TP Trie

Nombre : Palau Enzo

Ejercicio 1:

```
    """Ejercicio 1"""

    def insert(T,element):
        """inserta un elemento en un arbol Trie"""
        #coloco una raiz vacia si no tiene
        if T.root is None:
            NR=TrieNode()
            T.root=NR
        Nodo=T.root
        contador=1#servira para la posc de la cadena
        insertR(T,element,Nodo,contador)
```

```
def insertR(T,element,Nodo,contador):
    #segunda parte del insert
    if Nodo.children is not None:
        #verifico si en la lista del hijo coincide la letra actual
        for i in range(0,len(Nodo.children)):
            if Nodo.children[i].key is element[contador-1]:
                #caso de que recorra y el elemento este dentro del arbol sin agregar nada
                if contador==len(element):
                    Nodo.children[i].isEndOfWord = True
                    MuestraListaTrie(Nodo.children)
                    return
                contador=contador+1
                MuestraListaTrie(Nodo.children)
                #si lo encuentro me voy por el ese nodo en la lista
                return insertR(T,element,Nodo.children[i],contador)
        #si no encontro la letra
```

```
            #si no encontro la letra
            if contador==len(element):
                t=TrieNode()
                t.key=element[contador-1]
                t.parent=Nodo
                Nodo.children.append(t)
                t.isEndOfWord = True
                MuestraListaTrie(Nodo.children)
            else:
                t=TrieNode()
                t.key=element[contador-1]
                t.parent=Nodo
                Nodo.children.append(t)
                contador=contador+1
                MuestraListaTrie(Nodo.children)
                insertR(T,element,t,contador)
        else:
```

```
    else:
        if contador==len(element):
            #llegamos al fin de la palabra
            t=TrieNode()
            t.key=element[contador-1]
            t.parent=Nodo
            Nodo.children=[]
            Nodo.children.append(t)
            #marco fin de palabra y termino
            t.isEndOfWord = True
            MuestraListaTrie(Nodo.children)
        else:
            #agrego a la lista de turno el nuevo nodo con la key
            t=TrieNode()
            t.key=element[contador-1]
            t.parent=Nodo
            Nodo.children=[]
            contador=contador+1
            Nodo.children.append(t)
            MuestraListaTrie(Nodo.children)
            #sigo para abajo
            insertR(T,element,t,contador)

```

```
def search(T,element):
    #busca una cadena en un arbol trie
    Nodo=T.root
    contador=0
    return searchR(T,element,Nodo,contador)
def searchR(T,element,Nodo,contador):
    #segunda parte del serch
    if Nodo.children is not None:
        if contador<len(element):
            #recorro la lista de su hijo en busca de una coincidencia
            for i in range(0,len(Nodo.children)):
                if Nodo.children[i].key is element[contador]:
                    contador=contador+1
                    #solo en este caso es cuando se devuelve true
                    if Nodo.children[i].isEndOfWord is True and contador==len(element) :
                        return True
                    return searchR(T,element,Nodo.children[i],contador)
            return False
        else:
            return False
    else:
        return False

```

Ejercicio 2:

