

Algoritmos y Estructuras de Datos II

Segundo Cuatrimestre de 2009

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2 (Recuperatorio)

Diseño de Tipos Abstractos de Datos

Enrutador de tráfico IP

Grupo 7

Integrante	LU	Correo electrónico
Bianchi, Mariano	92/08	marianobianchi08@gmail.com
Brusco, Pablo	527/08	pablo.brusco@gmail.com
Di Pietro, Carlos Augusto Lyon	126/08	cdipietro@dc.uba.ar
Hernandez, Fernando	—	matematica527@yahoo.es

Aclaraciones

Extension del Secu(α)

El Tad SecuOrdSinRep(α) es una extension del Tad Secu(α) en la cual se agregan algunas operaciones y se reemplaza el generador \bullet por agregarOrdenado. En consecuencia, las operaciones hechas sobre el operador puntito pierden validez.

Ordenes de los elementos α

Durante este trabajo práctico se dió por sentado que se tenían los siguientes órdenes totales:

- En la estructura del enrutador, el campo *reglas* es de tipo secuOrdSinRep. Las tuplas que esta contiene, se ordenan por versión, es decir, se ordenan según el orden dado por el primer campo de cada tupla.
- En las secuencias de eventos, se da por sentado que el orden de los mismos está dado por el timestamp, ordenándose de forma creciente.

Conjunto de interfaces

Como se indica en el enunciado, decidimos reemplazar al conjunto de interfaces por una cantidad máxima de interfaces, las cuales van de 0 a cantInterfaces-1.

Aliasing Router

En la interfaz del router, los siguientes parámetros sean de tipo in, inout o devueltos como resultado, son pasados por referencia:

- Router
- ReglaDir
- Evento
- DirIp
- Conjunto(α)
- SecuOrdSinRep(α)

El resto de los parámetros, son tipos básicos, por lo cual son pasados por copia.

En el caso de los elementos devueltos como resultado, si no son de alguno de los tipos compuestos aclarados anteriormente, siempre se devuelven por copia.

Aliasing SecuOrdSinRep

En la interfaz de la secuOrdSinRep, cuando se recibe un parámetro de tipo α , si es un tipo básico, se pasa por copia, mientras que si es un tipo complejo, se pasa por referencia.

El resto de los parámetros, se pueden considerar parámetros básicos, por lo cual se pasan por copia.

En el caso de los elementos devueltos como resultado, siempre se devuelven por copia si son de tipos básicos, en caso de devolver una Secuencia, esta se devuelve por referencia.

Aliasing Conjunto

En la interfaz del conjunto, los parámetros recibidos como in o inout de tipo Conjunto, son pasados por referencia.

En el caso de los elementos paramétricos, si el parámetro es un tipo básico, se pasa por copia, mientras que si es complejo, se pasa por referencia.

En el caso de los elementos devueltos como resultado, siempre se devuelven por copia si son de tipos básicos, en caso de devolver un Conjunto, este se devuelve por referencia.

Aliasing Arbol

En la interfaz del arbolDeReglas, los siguientes parámetros sean de tipo in o inout, son pasados por referencia:

- arbolDeReglas
- ReglaDir
- DirIp

En el caso de los elementos devueltos como resultado, siempre se devuelven por copia.

Descripción de funciones utilizadas

Las siguientes funciones fueron utilizadas para describir Ordenes de complejidad o funciones de representación o abstracción. Las mismas están incluidas dentro de los TAD correspondientes, pero no tienen ninguna función análoga dentro de los módulos presentados para este trabajo práctico. Además, la función altura solo se utiliza para la función de representación del Enrutador y rompería la igualdad observacional del TAD arbolDeReglas, por lo cual la misma se encuentra axiomatizada en este apartado.

$$\text{posicionRelativa} : \text{secuOrdSinRep}(\alpha) \times \alpha \longrightarrow \text{nat}$$

$$\text{posicionRelativa} : \text{conj}(\alpha) \times \alpha \longrightarrow \text{nat}$$

$$\text{altura} : \text{ArbolDeReglas} \longrightarrow \text{nat}$$

$$\text{altura}(\text{nuevo}()) \equiv 0$$

$$\text{altura}(\text{agRegla}(a,r)) \equiv \max(\text{cantBits}(r), \text{altura}(r))$$

Modulo Router

Interfaz

interfaz *Router*

se explica con la especificación de *Router*

usa interfaces BOOL, NAT, CONJUNTO(versión), SECUENCIA(EVENTO), REGLADIR, RESPUESTADIR, EVENTO, VERSIÓN, INTERFAZ, DIRIP

genero *router*

operaciones

nuevo(in n: nat) → res: router (O(n))
{true}
{res =_{obs} nuevo(n)}

agVersión(inout r: router, in v: versión) (O(v))
{(r =_{obs} r₀) ∧ (v ∉ versiones(r))}
{r =_{obs} agVersión(r₀, v)}

agRegla(inout r: router, in u: reglaDir) (O(version(dirIp(u))))
{(r =_{obs} r₀) ∧ (versión(dirIp(u)) ∈ versiones(r)) ∧ (inter(u) interfaces(r))}
{r =_{obs} agRegla(r₀, u)}

agEvento(inout r: router, in e :evento) (O(a)¹)
{(r =_{obs} r₀) ∧ (¬∃ i: interfaz)(i < interfaces(r)) ∧_L existeTimestamp?(eventos(r,i),timestamp(e)))}
{r =_{obs} agEvento(r₀, e)}

versiones(in r: router) → res: conj(versión) (O(1))
{true}
{res =_{obs} versiones(r)}

interfaces(in r: router) → res: nat (O(1))
{true}
{res =_{obs} interfaces(r)}

enrutar(in r: router, in d: dirIp) → res: respDir (O(version(d)))
{true}
{res =_{obs} enrutar(r, d)}

eventos(in r: router, in i: interfaz) → res: secuOrdSinRep(eventos) (O(1))
{true}
{res =_{obs} eventos(r, i)}

estaCaida?(in r: router, in i: interfaz) → res: bool (O(1))
{i < interfaces(r)}

¹a = longitud(eventos(r,inter(e))) - posicionRelativa(eventos(r,inter(e)), e)

$\{res =_{\text{obs}} \text{estaCaida}(r,i)\}$

$\text{tiempoCaida}(\text{in } r: \text{router}, \text{in } i: \text{interfaz}) \longrightarrow res: \text{nat} \quad (O(1))$
 $\{i < \text{interfaces}(r)\}$
 $\{res =_{\text{obs}} \text{tiempocaida}(r,i)\}$

Representación

enrutador se representa con estr

donde estr es tupla \langle *versiones: conj(version)*
 \times *cantInterfaces: nat*
 \times *reglas: secuOrdSinRep(tupla(version: version, aR: abr))*
 \times *status_inter: ad(tupla(tiempoCaida: nat, eventos: secuOrdSinRep(evento)))* \rangle

