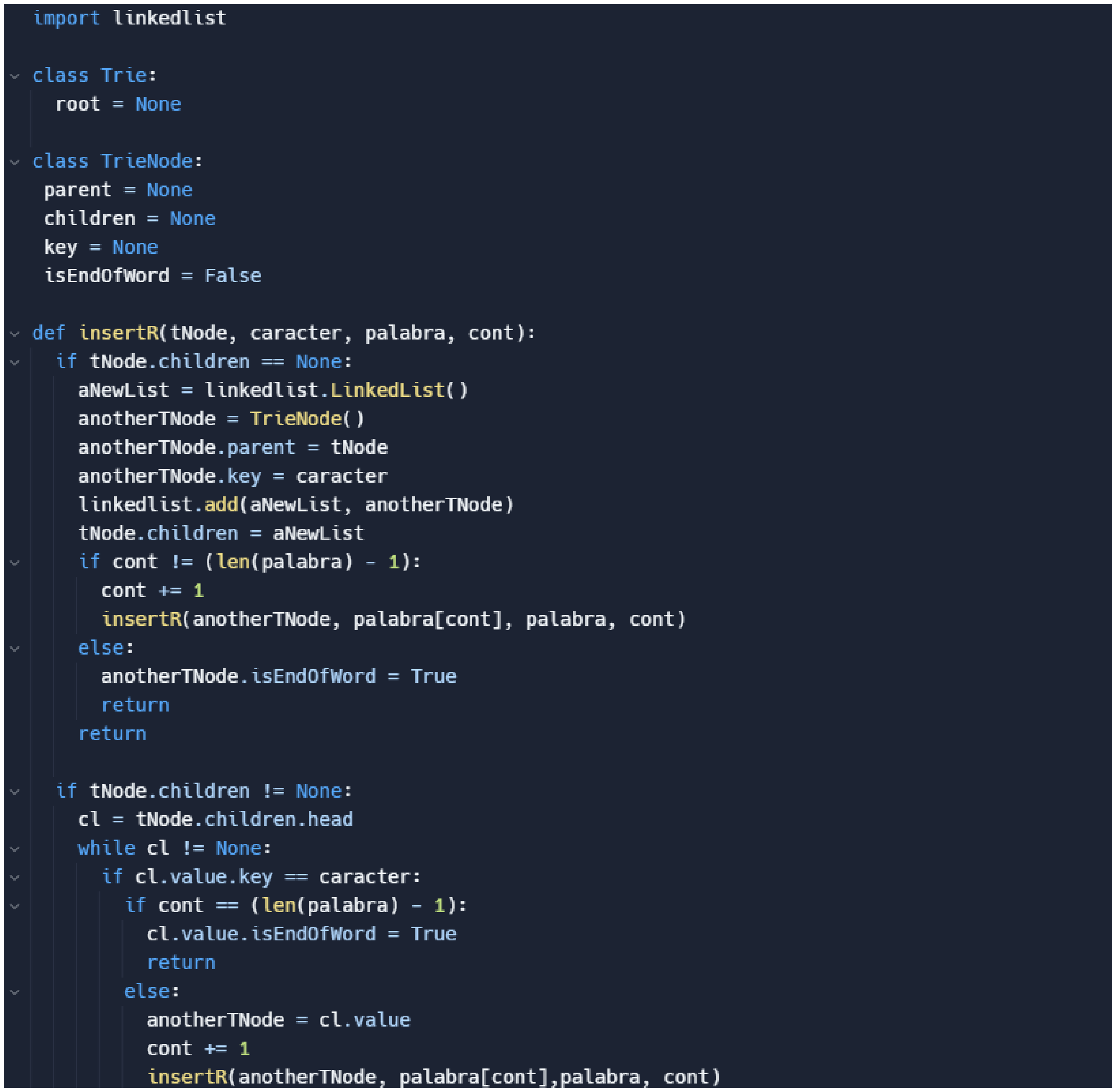
ALGORITMO Y ESTRUCTURA DE DATOS II

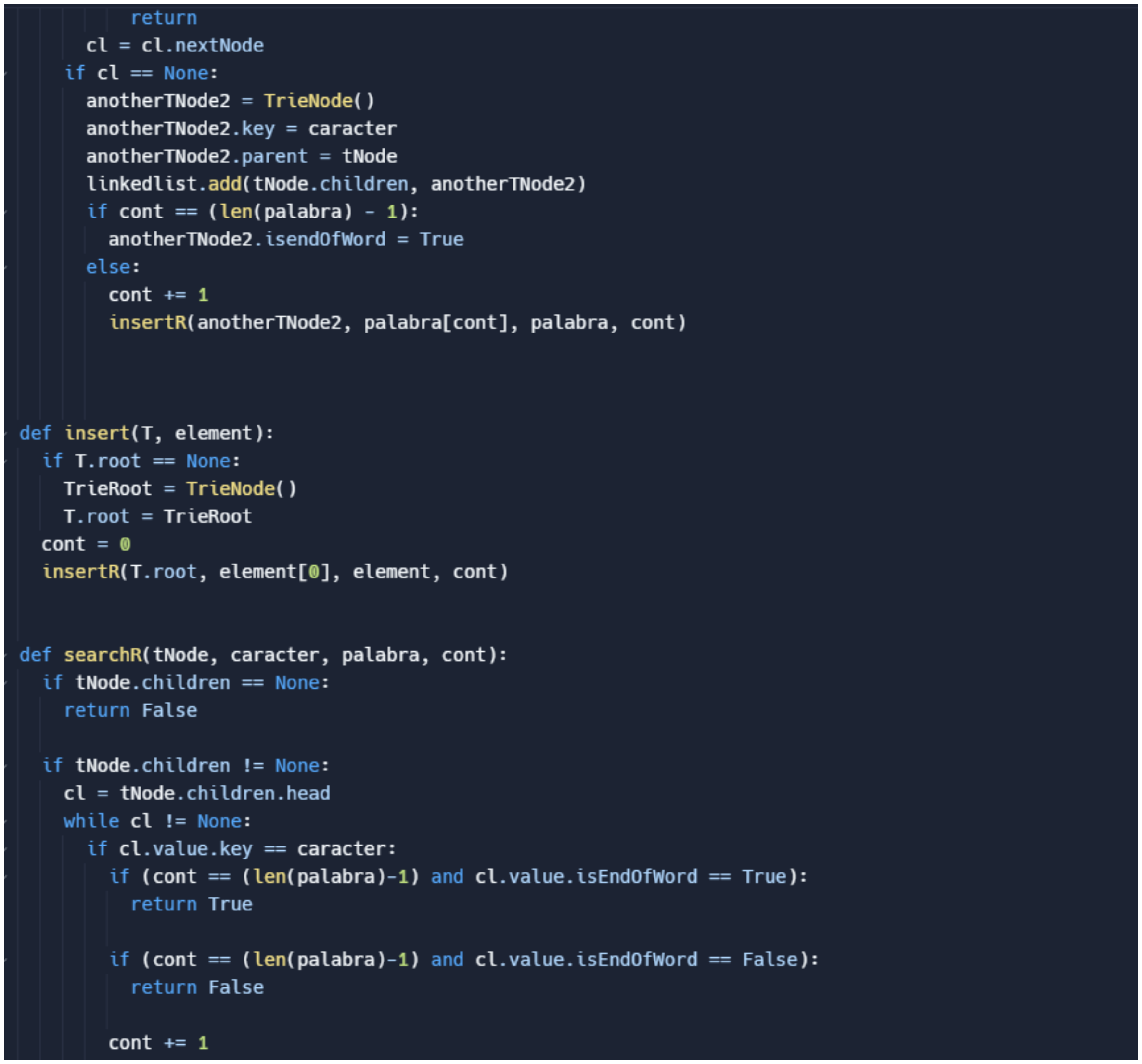
TP TRIE   Estudiante: Joaquin Villegas

PARTE 1:

