1. Módulo Avld(nat, itLista(compu))

Interfaz

parámetros formales

se explica con: DICCIONARIO(NAT, ITLISTA(COMPU)).

géneros: `avld(nat, itLista(compu))`.

Operaciones básicas de Avld

VACÍO() $\rightarrow$ *res* : `avld`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ vacio()}
Complejidad: O(1)
Descripción: Crea un AVL vacío.

DEFINIDO?AVL(**in** A: `avld`, **in** id: `nat`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ def?(id,A)}
Complejidad: O(log(n) + log(k))
Descripción: Indica si hay un nodo en el AVL cuyo id es el nat pasado por parámetro.

SIGNIFICADOAVL(**in** A: `avld`, **in** id: `nat`) $\rightarrow$ *res* : `itLista(compu)`
Pre $\equiv$ {def?(id,A)}
Post $\equiv$ {res $=_{\text{obs}}$ obtener(id,A)}
Complejidad: O(log(n) + log(k))
Descripción: Busca el nodo del AVL cuyo id es el pasado por parámetro y devuelve su camino.
Aliasing: res es modificable si y sólo si A es modificable.

DEFINIRAVL(**in/out** A: `avld`,**in** id: `nat`,**in** *camino*: `itLista(compu)`)
Pre $\equiv$ {$A \equiv A_0$}
Post $\equiv$ {$A =_{\text{obs}} definir(id, camino, A_0)$}
Complejidad: O(log(n) + log(k))
Descripción: Inserta un Nodo al AVL, que tiene como id y camino los pasados por parámetro.

BORRARAVL(**in/out** A: `avld`,**in** id: `nat`)
Pre $\equiv$ {def?(id,A) $\wedge A \equiv A_0$}
Post $\equiv$ {$A =_{\text{obs}} borrar(id, A_0)$}
Complejidad: O(log(n) + log(k))

CLAVESAVL(**in** A: `avld`) $\rightarrow$ *res* : `conj(nat)`
Pre $\equiv$ {true}
Post $\equiv$ {$A =_{\text{obs}} claves(A)$}
Complejidad: O(n x k)
Descripción: Devuelve un conjunto que contiene todas las claves definidas en el AVL

Funciones auxiliares:

ConstruirClaves(inout p : puntero(nodo)) $\rightarrow$ res : conj(nat) -Dado un puntero a nodo de un avl, se agrega la id de ese nodo a un conjunto vacío. Después se hace lo mismo con los dos hijos del nodo y se agregan los resultados al conjunto. Con esto recorremos todo el árbol guardando las claves de los nodos y devolviendo las mismas como resultado en un conjunto.

FactorDeBalanceo(inout p : puntero(nodo))$\rightarrow$ res : int -Dado un puntero a nodo de un avl, se calcula su factor de balanceo. Esta valor numérico se consigue restando la altura de su hijo derecho con el izquierdo.

BalancearNodo(inout p : puntero(nodo)) $\rightarrow$ res : puntero(nodo) -Dado un puntero a nodo de un avl, se lo balancea si es necesario. Básicamente se trata de si el factor de balanceo del nodo es 2 o -2 realizar las operaciones pertinentes

para restablecer el invariante de balanceo del avl.

Balancear(inout iteradorRama : itLista(puntero(nodo)) -Dado el iterador de una lista de punteros a nodo de un avl, se busca aplicar la función BalancearNodo a cada uno de los miembros de la lista. .La lista en la que se itera representa una rama del avl invertida. Exceptuando al último que es el de raiz. Esto ultimo se encarga la función que lo llama.

CorreguirAlturas(inout iteradorRama: itLista(puntero(nodo)) -Dado el iterador de una lista de punteros a nodo de un avl, se busca recalcular las alturas de todos los miembros de la lista. La lista en la que se itera representa una rama del avl invertida. Para cada uno se toma la altura máxima entre su hijo izquierdo y derecho y se le suma 1(Esto es porque el propio nodo también cuenta en la altura).