Invariante de Representación

$\text{Rep} : \text{estr} \longrightarrow \text{bool}$

$(\forall e: \text{estr}) \text{Rep}(e) = \text{versionesDeReglasValidas}(e.\text{reglas}, e.\text{versiones})$
 $\wedge \text{noFaltanVersionesEnReglas}(e)$
 $\wedge_L \text{interfacesDeArbolesValidas}(e.\text{reglas}, e.\text{cantInterfaces})$
 $\wedge \text{alturaDeArbolesValida}(e.\text{reglas})$
 $\wedge \text{longitudDeEventosValida}(e.\text{status_inter}, e.\text{cantInterfaces})$
 $\wedge_L \text{interfazDeEventosCorrecta}(e.\text{status_inter})$
 $\wedge_L \text{tiempoCaidaValido}(e.\text{status_inter})$

funciones auxiliares:

$\text{versionesDeReglasValidas} : \text{secuOrdSinRep}(\text{tupla}(\text{version: version} \times \text{aR: abr})) \times \text{conj}(\text{version}) \longrightarrow \text{bool}$

$\text{versionesDeReglasValidas}(s,c) \equiv \text{if } \text{vacía?}(s) \text{ then } \text{true}$
 $\text{else } \text{prim}(s).\text{version} \in c \wedge \text{versionesDeReglasValidas}(\text{fin}(s),c)$

$\text{noFaltanVersionesEnReglas} : \text{estr} \longrightarrow \text{bool}$

$\text{noFaltanVersionesEnReglas}(e) \equiv (\forall v: \text{version})(v \in e.\text{versiones} \Rightarrow_L \text{estaVersionEnTupla}(v,e.\text{reglas}))$

$\text{interfacesDeArbolesValidas} : \text{secuOrdSinRep}(\text{tupla}(\text{version: version} \times \text{aR: abr})) \times \text{nat} \longrightarrow \text{bool}$

$\text{interfacesDeArbolesValidas}(s,n) \equiv (\forall d: \text{dirIp}) (\text{estaVersionEnTupla}(\text{version}(d),s) \Rightarrow_L$
 $\text{interValidasAux}(s,n,d))$

$\text{estaVersionEnTupla} : \text{version} \times \text{secuOrdSinRep}(\text{tupla}(\text{version: version} \times \text{aR: abr})) \longrightarrow \text{bool}$

$\text{estaVersionEnTupla}(v,s) \equiv \text{if } \text{vacía?}(s) \text{ then } \text{false}$
 $\quad \text{else } \text{prim}(s).\text{version} = v \vee \text{estaVersionEnTupla}(v,\text{fin}(s))$

$\text{interValidasAux} : \text{secuOrdSinRep}(\text{tupla}(\langle \text{version:version} \times \text{aR:abr} \rangle)) s \times \text{nat} \times \text{dirIp } d \longrightarrow \text{bool}$
 $\quad \text{estaVersionEnTupla}(\text{version}(d),s)$

$\text{interValidasAux}(s,n,d) \equiv \text{if } \text{prim}(s).\text{version} = \text{version}(d)$
 $\quad \text{then } \text{interfazDeSalida}(\text{prim}(s).\text{aR},d) < n$
 $\quad \text{else } \text{interValidasAux}(\text{fin}(s),n,d)$

$\text{alturaDeArbolesValida} : \text{secuOrdSinRep}(\text{tupla}(\langle \text{version:version} \times \text{aR:abr} \rangle)) \longrightarrow \text{bool}$

$\text{alturaDeArbolesValida}(s) \equiv \text{if } \text{vacía?}(s) \text{ then } \text{true}$
 $\quad \text{else } 8 * \text{prim}(s).\text{version} \geq \text{altura}(\text{prim}(s).\text{aR})$
 $\quad \wedge \text{alturaDeArbolesValida}(\text{fin}(s))$

$\text{interfazDeEventosCorrecta} : \text{ad}(\text{tupla}(\text{nat} \times \text{secuOrdSinRep}(\text{evento}))) \longrightarrow \text{bool}$

$\text{interfazDeEventosCorrecta}(a) \equiv (\forall i:\text{interfaz}) (i < \text{tam}(a) \Rightarrow_{\text{L}} \text{sonEventosDeLaInterfaz}(a[i].\text{eventos},i))$

$\text{sonEventosDeLaInterfaz} : \text{secuOrdSinRep}(\text{evento}) \times \text{interfaz} \longrightarrow \text{bool}$

$\text{sonEventosDeLaInterfaz}(s,i) \equiv \text{if } \text{vacía?}(s) \text{ then } \text{true}$
 $\quad \text{else } \text{inter}(\text{prim}(s)) = i \wedge \text{sonEventosDeLaInterfaz}(\text{fin}(s),i)$

$\text{tiempoCaidaValido} : \text{ad}(\text{tupla}(\text{nat} \times \text{secu}(\text{evento}))) \longrightarrow \text{bool}$

$\text{tiempoCaidaValido}(a) \equiv (\forall i:\text{interfaz}) (i < \text{tam}(a) \Rightarrow_{\text{L}} a[i].\text{tiempoCaida} = \text{sumarCaidas}(a[i].\text{eventos}))$

$\text{sumarCaidas} : \text{secuOrdSinRep}(\text{evento}) \longrightarrow \text{nat}$

$\text{sumarCaidas}(s) \equiv \text{if } \text{vacía?}(s) \vee_{\text{L}} \text{vacía?}(\text{fin}(s)) \text{ then } 0$
 $\quad \text{else if } \text{caída?}(\text{ult}(s))$
 $\quad \quad \text{then } \text{timestamp}(\text{ult}(\text{com}(s))) - \text{timestamp}(\text{ult}(s)) + \text{sumarCaidas}(\text{com}(s))$
 $\quad \text{else } \text{sumarCaidas}(\text{com}(s))$

$\text{longitudDeEventosValida} : \text{ad}(\text{tupla}(\text{nat} \times \text{secuOrdSinRep}(\text{evento}))) \times \text{nat} \longrightarrow \text{bool}$

$\text{longitudDeEventosValida}(a,n) \equiv \text{tam}(a) = n$

Función de Abstracción

Abs : estr $e \longrightarrow$ enrutador Rep(e)

($\forall e$: estr) Abs(e) = r : router/
 versiones(r) =_{obs} e .versiones \wedge interfaces(r) =_{obs} e .cantInterfaces \wedge_L
 ($\forall i$: interfaz)($i < \text{interfaces}(r) \Rightarrow_L \text{eventos}(r,i) =_{\text{obs}} (e.\text{status_inter}[i]).\text{eventos}$) \wedge
 ($\forall d$: dirIp)(enrutar(r,d) =_{obs} dameRtaDir(e,d))

funciones auxiliares

dameRtaDir : estr $e \times \text{dirIp} \longrightarrow \text{respuestaDir}$ Rep(e)

dameRtaDir(e,d) \equiv **if** version(d) $\notin e$.versiones **then** DireccionNoSoportada()
 else if $\neg \text{tieneRegla}(\text{arbolDeLaVersion}(\text{version}(d),e.\text{reglas}),d)$
 then interfazDeSalidaNoEncontrada()
 else if estaCaida?(prim(($e.\text{status_inter}[\text{queInterfaz}(e,d)]$)).eventos))
 then interfazDeSalidaCaida(queInterfaz(e,d))
 else SalidaPorInterfaz(queInterfaz(e,d))

queInterfaz : estr $e \times \text{dirIp} \longrightarrow \text{interfaz}$ Rep(e)

queInterfaz(e,d) \equiv interfazDeSalida(arbolDeLaVersion(version(d), e .reglas), d)

arbolDeLaVersion : version $v \times \text{secu}(\text{tupla}(\text{version} \times \text{abr})) s \longrightarrow \text{abr}$

arbolDeLaVersion(v,s) \equiv **if** vacia?(s) **then** nuevo()
 else if prim(s).version = v **then** prim(s).aR
 else arbolDeLaVersion($v,\text{fin}(s)$)