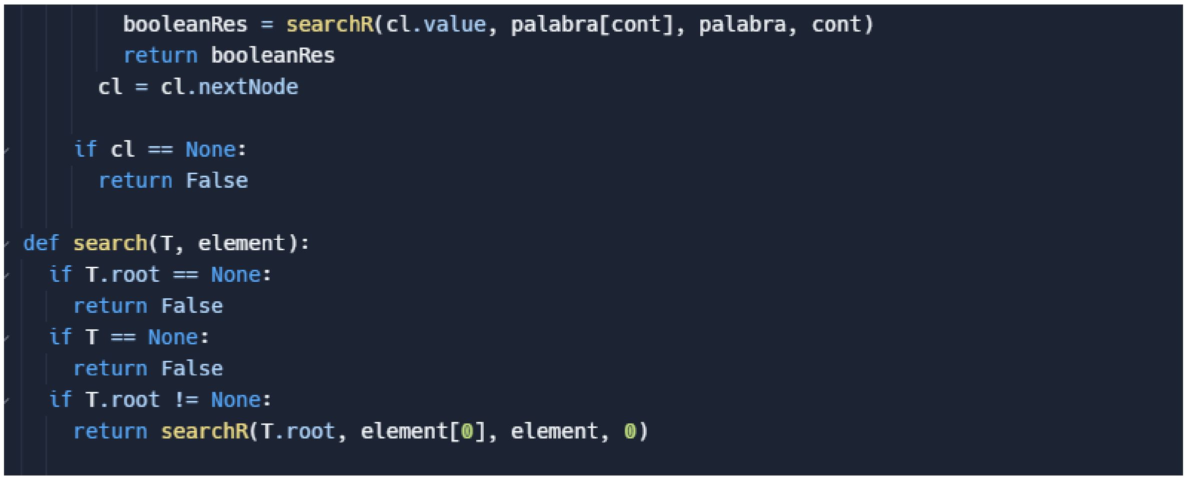
1-



```
import linkedlist

class Trie:
  root = None

class TrieNode:
 parent = None
 children = None
 key = None
 isEndOfWord = False

def insertR(tNode, caracter, palabra, cont):
  if tNode.children == None:
    aNewList = linkedlist.LinkedList()
    anotherTNode = TrieNode()
    anotherTNode.parent = tNode
    anotherTNode.key = caracter
    linkedlist.add(aNewList, anotherTNode)
    tNode.children = aNewList
    if cont != (len(palabra) - 1):
      cont += 1
      insertR(anotherTNode, palabra[cont], palabra, cont)
    else:
      anotherTNode.isEndOfWord = True
      return
    return

  if tNode.children != None:
    cl = tNode.children.head
    while cl != None:
      if cl.value.key == caracter:
        if cont == (len(palabra) - 1):
          cl.value.isEndOfWord = True
          return
        else:
          anotherTNode = cl.value
          cont += 1
          insertR(anotherTNode, palabra[cont],palabra, cont)
```



```python
                    return
            cl = cl.nextNode
        if cl == None:
            anotherTNode2 = TrieNode()
            anotherTNode2.key = caracter
            anotherTNode2.parent = tNode
            linkedlist.add(tNode.children, anotherTNode2)
