

Giovanni Azurduy

1)

```
16 #Ejercicio 1
17
18 def searchTrieNode(linkedlist:link.LinkedList,element):
19     """Busca la key de un elemento TrieNode dentro de la linkedlist y devuelve el trieNode, en caso contrario retorna None"""
20     if linkedlist == None:
21         return None
22
23     currentNode = linkedlist.head
24     while currentNode != None:
25         if currentNode.value.key == element:
26             return currentNode.value
27         currentNode = currentNode.nextNode
28
29     return None
30
31 def insertWrapper(linked_list,father,element):
32     if linked_list is None:
33         L = link.LinkedList()
34         newTrieNode = TrieNode(element)
35         link.add(L,newTrieNode)
36         father.children = L
37         return newTrieNode
38     else:
39         searchNode = searchTrieNode(linked_list,element)
40
41         if searchNode == None:
42             newTrieNode = TrieNode(element)
43             link.add(linked_list,newTrieNode)
44             return newTrieNode
45         else:
46             return searchNode
47
```

```
48 def insert(T:Trie,word):
49     """Inserta un palabra dentro del TrieNode"""
50     if T.root is None:
51         element = word[0]
52         L = link.LinkedList()
53         newTrieNode = TrieNode(element)
54         link.add(L,newTrieNode)
55         T.root = L
56         currentNode = newTrieNode
57         length_word = len(word)
58
59         for i in range(1,length_word):
60             currentNode = insertWrapper(currentNode.children,currentNode,word[i])
61             if i == length_word -1:
62                 currentNode.isEndOfWord = True
63     else:
64         currentNode = T
65         currentChildren = T.root
66         length_word = len(word)
67         for i in range(length_word):
68             currentNode = insertWrapper(currentChildren,currentNode,word[i])
69             if i == length_word -1:
70                 currentNode.isEndOfWord = True
71             else:
72                 currentChildren = currentNode.children
73
74 def search(trie,palabra):
75     """Verifica si se encuentra una palabra dentro del Trie Node"""
76     if trie == None:
77         return
78     currentNode = trie.root
79     for i in range(len(palabra)):
80         searchNode = searchTrieNode(currentNode,palabra[i])
81         if i == len(palabra)-1:
82             if searchNode.isEndOfWord == False:
83                 return False
84             if searchNode == None:
85                 return False
86             currentNode = searchNode.children
87
88     return True
89
```

2)

```

91 #Ejercicio 2
92 """
93 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
94 al indice y asi sucesivamente el unico problema con esto es que necesitas tener el alfabeto completo creado y asi para cada hijo dando una
95 complejidad espacial O(n^n) y complejidad temporal O(1)
96 """

```

3)

```

99 #Ejercicio 3
100 def delete(trie,word):
101     """Elimina un elemento que se encuentre dentro del Trie, Retorna True si lo logra o falso en caso contrario"""
102     pila = stack.Stack()
103     currentNode = trie
104     currentChildren = currentNode.root
105     for i in range(len(word)):
106         currentNode = searchTrieNode(currentChildren,word[i])
107         if currentNode == None:
108             return False
109         else:
110             stack.push(pila,currentNode)
111             currentChildren = currentNode.children
112
113     endNode = stack.pop(pila)
114
115     if endNode.isEndOfWord == True:
116         if endNode.children == None:
117             currentNode = stack.pop(pila)
118             while currentNode != None:
119                 link.delete(currentNode.children,endNode)
120                 if currentNode.isEndOfWord == True:
121                     break
122                 endNode = currentNode
123             currentNode = stack.pop(pila)
124             if currentNode == None:
125                 link.delete(trie.root,endNode)
126         else:
127             endNode.isEndOfWord = False
128     else:
129         return False
130     return True
131