BuscarElRemplazo(inout listaABalancear : lista(puntero(nodo), in aBorrar : puntero(nodo))→ res : puntero(nodo) -Cuando se elimina un nodo que tiene hijo izquierdo y derecho, esta función se encarga de buscar un candidato para remplazar el nodo eliminado. Se toma la una lista de punteros a nodos que forman la rama en la que se estuvo recorriendo para llegar al nodo que se busca quitar, solo que sin este último. También se utiliza un puntero al nodo que se pretende quitar. La función además va guardando los nodos que recorre en la lista de punteros a nodos ya que estos se tendrán que recalcular las alturas y balancear. Finalmente cuando se encuentra el candidato se debe devolver, no sin antes asegurarse que no se pierdan sus hijos si es que los tiene.

Representación

Representación de AVL

`avld` se representa con `avl`

donde `avl` es `tupla`(*raiz*: `puntero(nodo)`)
donde `nodo` es `tupla`(*id*: `nat` , *camino*: `itLista(compu)` , *izq*: `puntero(nodo)` , *der*: `puntero(nodo)` , *altura*: `nat`)

Invariante de representación en castellano: -El puntero raiz es NULL
-Si el puntero raiz no es NULL entonces la altura es 1 sii izq y der son NULL. Además es 2 sii o izq o der son NULL pero no ambos y además la suma entre la altura del hijo distinto de NULL debe ser 1. Y es 3 o mayor sii la suma de las alturas de izq y der mas 1 es de como resultado la altura del nodo padre.
-La resta de la altura del hijo derecho con el hijo izquierdo, si ninguno es vacío, de cualquier nodo no puede ser mayor en modulo a 1.
-Para todo nodo el hijo izquierdo tiene un id menor y el derecho mayor, si es que tiene alguno.
-Al ser un árbol y usar punteros tenemos que pedir que cada uno de los nodos no tenga punteros derecho o izquierdo a nodos que sean apuntados por otro. Osea que no de un bucle.
-Tanto el hijo izquierdo y derecho, si los hay, cumplen estas condiciones.

Abs : avl a $\longrightarrow$ Avld(nat, itLista(compu)) {Rep(a)}
Abs(a) $=_{obs}$ e: Avld(nat, itLista(compu)) | ($\forall$ n : nat)(def?(n, e) $\Leftrightarrow$ ExisteNodoConClave(*a.raiz, n) $\wedge_L$ (def?(n, e) $\Rightarrow_L$ obtener(n, e) = ObtenerAVL(*a.raiz, n)))

```
ExisteNodoConClave : nodo × nat  ⟶ bool
ExisteNodoConClave(nodo, n) ≡ if nodo.id = n then
                                  true
                              else
                                  if nodo.id <n then
                                      ExisteNodoConClave(*nodo.izq, n)
                                  else
                                      ExisteNodoConClave(*nodo.der, n)
                                  fi
                              fi
ObtenerAVL : nodo × nat  ⟶ itLista(compu)
```

ObtenerAVL(nodo, n) ≡ **if** nodo.id = n **then**
nodo.camino
else
if nodo.id <n **then** ObtenerAVL(*nodo.izq, n) **else** ObtenerAVL(*nodo.der, n) **fi**
fi

Algoritmos

---

iVacío() → *res* : *avl*
res ← <NULL> ▷ O(1)

Complejidad: O(1)
Justificación:

---

iDefinido?AVL(**in** *A* : `avl`, **in** *i* : `nat`) → *res* : *bool*
recorrido : puntero(nodo) ← A.raiz ▷ O(1)
res ← false ▷ O(1)
while ¬res ∧ recorrido ≠ NULL **do** ▷ O(2 x log(n)) = O(log(n))
if id = *recorrido.id **then** ▷ O(2)
res ← true ▷ O(1)
else if id <*recorrido.id **then** ▷ O(2)
recorrido ← *recorrido.izq ▷ O(1)
else ▷ O(2)
recorrido ← *recorrido.der ▷ O(1)
end if
end while

Complejidad: O(log(n))
Justificación: O(log(n) + 2) = O(log(n))

SignificadoAVL(**in** *A* : `avl`, **in** *id* : `nat`) → *res* : itLista(compu)
recorrido : puntero(nodo) ← A.raiz ▷ O(1)
llegue : bool ← false ▷ O(1)
while ¬llegue **do** ▷ O(2 x log(n)) = O(log(n))
if id = *recorrido.id **then** ▷ O(2)
llegue ← true ▷ O(1)
else if id <*recorrido.id **then** ▷ O(2)
recorrido ← *recorrido.izq ▷ O(1)
else ▷ O(2)
recorrido ← *recorrido.der ▷ O(1)
end if
end while
res ← *recorrido.camino ▷ O(1)