Algoritmos

$iNuevo(in\ n: nat) \rightarrow res: estr$

```

1 Complejidad:  $O(n)$ 
2 var inter: arreglo_dimensionable de tupla(nat, secuOrdSinRep(evento))
3 inter  $\leftarrow$  CrearArreglo(n); // 0(n)
4 for ( $i: nat \leftarrow 0; i < n; i++$ ) ; // 0(n)
5 do
6   | inter[i]  $\leftarrow$  (0, <>); // 0(1)
7 end
8 res  $\leftarrow$  ( $\emptyset$ , n, <>, inter); // 0(1)

```

$iagVersion(inout\ e: estr, in\ v: version)$

```

1 Complejidad:  $O(v)$ 
2 agregar(v, e.versiones); // 0(v)a
3 insertarOrdenadoDesdeAdelante((v, <>), e.reglas); // 0(v)b

```

^aLa complejidad es del orden de $posicionRelativa(e.versiones, v)$ como lo indica la interfaz de Conjunto, pero en el peor caso, $posicionRelativa(e.versiones, v) = v$

^bLa complejidad es del orden de $posicionRelativa(e.reglas, (v, <>)) = v$

$iagRegla(inout\ e: estr, in\ u: reglaDir)$

```

1 Complejidad:  $O(version(dirIp(u)))$ 
2 var it: itOrd(secuOrdSinRep(tupla (version, abr)))
3 var v: version
4 it  $\leftarrow$  crearIt(e.reglas) ; // 0(1)
5 v  $\leftarrow$  u.version ; // 0(1)
6 while ( $actualAdelante(it).version \neq v$ ) ; // 0(version(dirIp(u)))a
7 do
8   | avanzar(it)
9 end
10 agRegla(actualAdelante(it).aR, u) ; // 0(cantBits(u))b

```

^aEl ciclo se ejecuta tantas veces como la cantidad de versiones menores a la de la regla que estoy agregando haya en las tuplas de e.reglas. Por lo tanto, al no haber versiones repetidas, como máximo se va a recorrer $version(dirIp(u))$ veces la secuencia

^bEste orden de complejidad temporal se obtiene de la interfaz del Módulo ArbolDeReglas

iagEvento(inout e: estr, in m: evento)

```

1  Complejidad:  $O(a)^a$ 
2  var i: interfaz
3  var tc: nat
4  var evento_prev,evento_post: evento
5  var it: itOrd(secuOrdSinRep(evento))

6  i  $\leftarrow$  m.inter ; // 0(1)
7  tc  $\leftarrow$  (e.status_inter[i]).tiempoCaida ; // 0(1)
8  insertarOrdenadoDesdeAtras(m, (e.status_inter[i]).eventos) ; // 0(a)
9  it  $\leftarrow$  crearIt((e.status_inter[i]).eventos) ; // 0(1)
10 if (actualAtras(it) = m)  $\wedge$  tieneAnterior?(it); // 0(1)
11 then
12   retroceder(it) ; // 0(1)
13   if (actualAtras(it).estaCaida?); // 0(1)
14   then
15     tc  $\leftarrow$  tc + (m.timestamp - actualAtras(it).timestamp); // 0(1)
16   end
17 end
18 else
19   if (tieneAnterior?(it)?); // 0(1)
20   then
21     while (actualAtras(it)  $\neq$  m); // 0(a)b
22     do
23       evento_post  $\leftarrow$  actualAtras(it) ; // 0(1)
24       retroceder(it); // 0(1)
25     end
26     if ( $\neg$ tieneAnterior?(it)); // 0(1)
27     then
28       if (m.estaCaida?); // 0(1)
29       then
30         tc  $\leftarrow$  tc + (evento_post.timestamp - m.timestamp); // 0(1)
31       end
32     end
33     else
34       retroceder(it); // 0(1)
35       evento_prev  $\leftarrow$  actualAtras(it); // 0(1)
36       if (m.estaCaida?  $\wedge$   $\neg$ evento_prev.estaCaida?); // 0(1)
37       then
38         tc  $\leftarrow$  tc + (evento_post.timestamp - m.timestamp); // 0(1)
39       end
40       if ( $\neg$ m.estaCaida?  $\wedge$  evento_prev.estaCaida?); // 0(1)
41       then
42         tc  $\leftarrow$  tc - (evento_post.timestamp - m.timestamp); // 0(1)
43       end
44     end
45   end
46 end
47 (e.status_inter[i]).tiempoCaida  $\leftarrow$  tc; // 0(1)

```

^aa = longitud(e.status_inter[i].eventos) - posicionRelativa(e.status_inter[i].eventos,m)

^bEl ciclo se ejecuta a lo sumo “a” veces, dado que la secuencia se recorre de atrás hacia adelante. Es decir, que el costo temporal del ciclo es igual a la cantidad de eventos en la secuencia con timestamp mayor que el del evento que quiero agregar

$i\text{versiones}(\text{in } e: \text{estr}) \rightarrow \text{res: conj}(\text{version})$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  e.versiones; // 0(1)

```

$i\text{interfaces}(\text{in } e: \text{estr}) \rightarrow \text{res: nat}$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  e.cantInterfaces ; // 0(1)

```

$i\text{enrutar}(\text{inout } e: \text{estr}, \text{in } d: \text{dirIp}) \rightarrow \text{res: RespuestaDir}$

```

1 Complejidad:  $O(\text{version}(d))$ 
2 if ( $\neg \text{pertenece}(\text{version}(d), e.\text{versiones})$ );
   // 0(posicionRelativa(e.versiones, version(v)))a
3 then
4   | res  $\leftarrow$  direccionNoSoportada(); // 0(1)
5 else
6   | var it: itOrd(secuOrdSinRep(tupla(version, abr)))
7   | var i: interfaz
8   | it  $\leftarrow$  crearIt(e.reglas) ; // 0(1)
9   | while ( $\text{actualAdelante}(it).\text{version} \neq \text{version}(d)$ ) ; // 0(version(d))b
10  | do
11    | avanzar(it); // 0(1)
12  | end
13  | if ( $\neg \text{tieneRegla}(\text{actualAdelante}(it).aR, d)$ ) ; // 0(version(d))c
14  | then
15    | res  $\leftarrow$  interfazDeSalidaNoEncontrada(); // 0(1)
16  | else
17    | i  $\leftarrow$  interfazDeSalida(actualAdelante(it).aR, d) ; // 0(version(d))d
18    | if ( $\text{estaCaida?}(e, i)$ ); // 0(1)
19    | then
20      | res  $\leftarrow$  interfazDeSalidaCaida(i); // 0(1)
21    | else
22      | res  $\leftarrow$  salidaPorInterfaz(i); // 0(1)
23    | end
24  | end
25 end