```

4)

```

133 #Ejercicio 4
134 def patronTrieWrapper(trie,pattern,n,list):
135     if len(pattern) == n:
136         if trie.isEndOfWord:
137             list.append(pattern)
138         return
139     if trie == None:
140         return
141     linkedlist = trie.children
142     if linkedlist == None:
143         return
144     currentNode = linkedlist.head
145     while currentNode != None:
146         patronTrieWrapper(currentNode.value,pattern + currentNode.value.key,n,list)
147         currentNode = currentNode.nextNode
148
149 def patronTrie(trie,pattern,n):
150     """Algoritmo que dado un Trie , un patron y un entero n escribe todas las palabras del arbol que empiezan por p y sean de logitud n"""
151     lista = []
152     currentNode = trie
153     currentChildren = currentNode.root
154     for i in range(len(pattern)):
155         currentNode = searchTrieNode(currentChildren,pattern[i])
156         if currentNode == None:
157             return None
158         else:
159             currentChildren = currentNode.children
160
161     #bucle de la lista
162     patronTrieWrapper(currentNode,pattern,n,lista)
163
164     return lista
165

```

5)

```

168 #Ejercicio 5
169
170 def extraer_palabras(trie,palabra,lista):
171     if trie == None:
172         return
173
174     if isinstance(trie,Trie):
175         linkedlist = trie.root
176     else:
177         linkedlist = trie.children
178         if trie.isEndOfWord:
179             lista.append(palabra)
180         return
181
182     currentNode = linkedlist.head
183
184     while currentNode != None:
185         extraer_palabras(currentNode.value,palabra+currentNode.value.key,lista)
186         currentNode = currentNode.nextNode
187
188 def tries_iguales(trie_1,trie_2):
189     """Dados un Trie 1 y Trie 2 devuelve True si son iguales y contienen las mismas palabras en caso contrario devuelve False"""
190     lista_1 = []
191     lista_2 = []
192
193     extraer_palabras(trie_1,"",lista_1)
194     extraer_palabras(trie_2,"",lista_2)
195
196     lista_1.sort()
197     lista_2.sort()
198
199     if len(lista_1) == len(lista_2):
200         for i in range(len(lista_1)):
201             if lista_1[i] != lista_2[i]:
202                 return False
203     else:
204         return False
205

```

6)

```

209 #Ejercicio 6
210
211 def invertir_palabra(word):
212     palabra = ""
213     for i in range(len(word)-1,-1,-1):
214         palabra += word[i]
215     return palabra
216
217 def cadenas_invertidas(trie):
218     """Dado un Trie devuelve True si existe en el documento T dos cadenas invertidas.
219     Dos cadenas son invertidas si se leen de izquierda a derecha y contiene los mismos caracteres que si se lee de derecha a izquierda"""
220     lista = []
221     extraer_palabras(trie,"",lista)
222     for i in range(len(lista)):
223         for j in range(i,len(lista)):
224             if lista[i] == invertir_palabra(lista[j]):
225                 return True
226

```

7)

```
229 #Ejercicio 7
230 def autoComplete(trie,cadena):
231     """Dado un Trie y una cadena devuelve la forma de auto-completar de la palabra. ejemplo
232     autoComplete(T, ["groen"]) devolvería "land", ya que podemos tener "groenlandia" o "groenlandés"
233     """
234     currentNode = trie
235     currentChildren = currentNode.root
236     palabra = ""
237
238     for i in range(len(cadena)):
239         currentNode = searchTrieNode(currentChildren,cadena[i])
240         if currentNode == None:
241             return palabra
242         else:
243             currentChildren = currentNode.children
244
245     if currentNode == None:
246         return palabra
247     currentNode = currentNode.children
248     if currentNode == None:
249         return palabra
250     currentNode = currentNode.head
251     while currentNode != None:
252
253         if currentNode.nextNode != None:
254             break
255
256         trienode = currentNode.value
257         palabra += trienode.key
258         list = trienode.children
259         if list == None:
260             return palabra
261         currentNode = list.head
262
263     return palabra
```