Complejidad: O(log(n))
Justificación: O(3 + log(n)) = O(log(n))

DefinirAVL(**in/out** A : `avl`,**in** id : `nat`,**in** *camino* : `itLista(compu)`)

```
if A.raiz = NULL then                                                          ▷ O(2)
    *A.raiz ← δ<id, camino, NULL, NULL, 1>                                     ▷ O(1)
else                                                          ▷ O(3 x log(n) + 4) = O(log(k))
    recorrido : puntero(nodo) ← A.raiz                                         ▷ O(1)
    llegue : bool ← false                                                      ▷ O(1)
    necesitaBalancear : bool ← true                                            ▷ O(1)
    listaABalancear : lista(puntero(nodo)) ← Vacía()                           ▷ O(1)
    while ¬llegue do                                      ▷ O(5 x log(n)) = O(log(n))
        AgregarAdelante(listaABalancear, recorrido)       ▷ O(copy(recorrido)) = O(1)
        if id = *recorrido.id then                                             ▷ O(4)
            recorrido ← δ<id, camino, *recorrido.izq, *recorrido.der, *recorrido.altura>   ▷ O(1)
            llegue ← true                                                      ▷ O(1)
            necesitaBalancear ← false                                          ▷ O(1)
        else if id <*recorrido.id then                                         ▷ O(4)
            if *recorrido.izq = NULL then                                      ▷ O(3)
                llegue ← true                                                  ▷ O(1)
                *recorrido.izq ← δ<id,camino,NULL,NULL,1>                      ▷ O(1)
            else                                                               ▷ O(2)
                recorrido ← *recorrido.izq                                     ▷ O(1)
            end if
        else                                                                   ▷ O(4)
            if *recorrido.der = NULL then                                      ▷ O(3)
                llegue ← true                                                  ▷ O(1)
                *recorrido.der ← δ<id,camino,NULL,NULL,1>                      ▷ O(1)
            else                                                               ▷ O(2)
                recorrido ← *recorrido.der                                     ▷ O(1)
            end if
        end if
    end while
    if necesitaBalancear then                         ▷ O(log(n) + 3) = O(log(n))
        iteradorRama : itLista(puntero(nodo)) ← crearIT(listaABalancear)       ▷ O(1)
        CorreguirAlturas(iteradorRama)                                ▷ O(log(n))
        iteradorRama2 : itLista(puntero(nodo)) ← crearIT(listaABalancear)      ▷ O(1)
        Balancear(iteradorRama2)                                      ▷ O(log(n))
        A.raiz ← BalancearNodo(Anterior(iteradorRama2))                        ▷ O(1)
    end if
end if
```

Complejidad: O(log(n))
Justificación: O(2 x log(n) + 4) = O(log(n))

