NOMBRE: ALVARO BLANCO
LEGAJO: 10622
TRABAJO PRACTICO TRIE

EJERCICIO 1

```python
def insert(T, element):
    node = T.root
    for char in element:
        index = ord(char) - ord('a')
        if node.children is None:
            node.children = [None] * 26
        if node.children[index] is None:
            node.children[index] = TrieNode()
            node.children[index].key = char
            node.children[index].parent = node
        node = node.children[index]
    node.isEndOfWord = True



def search(T, element):
    node = T.root
    for char in element:
        index = ord(char) - ord('a')
        if not node.children[index]:
            return False
        node = node.children[index]

    return node.isEndOfWord
```

EJERCICIO 2

Una opcion seria utilizar arrays con acceso en tiempo constante en lugar de listas linkeadas

EJERCICIO 3

```python
def delete(T, element):
    node = searchNode(T, element)
    if node is None or not node.isEndOfWord:
        return False

    # si el nodo tiene hijos solo removemos la bandera de final de palabra
    if node.children is not None and any(node.children):
        node.isEndOfWord = False
        return True
    # recorremos el trie hacia arriba y eliminamos los nodos que no tienen hijos y que no son
final de palabra
    while node.parent is not None:
        parent = node.parent
        index = ord(node.key) - ord('a')
        parent.children[index] = None

        if any(parent.children) or parent.isEndOfWord:
            break
        node = parent

    return True

def searchNode(T, element):
    node = T.root
    for char in element:
        index = ord(char) - ord('a')
        if not node.children[index]:
            return None
        node = node.children[index]
    if node and node.isEndOfWord:
        return node
    return None
```

EJERCICIO 4

```
def printWithLenN(T, p, n):
    if startsWith(T, p) == None:
        return None
    else:
        cont = 0
        node = startsWith(T, p)
        size = len(p)
        max = n - size
        element = p
        while node.children != None:
            for char in node.children:
                if char != None:
                    index = ord(char.key) - ord('a')
                    cont += 1
                    element += char.key
                    if cont == max:
                        return element
                    else:
                        node = node.children[index]
        return None


def startsWith(T, element):
    node = T.root
    words = []
    for char in element:
        index = ord(char) - ord('a')
        if not node.children[index]:
            return False
        node = node.children[index]
    return node
```

EJERCICIO 5

```
def checkIdenticalTrie(T1, T2):
   words1 = []
   words1 = traverse(T1.root, '', words1)
   words2 = []
   words2 = traverse(T2.root, '', words2)

   print(len(words1), len(words2))

   if len(words1) != len(words2):
      return False
   else:
      flags = []
      for j in range(0, len(words2) - 1):
         if words2[j] in words1:
            print(j)
            flags.append(True)
         else:
            return False
      for i in range(0, len(flags) - 1):
         if flags[i] != True:
            return False

      return True
```

EJERCICIO 6

```
def checkInvertedString(T):
   i = 0
   words = []
   words = traverse(T.root, '', words)
   size = len(words)
   for i in range(0, i < size - 1):
      if search(T, words[i]) and search(T, invertStr(words[i + 1])):
         return True
      else:
         continue
   return False



def invertStr(str):
   return str[::-1]
```

EJERCICIO 7

```
def autoCompletar(T, cadena):
   cont = 0
   if search(T, cadena):
      print(cadena)
      return cadena
   else:
      if startsWith(T, cadena) != False:
         node = startsWith(T, cadena)
         for char in node.children:
            if char != None:
                cont += 1
                nextChar = char
        if cont > 1:
            return ''
        else:
            autoCompletar(T, cadena + nextChar.key)
```