Tendríamos la complejidad de O(m) si en vez de listas utilizáramos arrays

Ejercicio 3:

```python
"""Ejercicio 3"""

def delete(T,element):
    #elimina elementos de un arbol Trie
    if search(T,element) is False: return False #No se encontro la palabra
    #caso: palabra en otra mas larga
    Nodo=T.root
    contador=0
    Aux=GotoNode(T,element,Nodo,contador)
    if Aux.children is not None and Aux.isEndOfWord is True:
        Aux.isEndOfWord=False#desmarco porque es parte de otra
        return True
    elif Aux.children is None and Aux.isEndOfWord is True:
        #casos restantes
        contador=len(element)-1
        deleteR(T,Aux,contador,element)
        return True
```

```python
def deleteR(T,Nodo,contador,element):
    #segunda parte del Delete
    if Nodo==T.root: return
    if len(Nodo.parent.children) == 1:
        #significa que es un nodo unico por lo que borro todo el nodo
        if element[len(element)-1] is Nodo.key and Nodo.children is None and Nodo.isEndOfWord is True :
            #caso de la ultima hoja que siempre la borro siendo unica
            Aux=Nodo.parent
            Nodo.parent.children=None
            contador=contador-1
            return deleteR(T,Aux,contador,element)
        else:
            if Nodo.isEndOfWord==True:
                #en ese caso no sigo borrando
                return
            else:
                #caso para el resto de los nodos unicos
                Aux=Nodo.parent
                Nodo.parent.children=None
                contador=contador-1
                return deleteR(T,Aux,contador,element)
```

```
139         else:
140             #caso para cuando no es unico
141             for i in range  (0,len(Nodo.parent.children)):
142                 if Nodo.parent.children[i].key==element[contador]:
143                     if Nodo.parent.children[i].isEndOfWord==True:
144                         #en ese caso no sigo borrando
145                         return
146                     else:
147                         #caso para el cual no es unico ni es el final de una palabra
148                         Aux=Nodo.parent.children[i].parent
149                         Nodo.parent.children.remove(Nodo.parent.children[i])
150                         contador=contador-1
151                         return deleteR(T,Aux,contador,element)
```

(function) children: Any

```
289
290 def GotoNode(T,element,Nodo,contador):
291     #recorre el arbol y devuelve un nodo del final de palabra
292     if Nodo.children is not None:
293         if contador<len(element):
294             #recorro la lista de su hijo en busca de una coincidencia
295             for i in range(0,len(Nodo.children)):
296                 if Nodo.children[i].key is element[contador]:
297                     contador=contador+1
298                     #retorno el nodo final de la palabra
299                     if Nodo.children[i].isEndOfWord is True and contador==len(element) :
300                         return Nodo.children[i]
301                     return GotoNode(T,element,Nodo.children[i],contador)
302
```

Ejercicio 4:

```
154 """Ejercicio 4"""
155
156 def WordsinTrie(T,P,N):
157     """dado un árbol Trie T, escriba todas las palabras del árbol que empiezan por p y sean de longitud n. """
158     if len(P)-1>N: return None #tomo como que siempre empieza de 0
159     Nodo=T.root
160     contador=0
161     NodoAux=WordsinTrieR(T,P,N,contador,Nodo) #retorno la palabra
162     if NodoAux is None:
163         print (P) #ya tiene esa long
164         return
165     else:
166         Nodo=NodoAux
167         AllwordsWhithCondition(T,NodoAux.children,P,N,Nodo)
```

```python
def AllwordsWhithCondition(T,Nodo,cadena,N,NodoAux):
    #busca todas las palabras de un Trie y las devuelve en una lista con las condiciones antes dichas
    if Nodo is not NodoAux:
        for i in range (0,len(Nodo)):
            cadena=cadena+Nodo[i].key
            if Nodo[i].isEndOfWord==True and len(cadena)==N:
                print(cadena)
            if Nodo[i].children is None:
                cadena=cadena[:-1] #le voy sacando la ultima letra
                return cadena
            AllwordsWhithCondition(T,Nodo[i].children,cadena,N,NodoAux)
            if (i+1)<=len((Nodo)):
                cadena=cadena[:-1]
    else: return None
```

```python
def WordsinTrieR(T,P,N,contador,Nodo):
    #segunda parte
    if contador<len(P):
        for i in range(0,len(Nodo.children)):
            if Nodo.children[i].key==P[contador]:
                if contador is N and Nodo.children[i].isEndOfWord==True:
                    #En ese caso ya la longitud es la de la palabra ingresada
                    return None
                elif contador is N and Nodo.children[i].isEndOfWord==False:
                    #se alcanzo la longitud y no hay palabra
                    return None
                contador+=1
                return WordsinTrieR(T,P,N,contador,Nodo.children[i])
    return Nodo
```