BorrarAVL(**in**/**out** A : `avl`,**in** id : `nat`)

```
listaABalancear : lista(nodo) ← Vacía()                                          ▷ O(1)
if *A.raiz.id = id then                                                          ▷ O(log(n))
    if *A.raiz.izq = NULL ∧ *A.raiz.der = NULL then                              ▷ O(4)
        A.raiz ← NULL                                                            ▷ O(1)
    else if *A.raiz.izq = NULL then                                              ▷ O(2)
        A.raiz ← *A.raiz.der                                                     ▷ O(1)
    else if *A.raiz.der = NULL then                                              ▷ O(2)
        A.raiz ← *A.raiz.izq                                                     ▷ O(1)
    else                                                  ▷ O(3 x log(n) + 8) = O(log(n))
        remplazo : puntero(compu) ← BuscarElRemplazo(listaABalancear, A.raiz)    ▷ O(log(n))
        *remplazo.der ← *A.raiz.der                                              ▷ O(1)
        if Longitud(listaABalancear) ≠ 1 then                                    ▷ O(2)
            *remplazo.izq ← *A.raiz.izq                                          ▷ O(1)
        end if
        A.raiz ← remplazo                                                        ▷ O(1)
        iteradorRama : itLista(puntero(nodo)) ← crearIT(listaABalancear)         ▷ O(1)
        CorreguirAlturas(iteradorRama)                                           ▷ O(log(n))
        iteradorRama2 : itLista(puntero(nodo)) ← crearIT(listaABalancear)        ▷ O(1)
        Balancear(iteradorRama2)                                                 ▷ O(log(n))
        A.raiz ← BalancearNodo(Anterior(iteradorRama2))                          ▷ O(1)
    end if
else                                                  ▷ O(2 x log(n) + 2) = O(log(n))
    recorrido : puntero(nodo) ← A.raiz                                           ▷ O(1)
    llegue : bool ← false                                                        ▷ O(1)
    while ¬llegue do                                      ▷ O(5 x log(n)) = O(log(n))
        AgregarAdelante(listaABalancear, recorrido)       ▷ O(copy(recorrido)) = O(1)
        if *recorrido.id = id then                                               ▷ O(2)
            llegue ← true                                                        ▷ O(1)
        else if *recorrido.id < id then                                          ▷ O(2)
            recorrido ← *recorrido.izq                                           ▷ O(1)
        else if *recorrido.id > id then                                          ▷ O(2)
            recorrido ← *recorrido.der                                           ▷ O(1)
        end if
    end while
    if *recorrido.izq = NULL ∧ *recorrido.der = NULL then                        ▷ O(5)
        if *listaABalancear[2].izq.id = *recorrido.id then                       ▷ O(2)
            *listaABalancear[2].izq ← NULL                                       ▷ O(1)
        else                                                                     ▷ O(2)
            *listaABalancear[2].der ← NULL                                       ▷ O(1)
        end if
    else if *recorrido.izq = NULL then                                           ▷ O(3)
        if *listaABalancear[2].izq.id = *recorrido.id then                       ▷ O(2)
            *listaABalancear[2].izq ← *recorrido.der                             ▷ O(1)
        else                                                                     ▷ O(2)
            *listaABalancear[2].der ← *recorrido.der                             ▷ O(1)
        end if
    else if *recorrido.der = NULL then                                           ▷ O(3)
        if *listaABalancear[2].izq.id = *recorrido.id then                       ▷ O(2)
            *listaABalancear[2].izq ← *recorrido.izq                             ▷ O(1)
        else                                                                     ▷ O(2)
            *listaABalancear[2].der ← *recorrido.izq                             ▷ O(1)
        end if
    else                                                  ▷ O(3 x log(n) + 6) = O(log(n))
        Aux ← listaABalancear[2]                                                 ▷ O(1)
        remplazo ← BuscarElRemplazo(listaABalancear, recorrido)                  ▷ O(log(n))
        El resto de else está en la siguiente hoja
    end if
end if
```

Complejidad: O(log(n))
Justificación: O(log(n) + 1) = O(log(n))

if *Aux.izq.id = *recorrido.id **then** ▷ O(2)
  *Aux.izq ← remplazo ▷ O(1)
else ▷ O(2)
  *Aux.der ← remplazo ▷ O(1)
end if
iteradorRama : itLista(puntero(nodo)) ← crearIT(listaABalancear) ▷ O(1)
CorreguirAlturas(iteradorRama) ▷ O(log(n))
iteradorRama2 : itLista(puntero(nodo)) ← crearIT(listaABalancear) ▷ O(1)
Balancear(iteradorRama2) ▷ O(log(n))
A.raiz ← BalancearNodo(Anterior(iteradorRama2)) ▷ O(1)

ClavesAVL(**in** A : `avl`) → res : conj(nat)
res ← ConstruirClaves(A.raiz) ▷ O(n)

Complejidad: O(n)

