Caso de Estudio 2: Traductor AI2E (Abreviación de Internet a Español)

Con el uso constante de abreviaciones en redes sociales y mensajes de texto, es común (al menos para el profesor Alejandro) tener que averiguar qué significa una abreviación recibida por mensaje de texto.

Haremos un programa que nos permita guardar abreviaciones y su significado, y dada una abreviación, buscar su significado. Por ejemplo, si tenemos la abreviación "`lol`" nos gustaría poder encontrar "`riéndome`". Guardaremos esta información en una agenda.

Descomposición funcional:

- Buscar abreviación y entregar significado
- Agregar abreviación y significado
- Borrar un abreviación
- Cambiar significado de una abreviación

Nota. Por simplicidad, no se aceptan abreviaciones repetidas.

Solución con Listas

Estructura de datos: lista ordenada de registros con abreviaciones y significados

```
#registro: abrev(str) sig(str)
import estructura
estructura.crear("registro","abrev sig")

#agenda: lista(registro)
from lista import *
#ejemplo
agenda= lista(registro("afk","lejos del teclado"),\
         lista(registro("omg","sorprendido"),\
           lista(registro("rofl","riendome"),None)))
```

Buscar por abreviacion en una lista ordenada y entregar significado

```
#buscar: str lista(registro) -> str
```

```
#buscar abreviacion en agenda y devolver significado
#(None si no está)
#ej: buscar("rofl",agenda)->"riendome"
#ej: buscar("vip",agenda)->None
def buscar(abr,agenda):

    assert agenda==None or type(agenda)==lista

    if agenda==None:
        return None

    reg=cabeza(agenda)
    if reg.abrev==abr:
        return reg.sig

    if reg.abrev > abr:
        return None

    return buscar(abr,cola(agenda))

assert buscar("rofl",agenda)=="riendome"
assert buscar("vip",agenda)==None
```

Agregar nueva abreviación y significado

```
#agregar: str str lista(registro) -> lista(registro)
#devuelve una copia de la agenda donde se ha agregado
#la nueva abreviacion y significado
#(retorna misma agenda si abreviacion ya existia en la agenda)
#ej: agregar("wtf","confundido",agenda)->
#    lista(registro("afk","lejos del teclado"), \
#    lista(registro("omg","sorprendido"), \
#    lista(registro("rofl","riendome"), \
#    lista(registro("wtf","confundido"),None))))
def agregar(abr,sig,agenda):
    assert agenda==None or type(agenda)==lista

    if agenda==None:
        return lista(registro(abr,sig),None)

    #caso no es vacia la lista
    reg=cabeza(agenda)

    # 3 casos: 1) ya existe abr, 2)reg.abrev es mayor, y 3)reg.abrev es menor
    if reg.abrev==abr: #ya tenemos esta abreviacion
        return agenda

    if reg.abrev > abr:
```

```
        return lista(registro(abr,sig),agenda) # lo agrego al ppio de la lista

    return lista(reg,agregar(abr,sig,cola(agenda))) # tengo que agregarlo en la cola

# Tests
nueva_agenda = lista(registro("afk","lejos del teclado"), \
    lista(registro("omg","sorprendido"), \
    lista(registro("rofl","riendome"), \
    lista(registro("wtf","confundido"),None))))
assert agregar("wtf","confundido",agenda)== nueva_agenda
```

Borrar un abreviación

```
#borrar: str lista(registro)->lista(registro)
#devuelve una nueva agenda igual a la dada pero borra el registro con abrev (si existe
#ej: borrar("omg",agenda)->
# lista(registro("afk","lejos del teclado"),lista(registro("rofl","riendome"),None))

def borrar(abr,agenda):
    assert agenda==None or type(agenda)==lista

    if agenda==None:
        return None
    reg=cabeza(agenda)
    # 3 casos posibles
    if reg.abrev==abr: # es la cabeza
        return cola(agenda)

    if reg.abrev>abr: #no esta
        return agenda

    return lista(reg,borrar(abr,cola(agenda)))#tengo que borrarlo de la cola

assert borrar("omg",agenda)== \
  lista(registro("afk","lejos del teclado"), \
    lista(registro("rofl","riendome"),None))
```

Cambiar el significado de una abreviación

```
#cambiar: str str lista(registro)->lista(registro)
#Retorna una copia de la agenda pero donde se ha cambiado el significado
#de la abreviacion (si existia)
#ej: cambiar("omg","impactado",agenda)->lista(registro("afk","lejos del teclado"),
#  lista(registro("omg","impactado"),lista(registro("rofl","riendome"),None)))

def cambiar(abr,sig,agenda):
```

```
    assert agenda==None or type(agenda)==lista

    if buscar(abr,agenda)==None:
        return agenda

    return agregar(abr,sig,borrar(abr,agenda))

assert cambiar("omg","impactado",agenda)==\
  lista(registro("afk","lejos del teclado"), \
    lista(registro("omg","impactado"), \
    lista(registro("rofl","riendome"),None)))
```