```

^aCumple esta complejidad por la información que aparece en la interfaz Conjunto

^bLa complejidad se da porque avanzamos sobre la secuencia de versiones hasta encontrar las reglas que corresponden con la versión de la dirIp y éstas están ordenadas

^cCumple esta complejidad por la información que aparece en la interfaz de ArbolDeReglas

^dCumple esta complejidad por la información que aparece en la interfaz de ArbolDeReglas

$i\text{eventos}(\text{in } e: \text{estr}, \text{in } i: \text{interfaz}) \rightarrow \text{res: secuOrdSinRep}(\text{evento})$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  (e.status_inter[i]).eventos; // 0(1)

```

$i\text{estaCaida?}(\text{in } e: \text{estr}, \text{in } i: \text{interfaz}) \longrightarrow \text{res: bool}$

```
1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  false ; // 0(1)
3 if ( $\neg \text{vacía?}((e.\text{status\_inter}[i]).\text{eventos})$ ); // 0(1)
4 then
5   var it: itOrd(secuOrdSinRep(tupla  $\langle \text{nat}, \text{evento} \rangle$ ))
6   it  $\leftarrow$  crearIt((e.status_inter[i]).eventos) ; // 0(1)
7   res  $\rightarrow$  actualAdelante(it).estaCaida? ; // 0(1)
8 end
```

$i\text{tiempoCaida}(\text{in } e: \text{estr}, \text{in } i: \text{interfaz}) \longrightarrow \text{res: nat}$

```
1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  (e.status_inter[i]).tiempoCaida ; // 0(1)
```

Modulo SecuenciaOrdSinRep($\alpha, <_\alpha, =_\alpha$)

Interfaz

interfaz *SecuenciaOrdSinRep*(α)

se explica con la especificación de *Secuencia*(α) *extendido*, *itOrd*(α)

usa interfaces **BOOL**, α , **ITORD**(α)

genero *secuOrdSinRep*(α), *itOrd*(α)

operaciones

$<>() \longrightarrow \text{res: secuOrdSinRep}(\alpha)$ (O(1))

{true}

{res =_{obs} <>}

insertarOrdenadoDesdeAdelante(in a: α , inout s: secuOrdSinRep(α)) (O(posicionRelativa(s,a)))

{s =_{obs} s₀ $\wedge \neg \text{esta?}(a, s)$ }

{s =_{obs} agregarOrdenado(a, s₀)}

insertarOrdenadoDesdeAtras(in a: α , inout s: secuOrdSinRep(α)) (O(longitud(s) - posicionRelativa(s,a)))

{s =_{obs} s₀ $\wedge \neg \text{esta?}(a, s)$ }

{s =_{obs} agregarOrdenado(a, s₀)}

vacia?(in s: secuOrdSinRep(α)) $\longrightarrow \text{res: bool}$ (O(1))

{true}

{res =_{obs} vacia?(a, s)}

esta?(in a: α , in s: secuOrdSinRep(α)) $\longrightarrow \text{res: bool}$ (O(posicionRelativa(s,a)))

{true}

{res =_{obs} esta?(a, s)}

crearIt(in s: secuOrdSinRep(α)) $\longrightarrow \text{res: itOrd}(\alpha)$ (O(1))

{true}

{res =_{obs} crearIt(s)}

actualAdelante(in it: itOrd(α)) $\longrightarrow \text{res: } \alpha$ (O(1))

{ $\neg \text{vacia?}(\text{verSecuSuby(it)})$ }

{res=_{obs}prim(verSecuSuby(it))}

actualAtras(in it: itOrd(α)) $\longrightarrow \text{res: } \alpha$ (O(1))

{ $\neg \text{vacia?}(\text{verSecuSuby(it)})$ }

{res=_{obs}ult(verSecuSuby(it))}

tieneProximo?(in it: itOrd(α)) $\longrightarrow \text{res: bool}$ (O(1))

{ $\neg \text{vacia?}(\text{verSecuSuby(it)})$ }

{res =_{obs} $\neg \text{vacia?}(\text{fin}(\text{verSecuSuby(it))))$ }

tieneAnterior?(in it: itOrd(α)) \longrightarrow res: bool (O(1))
 $\{\neg \text{vacía}?(verSecuSuby(it))\}$
 $\{res =_{obs} \neg \text{vacía}?(com(verSecuSuby(it)))\}$

avanzar(inout it: itOrd(α)) (O(1))
 $\{it =_{obs} it_0 \wedge \neg \text{vacía}?(fin(verSecuSuby(it)))\}$
 $\{verSecuSuby(it) =_{obs} fin(verSecuSuby(it_0))\}$

retroceder(inout it: itOrd(α)) (O(1))
 $\{it =_{obs} it_0 \wedge \neg \text{vacía}?(com(verSecuSuby(it)))\}$
 $\{verSecuSuby(it) =_{obs} com(verSecuSuby(it_0))\}$

Representación

secuOrdSinRep(α) se representa con estrSecuOrdSinRep
donde *estrSecuOrdSinRep es tupla* $\langle \text{prim: puntero(nodo), ult: puntero(nodo)} \rangle$
donde *nodo es tupla* $\langle \text{dato: } \alpha, \text{proximo: puntero(nodo), anterior: puntero(nodo)} \rangle$

itOrd(α) se representa con estrIterOrd
donde *estrIterOrd es tupla* $\langle \text{prim: puntero(nodo), ult: puntero(nodo)} \rangle$

Invariante de Representación

Rep : estrSecuOrdSinRep \longrightarrow bool

$(\forall e: \text{estrSecuOrdSinRep}) \text{Rep}(e) = \text{No hay ciclos en la secuencia} \wedge \text{La secuencia está ordenada} \wedge \text{No hay repetidos en la secuencia} \wedge \text{Recorriendo desde el primero de la secuencia se puede llegar al ultimo y viceversa.}$

Rep : estrIterOrd \longrightarrow bool

$(\forall e: \text{estrIterOrd}) \text{Rep}(e) = \text{No hay ciclos en la secuencia} \wedge \text{La secuencia está ordenada} \wedge \text{No hay repetidos en la secuencia} \wedge \text{Recorriendo desde el primero de la secuencia se puede llegar al ultimo y viceversa.}$

Función de Abstracción

Abs : estrSecuOrdSinRep $e \longrightarrow \text{secuOrdSinRep}(\alpha)$ Rep(e)

$(\forall e: \text{estrSecuOrdSinRep}) \text{Abs}(e) = s: \text{secuOrdSinRep}(\alpha) /$
 $(\text{vacía}?(s) \equiv (e.\text{prim} == \text{NULL}) \wedge (e.\text{ult} == \text{NULL})) \wedge_L \neg \text{vacía}?(s) \Rightarrow_L ((\text{prim}(s) \equiv (e.\text{prim} \rightarrow \text{dato})) \wedge$
 $(\text{fin}(s) \equiv (\text{final}(e))))$

funciones auxiliares

$\text{final} : \text{estr} \longrightarrow \text{estr}$

$\text{final}(e) \equiv \langle (e.\text{prim}) \rightarrow \text{proximo}, e.\text{ult} \rangle$

$\text{Abs} : \text{estrIterOrd } e \longrightarrow \text{itOrd}(\alpha)$

$\text{Rep}(e)$

$(\forall e: \text{estrIterOrd}) \text{ Abs}(e) = \text{it: itOrdRef}(\alpha) / \text{verSecuSuby}(\text{it}) \equiv \text{Abs}_{\text{estrSecuOrdSinRep}}(e)$

Algoritmos

$i<>() \rightarrow \text{res:estrSecuOrdSinRep}$

1 Complejidad: $O(1)$	
2 $\text{res} \leftarrow \langle \text{NULL}, \text{NULL} \rangle$;	// 0(1)

$i\text{insertarOrdenadoDesdeAdelante}(\text{in elem: } \alpha, \text{ inout e: estrSecuOrdSinRep})$

1 Complejidad: $O(\text{posicionRelativa}(e, \text{elem}))$	
2 var n: nodo	
3 $\text{n.dato} \leftarrow \text{elem}$;	// 0(1)
4 if $(\neg \text{vacía?}(e))$;	// 0(1)
5 then	
6 if $((\text{*e.prim}).\text{dato} >_{\alpha} \text{elem} \vee (\text{*e.ult}).\text{dato} <_{\alpha} \text{elem})$;	// 0(1)
7 then	
8 if $((\text{*e.prim}).\text{dato} >_{\alpha} \text{elem})$;	// 0(1)
9 then	
10 $(\text{*e.prim}).\text{anterior} \leftarrow \&\text{n}$;	// 0(1)
11 $\text{n.anterior} \leftarrow \text{NULL}$;	// 0(1)
12 $\text{n.proximo} \leftarrow \text{e.prim}$;	// 0(1)
13 $\text{e.prim} \leftarrow \&\text{n}$;	// 0(1)
14 else	
15 if $((\text{*e.prim}).\text{dato} <_{\alpha} \text{elem})$;	// 0(1)
16 then	
17 $(\text{*e.ult}).\text{proximo} \leftarrow \&\text{n}$;	// 0(1)
18 $\text{n.anterior} \leftarrow \text{e.ult}$;	// 0(1)
19 $\text{n.proximo} \leftarrow \text{NULL}$;	// 0(1)
20 $\text{e.ult} \leftarrow \&\text{n}$;	// 0(1)
21 end	
22 end	
23 else	
24 var aux: puntero(nodo);	// 0(1)
25 $\text{aux} \leftarrow \text{e.prim}$;	// 0(1)
26 while $((\text{*(*aux).proximo}).\text{dato} <_{\alpha} \text{elem})$;	// 0(posicionRelativa(e,elem)) ^a
27 do	
28 $\text{aux} \leftarrow (\text{*aux}).\text{proximo}$;	// 0(1)
29 end	
30 $(\text{*(*aux).proximo}).\text{anterior} \leftarrow \&\text{n}$;	// 0(1)
31 $\text{n.proximo} \leftarrow (\text{*aux}).\text{proximo}$;	// 0(1)
32 $\text{n.anterior} \leftarrow \text{aux}$;	// 0(1)
33 $(\text{*aux}).\text{proximo} \leftarrow \&\text{n}$;	// 0(1)
34 end	
35 else	
36 $\text{e.prim} \leftarrow \&\text{n}$;	// 0(1)
37 $\text{e.ult} \leftarrow \&\text{n}$;	// 0(1)
38 $\text{n.anterior} \leftarrow \text{NULL}$;	// 0(1)
39 $\text{n.proximo} \leftarrow \text{NULL}$;	// 0(1)
40 end	

^aEl ciclo se ejecuta tantas veces como la cantidad de elementos menores al que estoy agregando haya en la secuencia. Por lo tanto, al no haber elementos repetidos, como máximo se va a recorrer $\text{posicionRelativa}(e, \text{elem})$ veces

insertarOrdenadoDesdeAtras(in elem: α , inout e: *estrSecuOrdSinRep*)