---

iConstruirClaves(**in** p : `puntero(nodo)`) → res : conj(nat)
res ← Vacío() ▷ O(1)
if p = NULL **then** ▷ O(3 + Cardinal(resIzq) + Cardinal(resDer) + $(2^{*(*p.izq.altura)} - 1) + (2^{*(*p.izq.altura)} - 1)$)
  AgregarRapido(res, *p.id) ▷ O(1)
  resIzq : conj(nat) ← ConstruirClaves(*p.izq) ▷ O($(2^{*(*p.izq.altura)} - 1)$)
  resDer : conj(nat) ← ConstruirClaves(*p.der) ▷ O($(2^{*(*p.izq.altura)} - 1)$)
  iteradorIzq : itConj(compu) ← crearIT(resIzq) ▷ O(1)
  iteradorDer : itConj(compu) ← crearIT(resDer) ▷ O(1)
  while HaySiguiente(iteradorIzq) **do** ▷ O(3 x Cardinal(resIzq)) = O(Cardinal(resIzq))
    AgregarRapido(res, *Siguiente(iteradorIzq).id) ▷ O(copy(*Siguiente(iteradorIzq).id)) = O(1)
    Avanzar(iteradorIzq) ▷ O(1)
  end while
  while HaySiguiente(iteradorDer) **do** ▷ O(3 x Cardinal(resDer)) = O(Cardinal(resDer))
    AgregarRapido(res, *Siguiente(iteradorDer).id) ▷ O(copy(*Siguiente(iteradorDer).id)) = O(1)
    Avanzar(iteradorDer) ▷ O(1)
  end while
end if

Complejidad: O(n)
Justificación: O(4 + Cardinal(resIzq) + Cardinal(resDer) + $(2^{*(*p.izq.altura)} - 1) + 2^{*(*p.izq.altura))}$ = O(6 + 2 x n) = O(n). $(2^{*(*p.izq.altura)} - 1)$ es una cota superior para la cantidad de elementos del subárbol izquierdo, idem derecho. Ya que la suma de los dos cardinales representan a la totalidad de elemntos del diccionario menos uno. AgregarRapido se puede usar porque la id no se repite nunca en la recurción(es un árbol binario).

---

iFactorDeBalanceo(**in** p : `puntero(nodo)`) → res : int
res ← 0 ▷ O(1)
if *p.izq ≠ NULL **then** ▷ O(2)
  res ← res - *p.izq.altura ▷ O(1)
end if
if *p.der ≠ NULL **then** ▷ O(2)
  res ← res + *p.der.altura ▷ O(1)
end if

Complejidad: O(1)
Justificación: O(2 + 2 + 1) = O(5) = O(1)

iBalancearNodo(**in**/**out** p: `puntero(nodo)`)→ res : puntero(nodo)

```
if FactorDeBalanceo(p) = 2 then                                  ▷ O(12)
    P : puntero(nodo) ← p                                        ▷ O(1)
    Q : puntero(nodo) ← *P.izq                                   ▷ O(1)
    if FactorDeBalanceo(Q) = 1 then                              ▷ O(4)
        *P.der ← *Q.izq                                          ▷ O(1)
        *Q.izq ← P                                               ▷ O(1)
        *P.altura ← *P.altura - 2                                ▷ O(1)
        res ← Q                                                  ▷ O(1)
    else                                                         ▷ O(10)
        R : puntero(nodo) ← *Q.izq                               ▷ O(1)
        *P.der ← *R.izq                                          ▷ O(1)
        *Q.izq ← *R.der                                          ▷ O(1)
        *R.izq ← P                                               ▷ O(1)
        *R.der ← Q                                               ▷ O(1)
        *P.altura ← *P.altura - 2                                ▷ O(1)
        *Q.altura ← *Q.altura - 1                                ▷ O(1)
        *R.altura ← *R.altura + 1                                ▷ O(1)
        res ← R                                                  ▷ O(1)
    end if
else if FactorDeBalanceo(p) = -2 then                            ▷ O(12)
    P : puntero(nodo) ← p                                        ▷ O(1)
    Q : puntero(nodo) ← *P.der                                   ▷ O(1)
    if FactorDeBalanceo(Q) = -1 then                             ▷ O(5)
        *P.izq ← *Q.der                                          ▷ O(1)
        *Q.der ← P                                               ▷ O(1)
        *P.altura ← *P.altura - 2                                ▷ O(1)
        res ← Q                                                  ▷ O(1)
    else                                                         ▷ O(10)
        R : puntero(nodo) ← *Q.der                               ▷ O(1)
        *P.izq ← *R.izq                                          ▷ O(1)
        *Q.der ← *R.der                                          ▷ O(1)
        *R.izq ← P                                               ▷ O(1)
        *R.der ← Q                                               ▷ O(1)
        *P.altura ← *P.altura - 2                                ▷ O(1)
        *Q.altura ← *Q.altura - 1                                ▷ O(1)
        *R.altura ← *R.altura + 1                                ▷ O(1)
        res ← R                                                  ▷ O(1)
    end if
end if
```

