10/30/21 12:55:46 C:\Users\Edu\Desktop\ALGO 2\Practica\Diseño\guia 3\PARCIAL 2.tad

```
1
2  Sebastian Eduardo Cagnoni
3  Lu: 120/19
4
5  representación:
6      central se representa con estr, donde
7          estr es tupla(
8              diccIdInterno: dicc_AVL( id: idUsuario, tupla⟨interno: interno_Type,
9                                       itUsuariosA: itConj(UsuariosActuales)
10                                      )), //iterador que va al siguiente de la
   clave
11                                      //correspondiente en el conjunto
   UsuariosActuales
12             UsuariosActuales: conjunto_lineal(idUsuario) //referencia al conjunto
   lineal
13             diccInternoRegistrados: diccTrie(interno, usuario) //
14             //forma del trie
15             //                          x
16             //                        /   \
17             //                      1      0        es acotado segun l,
18             //                    ...    ...      el largo del numero ingresado
   al sistema
19             //Asumo que la cantidad que se puede registrar es sensatamente finita
20
21         ⟩
22         idUsuario    es el tipo nat
23         digit        es el tipo nat   //asumo que este nat va a estar limitado a
   1 o 0 en la rep
24         interno_Type es el tipo lista_enlazada(digit) // esto representa al
   numero interno
25
26
27
28
29
30 solución informal:
31
32 1. No hay prefijos entre cada interno
33 2. Los usarios del diccionario son los mismos que el de los UsuariosActuales
34 3. los elementos de interno pueden ser 1 o 0
35 4. //Asumo que no hay usuarios repetidos
36 5. los internos del Trie pertenecen a todos los internos del dicc_AVL al igual
   que los usuarios
37 de este
38
39
40 invariante de representación:
41     Rep: estr → boolean
42     (∀e: estr) Rep(e) ≡ true ⇔
43             1. (∀ k: idUsuario)(∀ p: idUsuario)(k ∈ claves(e.diccIdInterno) ∧
44                                     p ∈ claves(e.diccIdInterno) ∧ ¬(p = k) ⇒L
45                                    ¬esPrefijo(significado(k,
   e.diccIdInterno).interno,
46                                     significado(p, e.diccIdInterno).interno ∧
47                                     ¬esPrefijo(significado(p,
   e.diccIdInterno).interno,
48                                     significado(k, e.diccIdInterno).interno) ∧
49             2. (∀ i: idUsuario)(i ∈ e.UsuariosActuales ⇔ i ∈
   claves(e.diccIdInterno)) ∧L
50             3. (∀ s: interno_Type)((∀ d: digit)(∀ k: idUsuario))(
51                 k ∈ claves(e.diccIdInterno) ∧ significado(k, e.diccIdInterno) = s
52                         esta?(d, s)  ⇒L d = 1 ∨ d = 0)
53
54             4. //Asumo que no hay repetidos de usuarios
55             5. (∀ i: interno)(i ∈ claves(e.diccInternoRegistrados) ⇔
```

En realidad tenias que hacer que digito sea {0,1}, como en el TAD

Esta bien pues dicc y conj no pueden tener repetidos (de otra forma, sus reps no serian validos)

YLuego

```
                    (∃ k : idUsuario)(k ∈ claves(e.diccIdInterno) ∧
                        significado(e.diccIdInterno, k).interno = i)) ∧
            6. (∀ k: idUsuario)(∀ i: interno)(k ∈
significado(e.diccInternoRegistrados, k)
                                    ⟺ k ∈ e.UsuariosActuales)


función de abstracción:
    Abs: estr e → algo { Rep(e) }
    (∀e: estr) Abs(e) =obs ct |
                (∀ k: idUsuario)(k ∈ claves(e.diccIdInterno) ⟺ k ∈ usuarios(ct))
∧
                (∀ k: idUsuario)(k ∈ e.UsuariosActuales ⟺ k ∈ usuarios(ct) ∧ k
∈
claves(e.diccIdInterno)) ∧
                (∀ k: idUsuario)(k ∈ claves(e.diccIdInterno)⟺
                                    significado(k, e.diccIdInterno) = interno(ct, k))
∧
                (∀ i: interno)(i ∈ claves(e.diccInternoRegistrados) ⟺
                                    (∃ k: idUsuario)
(significado(e.diccInternoRegistrados, k) =
                                        interno(ct, k))


Interfaz

 parámetros formales:
     géneros: α
     funciones:
         copiar(in x: α) → res: bool
         Pre ≡ { true }
         Post ≡ { res =obs x }
         Descripción: Devuelve una copia de x
         Complejidad: Θ(copiar(x))
         Aliasing: No presenta aspectos de aliasing

 se explica con: CentralCeroInfinita(α)

 géneros: central

 operaciones:
     AgregarInterno(in/out sistema: central, in interno: interno_Type,
                                                in usuario: idUsuario)
     Pre ≡ { Interno indicado no es prefijo
             de ningún otro en la central, Además,
             el usuario no está previamente definido en la central }
     Descripción: Agrega a la central el interno
                  para el usuario indicado al sistema.
     Complejidad: O(l + log(n))
     Aliasing: Agrega por copia y crea un aliasing con el conjunto
UsuariosActuales
     y su correspondiente usuario

     EliminarInterno(in/out sistema: central, in usuario: idUsuario)
     Pre ≡ { El interno del usuario ya
             existe en el sistema}
     Descripción: Elimina de la central
                  el interno asociado al usuario indicado
     Complejidad: O(log(n))
     Aliasing: hay aliasing, se elimina por referencia en el conjunto
UsuariosActuales

     ExisteInterno?(in sistema: central,
          in interno: interno_Type) → res: bool
```

Significado(e.diccInterno, i) no asegura que i sea un interno valido, esto se indefine.