            if cont == (len(palabra) - 1):
                anotherTNode2.isendOfWord = True
            else:
                cont += 1
                insertR(anotherTNode2, palabra[cont], palabra, cont)



def insert(T, element):
    if T.root == None:
        TrieRoot = TrieNode()
        T.root = TrieRoot
    cont = 0
    insertR(T.root, element[0], element, cont)


def searchR(tNode, caracter, palabra, cont):
    if tNode.children == None:
        return False

    if tNode.children != None:
        cl = tNode.children.head
        while cl != None:
            if cl.value.key == caracter:
                if (cont == (len(palabra)-1) and cl.value.isEndOfWord == True):
                    return True

                if (cont == (len(palabra)-1) and cl.value.isEndOfWord == False):
                    return False

                cont += 1
```

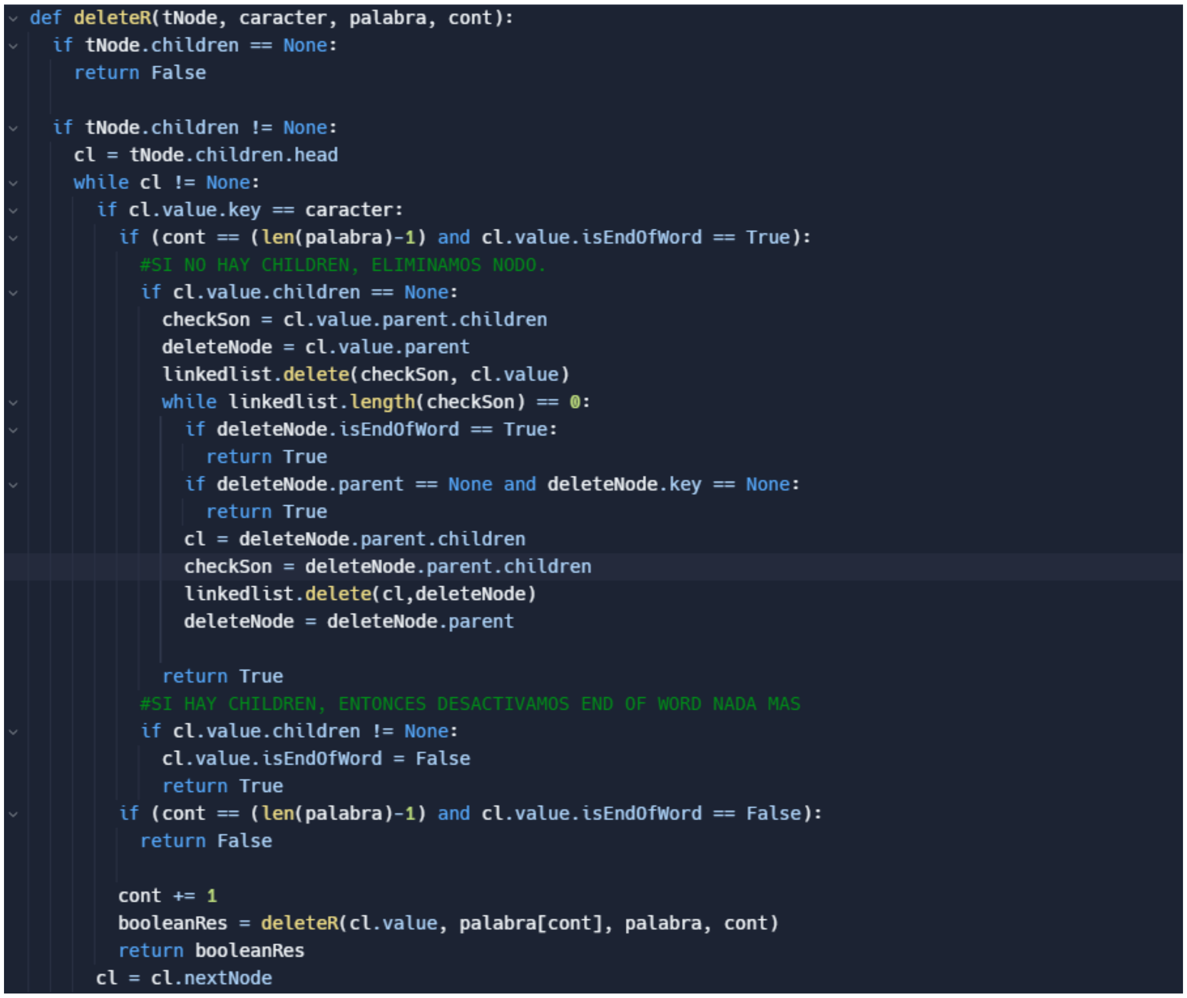


```python
                booleanRes = searchR(cl.value, palabra[cont], palabra, cont)
                return booleanRes
            cl = cl.nextNode

        if cl == None:
            return False

def search(T, element):
    if T.root == None:
        return False
    if T == None:
        return False
    if T.root != None:
        return searchR(T.root, element[0], element, 0)
```