```

1  Complejidad:  $O(a)^a$ 
2  var n: nodo
3  n.dato  $\leftarrow$  elem ;                                // 0(1)
4  if ( $\neg$  vacía?(e));                                // 0(1)
5  then
6      if ((*e.prim).dato  $>_{\alpha}$  elem  $\vee$  (*e.ult).dato  $<_{\alpha}$  elem); // 0(1)
7      then
8          if ((*e.prim).dato  $>_{\alpha}$  elem);                // 0(1)
9          then
10             (*e.prim).anterior  $\leftarrow$  &n;           // 0(1)
11             n.anterior  $\leftarrow$  NULL;                 // 0(1)
12             n.proximo  $\leftarrow$  e.prim;                // 0(1)
13             e.prim  $\leftarrow$  &n;                       // 0(1)
14         else
15             if ((*e.prim).dato  $<_{\alpha}$  elem);           // 0(1)
16             then
17                 (*e.ult).proximo  $\leftarrow$  &n;          // 0(1)
18                 n.anterior  $\leftarrow$  e.ult;             // 0(1)
19                 n.proximo  $\leftarrow$  NULL;               // 0(1)
20                 e.ult  $\leftarrow$  &n;                     // 0(1)
21             end
22         end
23     else
24         var aux: puntero(nodo);                        // 0(1)
25         aux  $\leftarrow$  e.ult;                            // 0(1)
26         while ((*aux).anterior).dato  $>_{\alpha}$  elem);    // 0(a)b
27         do
28             aux  $\leftarrow$  (*aux).anterior;              // 0(1)
29         end
30         (*aux).anterior).proximo  $\leftarrow$  &n;           // 0(1)
31         n.anterior  $\leftarrow$  (*aux).anterior;           // 0(1)
32         n.proximo  $\leftarrow$  aux;                        // 0(1)
33         (*aux).anterior  $\leftarrow$  &n;                   // 0(1)
34     end
35 else
36     e.prim  $\leftarrow$  &n;                                // 0(1)
37     e.ult  $\leftarrow$  &n;                                // 0(1)
38     n.anterior  $\leftarrow$  NULL;                          // 0(1)
39     n.proximo  $\leftarrow$  NULL;                          // 0(1)
40 end

```

^aa = longitud(e) - posicionRelativa(e,elem)

^bEl ciclo se ejecuta tantas veces como la cantidad de elementos mayores al que estoy agregando haya en la secuencia. Por lo tanto, al no haber elementos repetidos, como máximo se va a recorrer longitud(e) - posicionRelativa(e,elem) veces

ivacia?(in e: *estrSecuOrdSinRep*) \rightarrow res: bool

```

1  Complejidad:  $O(1)$ 
2  res  $\leftarrow$  (e.prim == NULL)

```


$i\text{esta?}(\text{in } a: \alpha, e: \text{estrSecuOrdSinRep}) \longrightarrow \text{res: bool}$