Te complicaste mucho, tenias que igualar los dos obs basicos y listo:
usuarios(ct) = claves(e.diccid..)
y
u ∈ usuarios(ct) =>L
interno(u) = significado(k,...)

```
    Pre ≡ { true }
    Descripción: Indica si el interno esta
                 informado esta registrado en la central
    Complejidad: O(1)
    Aliasing: Solo es busqueda sin efectos colaterales

    PuedeAgregarse?(in sistema: central,
                    in interno: interno_Type) → res: bool
    Pre ≡ { true }
    Descripción: Indica si el interno indicado podria
                 agregarse a la central de acuerdo a las restricciones
    Complejidad: O(1)
    Aliasing: Solo es Busqueda y Devuelve un bool

    Interno(in sistema: central,
            in usuario: idUsuario) → interno: interno_Type
    Pre ≡ { El usuario indicado
            esta registrado en el sistema }
    Descripción: ...
    Complejidad: O(log(n))
    Aliasing: Devuelve por copia el interno

    Usuarios(in sistema: central) → usrs: conjunto(idUsuario)
    Pre ≡ { true }
    Descripción: ...
    Complejidad: O(1)
    Aliasing: Devuelve una referencia no modificable

 algoritmos:
    iAgregarInterno(in/out sistema: central, in interno: interno_Type,
                                             in usuario: idUsuario)

        itParaDicc <-
sistema.UsuariosActuales::AgregarRapido(sistema.UsuariosActuales, usuario)
        //θ(copy(usuario))
        sistema.diccIdInterno::definir(sistema.diccIdInternoRegistrados,
                                  usuario, tupla⟨interno, itParaDicc⟩) //
        sistema.diccInternoRegistrados::definir(sistema.diccIdInternoRegistrados,
                         interno, ususario) //O(1)

        //como es AVL, tarda log(n) en insertar
        //pero redefino la funcion definir con un while interno
        //para que vaya añadiendo digito a digito hasta llegar
        //a la longitud el interno
        //...Busqueda y luego insertare el significado despues de hacer lo de
abajo
        // res en este momento es parte del significado del usuario que estoy
añadiendo
        //res <-lista_enlazada::Vacia()
        //
        //itlista <- CrearItLista(interno)
        //
        //while(HaySiguiente?(it))
        //   elem <- Siguiente(itLista)
        //   lista_enlazada::AgregarAdelante(res, elem)
        //   lista_enlazada::Avanzar(it)
        // Aclaracion: Luego devuelvo res para definirlo en el diccionario
Peor Caso: interno mide l de longitud
Complejidad Final: O(log(n) + l + l) ≡ O(log(n) + 2l) ≡ O(log(n) + l)
//l es un numero sensatamente acotado
Busqueda del AVL e insercion del res vacío : O(log(n))
+
copiado de l elementos y elemento a elemento hasta llegar a la longitud de
interno
en una lista enlazada: l*O(1) ≡ O(l), es Copy(interno)
+
Inserccion en diccTrie: O(l)
```

```
Mejor Caso: Interno mide 1 de longitud
Complejidad: Omega(Log(n)) y copy(interno) + O(1) ≡ Omega(log(n) + l=1)









     iEliminarInterno(in/out sistema: central, in usuario: idUsuario)
        //Por medio del iterador que busco en el diccionario
        //elimino el elemento en el conjunto lineal UsuariosActuales
        itBorrar <- sistema.diccIdInterno::Significado(sistema,
usuario).itParaDicc
        itBorrar::EliminarSiguiente(itBorrar) //elimino por referencia en un
        //conjunto lineal es O(1)
        itBorrar <- NULL //apunto a null
        sistema.diccIdInterno::Borrar(sistema.diccIdInterno, usuario)
        //Borrado en un diccionario AVL es Log(n)

     Fin Interfaz
Peor caso ≡ Mejor Caso ≡ Siempre voy a tener que eliminar una lista de longitud
l
Complejidad Final: θ (log(n))


Explicacion

Existe interno? busca en un diccionario Trie de interno: usuario, no hay
repetidos, por lo tanto
la búsqueda es O(l), l es el largo del interno

PuedeAgregarse? busca en el trie si hay coincidencias por alguna rama inicial (1
o 0) y
busca coincidencias, entre la totalidad del interno buscado con algun interno
clave del diccionario Trie, esto esta acotado
por l, por lo tanto es O(l) en el PeorCaso que es que para todos los internos el
interno buscado
se diferencie de todos por un solo digito

Interno Busca en el diccAVL, por lo tanto busqueda es O(log(n))

Usuarios Me devuelve la referencia al conjunto UsuariosActuales, es O(1)

```

No borras el interno del diccTrie donde lo tenes guardado. Te queda una estructura invalida

Tenes varias cosas muy bien pero:
- No cumplis la complejidad de EliminarInterno (o lo haces pero rompiendo la estructura)
- Tu estructura no permite cumplir todas las complejidades pedidas
- El abs no sigue la idea de lo que es la funcion de abstraccion y para lo que sirve
- No hay ninguna idea sobre saturado