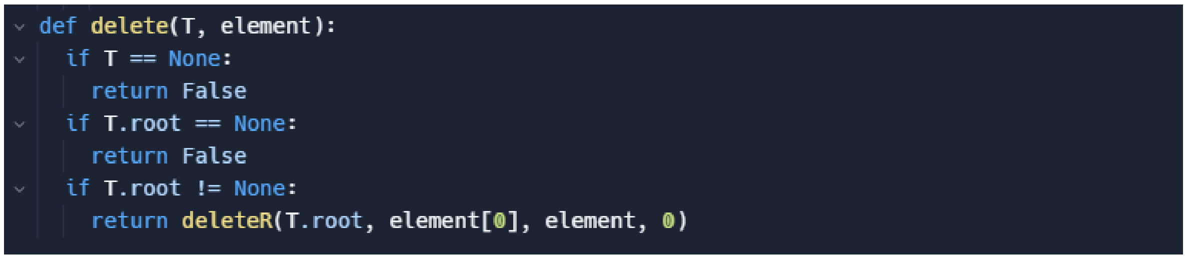
2-

Una versión para que la complejidad del search sea de O(m), es utilizando arrays, debido a que cuando busquemos por carácter sabemos que se logra en O(1). Entonces, para realizar el search completo nos tomaría la longitud de la palabra.

3-



```
def deleteR(tNode, caracter, palabra, cont):
  if tNode.children == None:
    return False

  if tNode.children != None:
    cl = tNode.children.head
    while cl != None:
      if cl.value.key == caracter:
        if (cont == (len(palabra)-1) and cl.value.isEndOfWord == True):
          #SI NO HAY CHILDREN, ELIMINAMOS NODO.
          if cl.value.children == None:
            checkSon = cl.value.parent.children
            deleteNode = cl.value.parent
            linkedlist.delete(checkSon, cl.value)
            while linkedlist.length(checkSon) == 0:
              if deleteNode.isEndOfWord == True:
                return True
              if deleteNode.parent == None and deleteNode.key == None:
                return True
              cl = deleteNode.parent.children
              checkSon = deleteNode.parent.children
              linkedlist.delete(cl,deleteNode)
              deleteNode = deleteNode.parent

            return True
          #SI HAY CHILDREN, ENTONCES DESACTIVAMOS END OF WORD NADA MAS
          if cl.value.children != None:
            cl.value.isEndOfWord = False
            return True
        if (cont == (len(palabra)-1) and cl.value.isEndOfWord == False):
          return False

        cont += 1
        booleanRes = deleteR(cl.value, palabra[cont], palabra, cont)
        return booleanRes
      cl = cl.nextNode
```



```
def delete(T, element):
  if T == None:
    return False
  if T.root == None:
    return False
  if T.root != None:
    return deleteR(T.root, element[0], element, 0)
```

https://replit.com/@CorexoPro/TrieImplementacion