```

1 Complejidad:  $O(\text{posicionRelativa}(e,a))$ 
2 res  $\leftarrow$  false ; // 0(1)
3 if ( $\neg \text{vacía?}(e)$ ) ; // 0(1)
4 then
5   var it: itOrd(secuOrdSinRep( $\alpha$ ))
6   it  $\leftarrow$  crearIt(e) ; // 0(1)
7   while ( $\text{tieneProximo}(it)? \wedge \text{actualAdelante}(it) <_{\alpha} a$ ); // 0(posicionRelativa(e,a))a
8   do
9     | avanzar(it); // 0(1)
10  end
11  if ( $a =_{\alpha} \text{actualAdelante}(it)$ ); // 0(1)
12  then
13    | res  $\leftarrow$  true; // 0(1)
14  end
15 end

```

^aLa secuencia se recorre hasta que se encuentra al elemento buscado, o se encuentre uno mayor o no queden más elementos por recorrer. Por esto es que la complejidad del peor caso es $\text{posicionRelativa}(e,a)$

$i\text{crearIT}(\text{in } e: \text{estrSecuOrdSinRep}) \longrightarrow \text{res: estrIterOrd}$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  e

```

$i\text{actualAdelante}(\text{in } it: \text{estrIterOrd}) \longrightarrow \text{res: } \alpha$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  ((it.prim)  $\rightarrow$  dato)

```

$i\text{actualAtras}(\text{in } it: \text{estrIterOrd}) \longrightarrow \text{res: } \alpha$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  ((it.ult)  $\rightarrow$  dato)

```

$i\text{tieneProximo?}(\text{in } it: \text{estrIterOrd}) \longrightarrow \text{res: bool}$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  ((it.prim)  $\rightarrow$  proximo  $\neq$  NULL)

```

$i\text{tieneAnterior?}(\text{in } it: \text{estrIterOrd}) \longrightarrow \text{res: bool}$

```

1 Complejidad:  $O(1)$ 
2 res  $\leftarrow$  ((it.ult)  $\rightarrow$  anterior  $\neq$  NULL)

```

$i\text{avanzar}(\text{inout } it: \text{estrIterOrd})$

```

1 Complejidad:  $O(1)$ 
2 it.prim  $\leftarrow$  ((it.prim)  $\rightarrow$  proximo)

```

iretroceder(inout it: *estrIterOrd*)

- 1 **Complejidad:** $O(1)$
- 2 $\text{it.ult} \leftarrow ((\text{it.ult}) \rightarrow \text{anterior})$

Modulo Conjunto(α)

Interfaz

interfaz *Conjunto*(α)

se explica con la especificación de *Conjunto*(α)

usa interfaces BOOL, NAT

genero *conj*(α)

operaciones

Vacio() $\longrightarrow res: conj(\alpha)$ (O(1))
{true}
{ $res =_{\text{obs}} \phi()$ }

agregar(in $a: \alpha$, inout $C: conj(\alpha)$) (O(posicionRelativa(C,a)))
{ $C =_{\text{obs}} C_0$ }
{ $C =_{\text{obs}} Ag(a, C_0)$ }

pertenece(in $a: \alpha$, in $C: conj(\alpha)$) $\longrightarrow res: bool$ (O(posicionRelativa(C,a)))
{true}
{ $res =_{\text{obs}} a \in C$ }

Representación

conjunto(α) **se representa con** *estrConj*

donde *estrConj* es *SecuOrdSinRep*(α)

Invariante de Representación

$Rep : estrConj \longrightarrow bool$

$(\forall e: estrConj) Rep(e) = true$

Función de Abstracción

$Abs : estrConj \ e \longrightarrow conj(\alpha)$ Rep(e)

$(\forall e: estrConj) Abs(e) = C: conj(\alpha) / (\forall a: \alpha)(a \in C =_{\text{obs}} esta?(a, e))$

Algoritmos

$iVacio() \rightarrow res: SecuOrdSinRep(\alpha)$

```
1 Complejidad:  $O(1)$   
2  $res \leftarrow \langle \rangle$ 
```

$iagregar(in\ a: \alpha, inout\ s: SecuOrdSinRep(\alpha))$

```
1 Complejidad:  $O(posicionRelativa(s,a))$   
2 if  $\neg\ esta?(a,s);$  //  $O(posicionRelativa(s,a))^a$   
3 then  
4 |  $insertarOrdenadoDesdeAdelante(a,s);$  //  $O(posicionRelativa(s,a))^b$   
5 end
```

^aEsta complejidad se cumple por la información que brinda la interfaz de SecuOrdSinRep

^bEsta complejidad se cumple por la información que brinda la interfaz de SecuOrdSinRep

$ipertenece(in\ a: \alpha, in\ s: SecuOrdSinRep(\alpha))$

```
1 Complejidad:  $O(posicionRelativa(s,a))^a$   
2  $res \leftarrow\ esta?(a,s)$ 
```

^aEsta complejidad se cumple por la información que brinda la interfaz de SecuOrdSinRep

Modulo ArbolDeReglas

Interfaz

interfaz *arbolDeReglas*
se explica con la especificación de *arbolDeReglas*
usa interfaces BOOL, NAT, DIRIP, REGLADIR
genero *abr*

operaciones

$Crear() \longrightarrow res: abr$ (O(1))
 $\{true\}$
 $\{res =_{obs} nuevo()\}$

$agRegla(inout a: abr, in r: regla)$ (O(version(r)))
 $\{a =_{obs} a_0\}$
 $\{a =_{obs} agRegla(a_0, r)\}$

$interfazDeSalida(in a: abr, in d: dirIp) \longrightarrow res: nat$ (O(version(d)))
 $\{tieneRegla?(a, d)\}$
 $\{res =_{obs} interfazDeSalida(a, d)\}$

Representación

arbolDeReglas **se representa con** *puntero(estr)*
donde *estr* **es**
 $tupla \langle inter: puntero(interfaz)$
 $\times izq: puntero(estr)$
 $\times der: puntero(estr) \rangle$

Invariante de Representación

$Rep : puntero(estr) \longrightarrow boolean$

$(\forall p: puntero(estr)) Rep(p) = true \Leftrightarrow \text{no existen ciclos}$

Función de Abstracción

$\text{Abs} : \text{puntero}(\text{estr}) \ p \longrightarrow \text{abr} \quad \text{Rep}(p)$

$(\forall p: \text{puntero}(\text{estr})) \ \text{Abs}(p) = a: \text{abr}/$

$(\forall d: \text{dirIp}) \ ((\text{tieneRegla}(a,d) = \text{existeRegla}(p,\text{pasarABits}(d),0) \wedge_L$

$(\text{tieneRegla}(a,d) \Rightarrow_L (\text{interfazDeSalida}(a,d) = \text{dameSalida}(p,\text{pasarABits}(d),0,\text{null}))))$

$\text{existeRegla} : \text{puntero}(\text{estr}) \ p \times \text{arregloDeBool} \times \text{nat} \longrightarrow \text{bool} \quad \text{Rep}(p)$

$\text{existeRegla}(p,a,n) \equiv \text{if } n = \text{tam}(a) \vee p = \text{null} \text{ then } false$
 $\quad \text{else if } (*p).\text{inter} = \text{null} \text{ then}$
 $\quad \quad \text{if } a[n] = true \text{ then } \text{existeRegla}((*e).\text{der},a,n+1)$
 $\quad \quad \text{else } \text{existeRegla}((*e).\text{izq},a,n+1)$
 $\quad \quad \text{fi}$
 $\quad \text{else } true$
 $\quad \text{fi}$

$\text{dameSalida} : \text{puntero}(\text{estr}) \ p \times \text{arregloDeBool} \ a \times \text{nat} \times \text{puntero}(\text{estr}) \longrightarrow \text{interfaz}$
 $\quad \quad \text{Rep}(p) \wedge_L \text{existeRegla}(p,a,0)$

$\text{dameSalida}(p,a,n,\text{pi}) \equiv \text{if } n = \text{tam}(a) \vee p = \text{null} \text{ then } *pi$
 $\quad \text{else if } a[n] = true \text{ then}$
 $\quad \quad \text{if } pi \neq \text{null} \text{ then } \text{dameSalida}((*p).\text{der},a,n+1,pi)$
 $\quad \quad \text{else } \text{dameSalida}((*p).\text{der},a,n+1,(*p).\text{int})$
 $\quad \quad \text{fi}$
 $\quad \text{else}$
 $\quad \quad \text{if } pi \neq \text{null} \text{ then } \text{dameSalida}((*p).\text{izq},a,n+1,pi)$
 $\quad \quad \text{else } \text{dameSalida}((*p).\text{izq},a,n+1,(*p).\text{int})$
 $\quad \quad \text{fi}$
 $\quad \text{fi}$

Aclaración: dameSalida funciona dado que asumimos que el árbol se poda cada vez que agregamos una regla como lo indica el algoritmo

Algoritmos

$icrear() \rightarrow res : \text{puntero}(\text{estr})$

1 Complejidad: $O(1)$

2 $res \leftarrow \text{NULL}$

Algorithm 1: crear

iagRegla(inout p : puntero(estr),in r : regla)

