Giovanni Azurduy

1)

```python
#Ejercicio 1

def searchTrieNode(linkedlist:link.LinkedList,element):
    """Busca la key de un elemento TrieNode dentro de la linkedlist y devuelve el trieNode, en caso contrario retorna None"""
    if linkedlist == None:
        return None

    currentNode = linkedlist.head
    while currentNode != None:
        if currentNode.value.key == element:
            return currentNode.value
        currentNode = currentNode.nextNode

    return None

def insertWrapper(linked_list,father,element):
    if linked_list is None:
        L = link.LinkedList()
        newTrieNode = TrieNode(element)
        link.add(L,newTrieNode)
        father.children = L
        return newTrieNode
    else:
        searchNode = searchTrieNode(linked_list,element)

        if searchNode == None:
            newTrieNode = TrieNode(element)
            link.add(linked_list,newTrieNode)
            return newTrieNode
        else:
            return searchNode
```

```python
def insert(T:Trie,word):
    """Inserta un palabra dentro del TrieNode"""
    if T.root is None:
        element = word[0]
        L = link.LinkedList()
        newTrieNode = TrieNode(element)
        link.add(L,newTrieNode)
        T.root = L
        currentNode = newTrieNode
        length_word = len(word)

        for i in range(1,length_word):
            currentNode = insertWrapper(currentNode.children,currentNode,word[i])
            if i == length_word -1:
                currentNode.isEndOfWord = True
    else:
        currentNode = T
        currentChildren = T.root
        length_word = len(word)
        for i in range(length_word):
            currentNode = insertWrapper(currentChildren,currentNode,word[i])
            if i == length_word -1:
                currentNode.isEndOfWord = True
            else:
                currentChildren = currentNode.children

def search(trie,palabra):
    """Verifica si se encuentra una palabra dentro del Trie Node"""
    if trie == None:
        return
    currentNode = trie.root
    for i in range(len(palabra)):
        searchNode = searchTrieNode(currentNode,palabra[i])
        if i == len(palabra)-1:
            if searchNode.isEndOfWord == False:
                return False
        if searchNode == None:
            return False
        currentNode = searchNode.children

    return True
```

2)

```python
#Ejercicio 2
"""
Si tenemos un Trie contruido con arrays y le asignamos un valor a cada letra podriamos acceder a la letra en Tiempo O(1) gracias a que accedemos directamente
al indicie y asi sucesivamente el unico problema con esto es que necesitas tener el alfabeto completo creado y asi para cada hijo dando una
complejidad espacial O(n^n) y complejidad temporal O(1)
"""
```

3)

```python
#Ejercicio 3
def delete(trie,word):
    """Elimina un elemento que se encuentre dentro del Trie, Retorna True si lo logra o falso en caso contrario"""
    pila = stack.Stack()
    currentNode = trie
    currentChildren = currentNode.root
    for i in range(len(word)):
        currentNode = searchTrieNode(currentChildren,word[i])
        if currentNode == None:
            return False
        else:
            stack.push(pila,currentNode)
            currentChildren = currentNode.children

    endNode = stack.pop(pila)

    if endNode.isEndOfWord == True:
        if endNode.children == None:
            currentNode = stack.pop(pila)
            while currentNode != None:
                link.delete(currentNode.children,endNode)
                if currentNode.isEndOfWord == True:
                    break
                endNode = currentNode
                currentNode = stack.pop(pila)
                if currentNode == None:
                    link.delete(trie.root,endNode)
        else:
            endNode.isEndOfWord = False
    else:
        return False
    return True

```

4)

```python
#Ejercicio 4
def patronTrieWrapper(trie,pattern,n,list):
    if len(pattern) == n:
        if trie.isEndOfWord:
            list.append(pattern)
            return
    if trie == None:
        return
    linkedlist = trie.children
    if linkedlist == None:
        return
    currentNode = linkedlist.head
    while currentNode != None:
        patronTrieWrapper(currentNode.value,pattern + currentNode.value.key,n,list)
        currentNode = currentNode.nextNode

def patronTrie(trie,pattern,n):
    """Algoritmo que dado un Trie , un patron y un entero n escribe todas las palabras del arbol que empiezan por p y sean de logintud n"""
    lista = []
    currentNode = trie
    currentChildren = currentNode.root
    for i in range(len(pattern)):
        currentNode = searchTrieNode(currentChildren,pattern[i])
        if currentNode == None:
            return None
        else:
            currentChildren = currentNode.children

    #bucle de la lista
    patronTrieWrapper(currentNode,pattern,n,lista)

    return lista
```

5)

```python
#Ejercicio 5

def extraer_palabras(trie,palabra,lista):
    if trie == None:
        return

    if isinstance(trie,Trie):
        linkedlist = trie.root
    else:
        linkedlist = trie.children
        if trie.isEndOfWord:
            lista.append(palabra)
            return

    currentNode = linkedlist.head

    while currentNode != None:
        extraer_palabras(currentNode.value,palabra+currentNode.value.key,lista)
        currentNode = currentNode.nextNode

def tries_iguales(trie_1,trie_2):
    """Dados un Trie 1 y Trie 2 devuelve True si son iguales y contienen las mismas palabras en caso contrario devuelve False"""
    lista_1 = []
    lista_2 = []

    extraer_palabras(trie_1,"",lista_1)
    extraer_palabras(trie_2,"",lista_2)

    lista_1.sort()
    lista_2.sort()

    if len(lista_1) == len(lista_2):
        for i in range(len(lista_1)):
            if lista_1[i] != lista_2[i]:
                return False
    else:
        return False
```

6)

```python
#Ejercicio 6

def invertir_palabra(word):
    palabra = ""
    for i in range(len(word)-1,-1,-1):
        palabra += word[i]
    return palabra

def cadenas_invertidas(trie):
    """Dado un Trie devuelve True si existe en el documento T dos cadenas invertidas.
    Dos cadenas son invertidas si se leen de izquierda a derecha y contiene los mismos caracteres que si se lee de derecha a izquierda"""
    lista = []
    extraer_palabras(trie,"",lista)
    for i in range(len(lista)):
        for j in range(i,len(lista)):
            if lista[i] == invertir_palabra(lista[j]):
                return True
```

7)

```python
#Ejercicio 7
def autoCompletar(trie,cadena):
    """Dado un Trie y una cadena devuelve la forma de auto-completar de la palabra. ejemplo
    autoCompletar(T, 'groen') devolvería "land", ya que podemos tener "groenlandia" o "groenlandés"
    """
    currentNode = trie
    currentChildren = currentNode.root
    palabra = ""

    for i in range(len(cadena)):
        currentNode = searchTrieNode(currentChildren,cadena[i])
        if currentNode == None:
            return palabra
        else:
            currentChildren = currentNode.children

    if currentNode == None:
        return palabra
    currentNode = currentNode.children
    if currentNode == None:
        return palabra
    currentNode = currentNode.head
    while currentNode != None:

        if currentNode.nextNode != None:
            break

        trienode = currentNode.value
        palabra += trienode.key
        list = trienode.children
        if list == None:
            return palabra
        currentNode = list.head

    return palabra
```