Ejercicio 5:

```python
"""Ejercicio 5"""

def CompTrie(T1,T2):
    """ dado los Trie T1 y T2 devuelva True
    si estos pertenecen al mismo documento y False en caso contrario."""
    Nodo1=T1.root
    Nodo2=T2.root
    cadena=""
    ListaR1=[]
    a=Allwords(T1,Nodo1.children,ListaR1,cadena)
    print("Lista1: ",a)
    ListaR2=[]
    cadena=""
    b=Allwords(T2,Nodo2.children,ListaR2,cadena)
    print("Lista2: ",b)
    contador=0
    for i in range (0,len(ListaR1)):
        for j in range (0,len(ListaR2)):
            if ListaR1[i] == ListaR2[j]:
                contador+=1
    if contador==len(ListaR2):
        return True
    else:
        return False
```

```python
def Allwords(T,Nodo,ListaR,cadena):
    #busca todas las palabras de un Trie y las devuelve en una lista
    if Nodo is not T.root:
        for i in range (0,len(Nodo)):
            cadena=cadena+Nodo[i].key
            if Nodo[i].isEndOfWord==True:
                ListaR.append(cadena)
            if Nodo[i].children is None:
                cadena=cadena[:-1]
                return cadena
            Allwords(T,Nodo[i].children,ListaR,cadena)
            if (i+1)<=len((Nodo)):
                cadena=cadena[:-1]
    return ListaR
```

En el peor caso compara todas las palabras de los arboles: O(n^2)

Ejercicio 6:

```
def Invert(T):
    """devuelve True si existen en el documento T dos cadenas invertidas"""
    Nodo=T.root
    cadena=""
    Lista=[]
    ListaAux=Allwords(T,Nodo.children,Lista,cadena)
    Lista=[]
    Lista=Allwords(T,Nodo.children,Lista,cadena)
    #Aqui doy vuelta la lista para luego comparar
    for i in range (0,len(Lista)):
        ListaAux[i]=ListaAux[i][::-1]
    print(Lista)
    print(ListaAux)
    for i in range (0,len(Lista)):
        for j in range (0,len(ListaAux)):
            if Lista[i] == ListaAux[j]:
                return True
    return False
```

Ejercicio 7:

```
249  def autoCompletar(T, cadena):
250      """devuelve la forma de auto-completar la palabra"""
251      Nodo=T.root
252      CAux=""
253      contador=0
254      #voy hacia el nodo final de la cadena ingresada
255      Aux=GotoNode2(T,cadena,Nodo,contador)
256      Completado=Complete(Aux.children,CAux)
257      print(Completado)
258      return Completado
259
260  def Complete(Nodo,cadena):
261      #recorre a los nodos siguientes y para hasta encontar uno que es una lista como mas de 1 elemnto
262      if Nodo is None:
263          return cadena
264      if len(Nodo)>1 :
265          return cadena
266      else:
267          for i in range(0,len(Nodo)):
268              if Nodo[i].isEndOfWord is True:
269                  cadena=cadena+Nodo[i].key
270                  return cadena
271              else:
272                  cadena=cadena+Nodo[i].key
273              return Complete(Nodo[i].children,cadena)
```

```python
def GotoNode2(T,element,Nodo,contador):
    #recorre el arbol y devuelve un nodo del final de palabra(sin necesariamente ser endofword)
    if Nodo.children is not None:
        if contador<len(element):
            #recorro la lista de su hijo en busca de una coincidencia
            for i in range(0,len(Nodo.children)):
                if Nodo.children[i].key is element[contador]:
                    contador=contador+1
                    if contador==len(element):
                        return Nodo.children[i]
                    return GotoNode2(T,element,Nodo.children[i],contador)
```

Link de Replit:

https://replit.com/@EnzoPalau/Trie