```

1  Complejidad:  $O(version(dirIp(r)))$ 
2  var i : int
3  var pAux : puntero(estr)
4  var a : arreglo_dimensionable de bool
5  var x : Interfaz
6  a ← CrearArreglo(version(dirIp(r))*8) ; // 0(version(dirIp(r)))
7  if p = NULL ; // 0(1)
8  then
9  | p ← & ⟨NULL,NULL,NULL⟩ ; // 0(1)
10 end
11 a ← pasarABits(dirIP(r)) ; // 0(version(dirIp(r)))
12 pAux ← p ; // 0(1)
13 i ← 0 ; // 0(1)
14 while i ≠ cantBits(r) ; // 0(cantBits(r))
15 do
16 | if a[i] = true ; // 0(1)
17 | then
18 | | if (*pAux).der = NULL ; // 0(1)
19 | | then
20 | | | (*pAux).der ← & ⟨NULL,NULL,NULL⟩ ; // 0(1)
21 | | end
22 | | pAux ← (*pAux).der ; // 0(1)
23 | else
24 | | if (*pAux).izq = NULL ; // 0(1)
25 | | then
26 | | | (*pAux).izq ← & ⟨NULL,NULL,NULL⟩ ; // 0(1)
27 | | end
28 | | pAux ← (*pAux).izq ; // 0(1)
29 | end
30 | i ← i + 1 ; // 0(1)
31 end
32 if (*pAux).inter = NULL ; // 0(1)
33 then
34 | x ← inter(r) ; // 0(1)
35 | (*pAux).inter ← &x ; // 0(1)
36 else
37 | *((*pAux).inter) ← inter(r); // 0(1)
38 end
39 (*pAux).der ← NULL ; // 0(1)
40 (*pAux).izq ← NULL ; // 0(1)a

```

^aEn estas lineas podamos el arbol sin pensar en la etapa implementativa ya que sería un desperdicio de memoria. En ese momento se resolvera de alguna manera que emule la poda.

$i\text{InterfazDeSalida}(\text{in } p : \text{puntero}(\text{estr}), \text{in } d : \text{dirIp}) \rightarrow \text{res} : \text{interfaz}$

```

1  Complejidad:  $O(\text{version}(d))$ 
2  var i : int
3  var pRes: puntero(interfaz)
4  var pResProvisorio: puntero(interfaz)
5  var pAux puntero(estr)
6  var dBits: arreglo dinamico de bool
7  pRes  $\leftarrow$  NULL ; // 0(1)
8  dBits  $\leftarrow$  pasarABits(d) ; // 0(version(d))
9  pResProvisorio  $\leftarrow$  NULL ; // 0(1)
10 pAux  $\leftarrow$  p ; // 0(1)
11 i  $\leftarrow$  0 ; // 0(1)
12 while pRes = NULL ; // 0(version(d))a
13 do
14   if (pAux  $\rightarrow$  inter)  $\neq$  NULL ; // 0(1)
15   then
16     pResProvisorio  $\leftarrow$  (pAux  $\rightarrow$  inter) ; // 0(1)
17   end
18   if dBits[i] = true ; // 0(1)
19   then
20     if (pAux  $\rightarrow$  der) = NULL ; // 0(1)
21     then
22       pRes  $\leftarrow$  pResProvisorio ; // 0(1)
23     else
24       pAux  $\leftarrow$  (pAux  $\rightarrow$  der) ; // 0(1)
25     end
26   else
27     if (pAux  $\rightarrow$  izq) = NULL ; // 0(1)
28     then
29       pRes  $\leftarrow$  pResProvisorio ; // 0(1)
30     else
31       pAux  $\leftarrow$  (pAux  $\rightarrow$  der) ; // 0(1)
32     end
33   end
34   i  $\leftarrow$  i + 1 ; // 0(1)
35 end
36 res  $\leftarrow$  *(pRes) ; // 0(1)

```

Algorithm 2: interfazDeSalida

^aEsta complejidad se obtiene ya que se recorre el árbol hasta como mucho $8 \cdot \text{version}(d)$ y esto es $O(\text{version}(d))$

$itieneRegla(in\ p : \text{puntero}(\text{estr}), in\ d : \text{dirIp}) \longrightarrow res : \text{bool}$

```

1  Complejidad:  $O(\text{version}(d))$ 
2  var i : int
3  var pAux : puntero(estr)
4  var sePuedeSeguir : bool
5  var dBits: arreglo dinamico de bool

6  pAux  $\leftarrow$  p ;                                // 0(1)
7  dBits  $\leftarrow$  pasarABits(d) ;                  // 0(version(d))
8  i  $\leftarrow$  0 ;                                    // 0(1)
9  sePuedeSeguir  $\leftarrow$  true ;                    // 0(1)
10 res  $\leftarrow$  false ;                              // 0(1)
11 if pAux = NULL ;                                // 0(1)
12 then
13   | res  $\leftarrow$  false ;                          // 0(1)
14   | sePuedeSeguir  $\leftarrow$  false ;                // 0(1)
15 end

