch\_node.right.right

self.size -= 1

return True

return False