Solución con ABB

Estructura de datos: ABB de registros con abreviaciones y significados

```
#Arbol Binario
#AB: valor(any), izq(AB), der(AB)
estructura.crear("AB","valor izq der")

#esArbol: AB->bool
#devuelve True si es AB y False si no
#ej...(completar)
def esArbol(A):
    return A==None or type(A)==AB
#test..(completar)


#registro: abrev(str) sig(str)
import estructura
estructura.crear("registro","abrev sig")

#agenda: AB(registro)

agenda=AB(registro("omg","sorprendido"), \
          AB(registro("afk","lejos del teclado"),None,None), \
          AB(registro("rofl","riendome"),None,None))
```

Buscar por abreviación en ABB y entregar significado

```
#buscar: str AB(registro) -> str
#buscar abreviacion en agenda y devolver significado
#(None si no esta)
#ej: buscar("rofl",agenda)->"riendome"
#ej: buscar("vip",agenda)->None
```

```
def buscar(abr,agenda):
    assert esArbol(agenda)

    if agenda==None:
        return None

    reg_actual = agenda.valor
    if abr < reg_actual.abrev:
        return buscar(abr,agenda.izq)

    if abr > reg_actual.abrev:
        return buscar(abr,agenda.der)

    return reg_actual.sig #son iguales

assert buscar("rofl",agenda)=="riendome"
assert buscar("vip",agenda)==None
```

Agregar un registro en un ABB

```
#agregar: str str AB(registro) -> AB(registro)
#devuelve una copia de la agenda donde se ha agregado
#la nueva abreviacion y significado
#(retorna misma agenda si abreviacion ya existia en la agenda)
#ej: agregar("wtf","confundido",agenda)->
#    AB(registro("omg","sorpendido"), \
#    AB(registro("afk","lejos del teclado"), None,None)), \
#    AB(registro("rofl","riendome"),
#          None, \
#          AB(registro("wtf","confundido"),None,None))) \

def agregar(abrev,sig,agenda):
    assert esArbol(agenda)
    if agenda==None:
        return AB(registro(abrev,sig),None,None)

    reg_actual=agenda.valor

    if abrev < reg_actual.abrev:
        return AB(reg_actual,agregar(abrev,sig,agenda.izq),agenda.der)

    if abrev > reg_actual.abrev:
        return AB(reg_actual,agenda.izq,agregar(abrev,sig,agenda.der))

    return agenda #si ya existe, dejar igual

# Tests
nueva_agenda = AB(registro("omg","sorprendido"), \
```

```
          AB(registro("afk","lejos del teclado"), None,None), \
          AB(registro("rofl","riendome"),
             None, \
             AB(registro("wtf","confundido"),None,None)))
assert agregar("wtf","confundido",agenda) == nueva_agenda
```

Borrar un registro de un ABB

```
#borrar: str AB(registro) -> AB(registro)
#devuelve una nueva agenda igual a la dada pero borra el registro con abrev (si existe
#ej: borrar("afk",agenda)->
#    AB(registro("omg","sorprendido"),\
#       None, \
#       AB(registro("rofl","riendome"),None,None))

def borrar(abr,agenda):
    assert esArbol(agenda)
    if agenda==None: return None

    reg_actual=agenda.valor

    if abr < reg_actual.abrev:
        return AB(reg_actual, borrar(abr,agenda.izq), agenda.der)

    if abr > reg_actual.abrev:
        return AB(reg_actual, agenda.izq, borrar(abr,agenda.der))

    #Lo encontramos en la raiz, es decir, abr==reg.abrev

    #caso 1: si arbol izquierdo vacio, devolver derecho
    if agenda.izq==None: return agenda.der

    #caso 2: si arbol derecho vacio, devolver izquierdo
    if agenda.der==None: return agenda.izq

    #caso 3: si arboles izquierdo y derecho no vacios
    #reemplazar por mayor del arbol izquierdo
    #y borrar mayor del arbol izquierdo
    reg_actual=mayorABB(agenda.izq)
    return AB(reg_actual,borrar(reg_actual.abrev,agenda.izq),agenda.der)

# Tests
nueva_agenda = AB(registro("omg","sorprendido"),\
          None, \
          AB(registro("rofl","riendome"),None,None))
assert borrar("afk",agenda) == nueva_agenda
```

Encontrar el mayor elemento en un ABB

```
#mayorABB: AB(any) -> any
#mayor valor de ABB A
#ej: mayorABB(abb)->"C“

def mayorABB(A):
    assert type(A)==AB #por lo menos un valor
    if A.der==None: return A.valor
    return mayorABB(A.der)

#Tests
abb=AB("B",AB("A",None,None),AB("C",None,None))
assert mayorABB(abb)=="C"
```