Trabajo Practico N°3

Ejercicio 1

a)

```python
def insert(T, element):
    aux = 0

    if T.root == None:
        node = TrieNode()
        T.root = node
        aux = 1

    current = T.root
    for i in range(len(element)):
        if aux == 1:
            current.children = []
        children = current.children
        new_node = False

        if children != [] and children != None: #Si la lista no esta vacia busco el elemento
            new_node = search_list(element[i] , children)  #Devuelve true o false
        else:
            current.children = children #Sino, dejo la lista vacia

        if new_node == False: #Si no se encontro el elemento en la lista children entonces:
            new_node = TrieNode()
            new_node.parent = current
            new_node.key = element[i]
            if children != None:
                children.append(new_node)
            else:
                children = []
                current.children = children
                children.append(new_node)
        else:                       #Si el elemento ya esta en la lista entonces:
            index = search_pos(element[i] , children)
            new_node = children[index]
```

```python

        if len(element)-1 == i:
            new_node.isEndOfWord = True

        current = new_node

    return T


def search_list(element , lista):

    for i in range(len(lista)):
        if lista[i] != None:
            if element == lista[i].key:
                return True
        if lista[i] == None:
            lista.pop(i)

    return False

def search_pos(element , lista):

    for i in range(len(lista)):
        if element == lista[i].key:
            return i

    return None

```

b)

```
73
74  def search(T, element):
75      node = T.root
76      return searchR(node,element)
77
78
79  def searchR(node,element):
80      children = node.children
81
82      if children == None:
83          return False
84      aux = search_list(element[0],children)
85      pos = search_pos(element[0],children)
86      if pos != None:
87          aux2 = children[pos]
88
89      if aux == False :
90          return False
91
92      if len(element) == 1 and aux2.isEndOfWord == True:
93          return True
94
95      element = element[1:]
96      return searchR(aux2 , element)
```

Ejercicio 2

Si le asigno una Key a cada letra del alfabeto y estos se los asigno a cada elemento de un Array. En ese caso la búsqueda dentro del Array serias O(1) y deberíamos recorrer la longitud de la palabra, que sería O(m).

Ejercicio 3

```python

def deleteR(node,element):
    ultimo_elemento = element[len(element)-1:]
    element = element[:len(element)-1]
    pos = search_pos(ultimo_elemento , node)

    #Caso 0: Borro una palabra que este dentro de otra mas grande (Hola , Holanda)
    if node[pos].children != None and node[pos].isEndOfWord == True:
        node[pos].isEndOfWord = False
        return True
    elif node[pos].children == None and len(node)==1: #Caso 1: Si el ultimo elemento no tiene hijos y es de longitud 1
        aux = node[pos].parent.parent.children
        node[0].parent.children = None
        return deleteR(aux, element)
    elif len(node)>1 and node[pos].children == None: #Caso 2: Si el ultimo nodo no tiene hijos y la longitud es mayor a 1
        aux = node[pos].parent.parent.children
        node[pos].parent.children.pop(pos)
        return deleteR(aux, element)


def last_node(node , element):
    children = node.children
    pos = search_pos(element[0],children)
    aux = children[pos]
    element = element[1:]

    if children != None and element != "":
        return last_node(aux , element)
    return node.children
```

Ejercicio 4

```python

  def buscar_patron(T , p , n):
      if T.root == None:
          return None
      node = T.root
      last_node = fin_patron(p , node.children)
      if last_node == None:
          return None
      if last_node != None:
          n = n - len(p)
          cantidad , lista =  buscar_patron_recursive(last_node , n , 0 , [])
          if cantidad == 0:
              return None
          return lista_palabras(p , lista , n)
```

```python
def buscar_patron_recursive(node , n , count , lista):
    i=0
    aux = n
    aux1 = True
    if node != None:
        longitud = len(node)
        while n > 0 and i < longitud:
            n -= 1
            if n == 0 and node[i].isEndOfWord == True:
                lista.append(node[i].key)
                count += 1
                aux1 = False
            i += 1

        for i in range(len(node)):
            if aux1 == True:
                lista.append(node[i].key)
            count , lista =  buscar_patron_recursive(node[i].children , aux - 1, count , lista)

    return count , lista

```

```python

def lista_palabras(p , lista , n):
# Crear la lista de subcadenas con p al comienzo
    substrings = []
    i = 0
    while i + n <= len(lista):
        substring = ''.join(lista[i:i+n])
        substrings.append(p + substring)
        i += n
    return substrings


def fin_patron(p , node):
    if len(p) == 0:
        return node
    pos = search_pos(p[0] ,node)

    if pos == None:
        return None
    else:
        node = node[pos].children
        p = p[1:]
        return fin_patron(p ,node)

```