Complejidad: O(1)
Justificación: O(12) = O(1)

iCorreguirAlturas(**in**/**out** *iteradorRama*: `itLista(puntero(nodo))`)

```
while HaySiguiente(iteradorRama) do                                                    ▷ O(8 x log(n))
    auxAltura : nat ← 1                                                                ▷ O(1)
    if *Siguiente(iteradorRama).izq ≠ NULL ∧ *Siguiente(iteradorRama).der ≠ NULL then  ▷ O(5)
        if *Siguiente(iteradorRama).der.altura <*Siguiente(iteradorRama).izq then      ▷ O(2)
            auxAltura ← auxAltura + *Siguiente(iteradorRama).izq.altura                ▷ O(1)
        else                                                                           ▷ O(2)
            auxAltura ← auxAltura + *Siguiente(iteradorRama).der.altura                ▷ O(1)
        end if
    else if *Siguiente(iteradorRama).izq ≠ NULL then                                   ▷ O(2)
        auxAltura ← auxAltura + *Siguiente(iteradorRama).izq.altura                    ▷ O(1)
    else if *Siguiente(iteradorRama).der ≠ NULL then                                   ▷ O(2)
        auxAltura ← auxAltura + *Siguiente(iteradorRama).der.altura                    ▷ O(1)
    end if
    *Siguiente(iteradorRama).altura ← auxAltura                                        ▷ O(1)
    Avanzar(iteradorRama)                                                              ▷ O(1)
end while
```

Complejidad: O(log(n))
Justificación: O(8 x log(n)) = O(log(n)). El iteradorRama recorre una rama del árbol y esta mide log(n) siendo n la cantidad de nodos del avl.

iBalancear(**in**/**out** *iteradorRama*: `itLista(puntero(nodo))`)

```
while HaySiguiente(iteradorRama) do                                                    ▷ O(4 x log(n))
    Avanzar(iteradorRama)                                                              ▷ O(1)
    if HaySiguiente(iteradorRama) then                                                 ▷ O(3)
        if Anterior(iteradorRama) = *Siguiente(iteradorRama).izq then                  ▷ O(2)
            *Siguiente(iteradorRama).izq ← BalancearNodo(Anterior(iteradorRama))       ▷ O(1)
        else                                                                           ▷ O(2)
            *Siguiente(iteradorRama).der ← BalancearNodo(Anterior(iteradorRama))       ▷ O(1)
        end if
    end if
end while
```

Complejidad: O(log(n))
Justificación: O(4 x log(n)) = O(log(n))

iBuscarElRemplazo(**in**/**out** *listaABalancear* : `lista(puntero(nodo))`, **in** *a* : `B` orrarpuntero(nodo))→ *res* : puntero(nodo)

```
recorrido : puntero(nodo) ← *aBorrar.izq                                   ▷ O(1)
llegue ← false                                                             ▷ O(1)
if *recorrido.der ≠ NULL then                                              ▷ O(8 + log(n)) = O(log(n))
    i : nat ← 1                                                            ▷ O(1)
    while ¬llegue do                                                       ▷ O(6 x log(n)) = O(log(n))
        AgregarAdelante(listaABalancear, recorrido)   ▷ O(copy(recorrido)) = O(1). Porque es un puntero.
        if *recorrido.der = NULL then                                      ▷ O(4)
            listaABalancear[2].der ← *recorrido.izq                        ▷ O(1)
            listaABalancear[i] ← recorrido                                 ▷ O(1)
            llegue ← true                                                  ▷ O(1)
        else                                                               ▷ O(2)
            recorrido ← *recorrido.der                                     ▷ O(1)
        end if
        i ← i + 1                                                          ▷ O(1)
    end while
else
    AgregarAdelante(listaABalancear, recorrido)                            ▷ O(copy(recorrido)) = O(1)
end if
res ← recorrido                                                            ▷ O(1)
```

Complejidad: O(log(n))
Complejidad: O(log(n) + 11) = O(log(n))