Martins Ezequiel 14000

Trabajo Práctico Arboles AVL (U2)

https://replit.com/@EzeMarts/tp-trie#trie.py

Ejercicio 1

```
def insert(T, element):
  if element == "":
    return None
  element = element.upper() # Hacer mayuscula
  if T.root == None:
    rootNode = TrieNode()
    T.root = rootNode
    newNode = TrieNode()
    newNode.key = element[0:1] # Darle de key la 1ra letra
    element = element[1: len(element)] # Cortar la cadena
    newList = []
    newList.append(newNode)
    rootNode.isEndOfWord = True
    rootNode.children = newList
    newNode.parent = rootNode
    if element == "":
      newNode.isEndOfWord = True
      return
    lowerList = []
    newNode.children = lowerList
    insertR(lowerList, element, newNode)
  else:
    insertR(T.root.children, element, T.root.children[0])
```

```
def insertR(L, element, parentNode):
  i = 0
  while i < len(L) and L[i].key != element[0]:
    i =+ 1
  if i == len(L):
    newNode = TrieNode()
    newNode.parent = parentNode # Darle el nodo superior como parent
    newNode.key = element[0:1] # Darle de key la 1ra letra
    L.append(newNode)
    lowerList = []
    newNode.children = lowerList
    element = element[1: len(element)] # Cortar la cadena
    if element == "":
      newNode.isEndOfWord = True
      return
    else:
      insertR(lowerList, element, newNode)
  else:
    element = element[1: len(element)] # Cortar la cadena
    if element == "":
      L[i].isEndOfWord = True
      return
    else:
      insertR(L[i].children, element, L[i])
```

```python
def search(T, element):
  if T.root == None or element == "":
    return None
  else:
    element = element.upper()
    return searchR(T.root.children, element)
def searchR(L, element):
  i = 0
  while i < len(L) and L[i].key != element[0]:
    i =+ 1
  if i == len(L):
    return False
  else:
    element = element[1: len(element)] # Cortar la cadena
    if element == "" and L[i].isEndOfWord:
      return True
    elif element == "" and L[i].isEndOfWord == False:
      return False
    else:
      return searchR(L[i].children, element)
```

Ejercicio 2

Ejercicio 3

```
def delete(T, element): # Optimizar
  if T.root == None or element == "":
    return None
  else:
    element = element.upper()
    if searchR(T.root.children, element):
      deleteR(T.root.children, element)
      return True
    else:
      return False
def deleteR(L, element): # Optimizar
  i = 0
  while i < len(L) and L[i].key != element[0]:
    i =+ 1
  if i == len(L):
    return False
  else:
    element = element[1: len(element)] # Cortar la cadena
    if element == "" and L[i].isEndOfWord:
      if len(L[i].children) > 0:
        L[i].isEndOfWord = False
        return True
      return deleteUnlink(L[i].parent, L[i])
    elif element == "" and L[i].isEndOfWord == False:
      return False
    else:
      return deleteR(L[i].children, element)
def deleteUnlink(node, lowNode):
  if node.isEndOfWord:
    if len(node.children) > 1:
      node.children.remove(lowNode)
      return True
    else:
      node.children = None
      return True
  elif len(node.children) > 1:
    node.children.remove(lowNode)
    return True
  else:
    return deleteUnlink(node.parent, node)
```

Ejercicio 4

```
def patronTrie(T, letter, long): # Esta planteado, error con el .index y .count
  if T.root == None or letter == "" or long == 0:
    return None
  L = []
  letter = letter.upper() # Hacer mayuscula
  if T.root.children.count(letter) > 0:
    i = T.root.children.index(letter)
    patronTrieR(T.root.children[i], L, letter, 2, long)
    print(L)
  else:
    return None
def patronTrieR(nodeL, L, word, level, longLevel):
  if level == longLevel:
    for i in range(0, len(nodeL)):
      if nodeL[i].isEndOfWord:
        wordTemp = word + nodeL[i].key
        L.append(wordTemp)
        return
  else:
    for i in range(0, len(nodeL)):
      wordTemp = word + nodeL[i].key
      if nodeL[i].children != None:
        patronTrieR(nodeL[i].children, L, wordTemp, level+1, longLevel)
```

Ejercicio 5

Ejercicio 6

```
def capicuaTrie(T, word):
  wordCapicua = ""
  wordTemp = word
  while len(wordTemp) > 0:
    wordCapicua = wordTemp[0:1] + wordCapicua
    wordTemp = wordTemp[1:len(wordTemp)]
  print(wordCapicua)
  valid1 = search(T, word)
  valid2 = search(T, wordCapicua)
  if valid1 and valid2:
    return True
  else:
    return False
```

Ejercicio 7