16 while res = false  $\wedge$  sePuedeSeguir = true ;    // 0(version(d))a
17 do
18   | if dBits[i] = true ;                          // 0(1)
19   | then
20     | if (pAux $\rightarrow$ der) = NULL ;                // 0(1)
21     | then
22       | sePuedeSeguir  $\leftarrow$  false ;            // 0(1)
23     | else
24       | pAux  $\leftarrow$  (pAux $\rightarrow$ der) ;          // 0(1)
25     | end
26   | else
27     | if (pAux $\rightarrow$ izq) = NULL ;                // 0(1)
28     | then
29       | sePuedeSeguir  $\leftarrow$  false ;            // 0(1)
30     | else
31       | pAux  $\leftarrow$  (pAux $\rightarrow$ izq) ;          // 0(1)
32     | end
33   | end
34   | i  $\leftarrow$  i + 1 ;                              // 0(1)
35   | if (pAux $\rightarrow$ inter)  $\neq$  NULL ;              // 0(1)
36   | then
37     | res  $\leftarrow$  true ;                          // 0(1)
38   | end
39 end

```

^aEsta complejidad se obtiene ya que se recorre el arbol hasta como mucho $8 \cdot \text{version}(d)$ y esto es $O(\text{version}(d))$

$ipasarABits(in\ d : dirIp) \longrightarrow res : arreglo_dinamico\ de\ bool$

```

1 Complejidad:  $O(version(d))$ 
2 var a : arreglo_dinamico de bool
3 var i, j, aux : nat
4 a  $\leftarrow$  crearArrego( $8*version(d)$ ) ; //  $O(version(d))$ 
5 i  $\leftarrow$  7 ; //  $O(1)$ 
6 j  $\leftarrow$  0 ; //  $O(1)$ 
7 while ( $j < version(d)$ ) ; //  $O(version(d))^a$ 
8 do
9   aux  $\leftarrow$  d[j] ; //  $O(1)$ 
10  while ( $i \geq 8*j$ ) ; //  $O(1)^b$ 
11  do
12    if aux mod 2 = 1 ; //  $O(1)$ 
13    then
14      a[i]  $\leftarrow$  true ; //  $O(1)$ 
15    else
16      a[i]  $\leftarrow$  false ; //  $O(1)$ 
17    end
18    aux  $\leftarrow$  aux / 2 ; //  $O(1)$ 
19    i  $\leftarrow$  i - 1 ; //  $O(1)$ 
20  end
21  i  $\leftarrow$  i + 15 ; //  $O(1)$ 
22  j  $\leftarrow$  j + 1 ; //  $O(1)$ 
23 end
24 res  $\leftarrow$  a ; //  $O(1)$ 

```

Algorithm 3: pasarABits

^aEsto se ejecuta $version(d)$ veces. Esto también podría haberse implementado con un “for” en vez de con un “while”. Por cada iteración de “j” se itera 8 veces con “i”, por lo que la complejidad total es $O(8*version(d))$, pero asintóticamente es $O(version(d))$

^bEsto se ejecuta 8 veces, porque cuando se entra aquí por primera vez vale que $i = j+7$. Esto también podría haberse implementado con un “for” en vez de con un “while”

TAD ($\text{SecuOrdSinRep}(\alpha), <_\alpha, =_\alpha$)

extiende $\text{Secu}(\alpha)$

géneros $\text{secuOrdSinRep}(\alpha)$

exporta genero, generadores, observadores básicos

generadores

agregarOrdenado : $\alpha \ a \times \text{secuOrdSinRep}(\alpha) \ s \longrightarrow \text{secuOrdSinRep}(\alpha) \quad (\neg \text{esta?}(a,s))$

otras operaciones

ordenada : $\text{secuOrdSinRep}(\alpha) \longrightarrow \text{bool}$

posicionRelativa : $\text{secuOrdSinRep}(\alpha) \times \alpha \longrightarrow \text{nat}$

com : $\text{secuOrdSinRep}(\alpha) \ s \longrightarrow \text{secuOrdSinRep}(\alpha) \quad \neg \text{vacía?}(s)$

axiomas $(\forall s: \text{secu}(\alpha))(\forall a: \alpha)$

$\text{vacía?}(\text{agregarOrdenado}(a, s)) \equiv \text{false}$

$\text{prim}(\text{agregarOrdenado}(a, s)) \equiv \text{if } (\text{vacía?}(s)) \text{ then } a \text{ else } \text{min}(a, \text{prim}(s))$

$\text{fin}(\text{agregarOrdenado}(a, s)) \equiv \text{if } (\text{vacía?}(s)) \vee_L a <_\alpha \text{prim}(s) \text{ then } s \text{ else } \text{agregarOrdenado}(a, \text{fin}(s))$

$\text{ordenada}(<>) \equiv \text{true}$

$\text{ordenada}(\text{agregarOrdenado}(a,s)) \equiv (a <_\alpha \text{prim}(s)) \wedge \text{ordenada}(s)$

$\text{com}(s) \equiv \text{if } (\text{vacía?}(\text{fin}(s))) \text{ then } \text{else } \text{agregarOrdenado}(\text{prim}(s), \text{com}(\text{fin}(s)))$

$\text{posicionRelativa}(s,a) \equiv \text{if } (\text{vacía?}(s) \vee_L a \leq_\alpha \text{prim}(s)) \text{ then } 0 \text{ else } 1 + \text{posicionRelativa}(\text{fin}(s),a)$

Fin TAD

TAD ItOrd(α)

Igualdad Observacional

$$(\forall \text{it}, \text{it}': \text{itOrd}(\alpha)) \left(\text{it} =_{\text{obs}} \text{it}' \iff (\text{VerSecuSuby}(\text{it}) =_{\text{secu}(\alpha)} \text{VerSecuSuby}(\text{it}')) \right)$$

généros	itOrd(α)
---------	-------------------

usa	Secu(α)
-----	------------------

exporta	género, generadores, observadores básicos
----------------	---

generadores

$$\text{CreaIt} \quad : \text{secu}(\alpha) \quad \longrightarrow \text{itOrd}(\alpha)$$

observadores básicos

$$\text{VerSecuSuby} : \text{itOrd}(\alpha) \longrightarrow \text{secu}(\alpha)$$

axioms $(\forall s: \text{secu}(\alpha))$

$$\text{VerSecuSuby}(\text{CreatIt}(s)) \equiv s$$

Fin TAD

TAD CONJUNTO(α)

extiende Conjunto(α)

géneros conj(α)

otras operaciones

posicionRelativa : conj(α) \times α \longrightarrow nat

axiomas ($\forall c: \text{conj}(\alpha)$)($\forall a: \alpha$)

posicionRelativa(c,a) \equiv **if** ($\emptyset?(c)$) **then** 0
 else if ($a \leq_{\alpha} \text{dameUno}(c)$) **then** posicionRelativa(sinUno(c),a)
 else 1 + posicionRelativa(sinUno(c),a)

Fin TAD