1. Módulo Conjunto(α)

Interfaz

se explica con: CONJUNTO(α)

géneros: `conjRapido(`α`)`

Operaciones básicas de conjunto

VACIO() $\rightarrow$ *res* : `conjRapido(`α`)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} \emptyset$}
Complejidad: $\Theta(1)$
Descripción: Crea un nuevo conjunto vacio

AGREGAR(**in** a: α **in/out** c: `conjRapido(`α`)`)
Pre $\equiv$ {c $=_{\text{obs}}$ c_0}
Post $\equiv$ {$c =_{\text{obs}}$ Ag(a, c_0)}
Complejidad: $\Theta(1)$
Descripción: Agrega el elemento a al conjunto c

VACIO?(**in** c: `conjRapido(`α`)`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} \emptyset?$(c)}
Complejidad: $\Theta(1)$
Descripción: Devuelve *true* si el conjunto esta vacio

EN(**in** a: α, **in** c: `conjRapido(`α`)`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ $a \in$ c}
Complejidad: O(N)
Descripción: Devuelve *true* si el elemento a pertenece al conjunto c

CANTIDAD(**in** c: `conjRapido(`α`)`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ #c}
Complejidad: O(N^2)
Descripción: Devuelve la cantidad de elementos definidos en c. En este conjunto se sacrifica la complejidad de *Cantidad* por *Agregar* siempre en O(1)

DAMEUNO(**in** c: `conjRapido(`α`)`) $\rightarrow$ *res* : α
Pre $\equiv$ {$\neg \emptyset?(c)$}
Post $\equiv$ {$res =_{\text{obs}}$ dameUno(c)}
Complejidad: $\Theta(1)$
Descripción: Devuelve un elemento del conjunto.

SINUNO(**in/out** c: `conjRapido(`α`)`)
Pre $\equiv$ {$\neg \emptyset?(c) \wedge c =_{\text{obs}} c_0$}
Post $\equiv$ {$c =_{\text{obs}}$ sinUno(c_0)}
Complejidad: O(N)
Descripción: Devuelve el conjunto sin el elemento que devuelve DameUno(c).

BORRAR(**in** a: α, **in/out** c: `conjRapido(`α`)`)
Pre $\equiv$ {$a \in c \wedge c =_{\text{obs}} c_0$}
Post $\equiv$ {$c =_{\text{obs}}$ c_0 - { a }}
Complejidad: O(N)

Descripción: Devuelve el conjunto sin el elemento a.

Representación

Representacion del Conjunto

`conjRapido(`α`)` **se representa con** `lista(`α`)`

Invariante de representacion en castellano:
Cualquier lista de elementos de tipo α es un conjunto valido de tipo α

Rep : lista(α) $\longrightarrow$ bool
Rep(l) $\equiv$ true $\iff$ true

Abs : lista(α) c $\longrightarrow$ conjRapido(α) {Rep(c)}
Abs(c) $\equiv$ con : conj(α) /
$(\forall\ a : \alpha)\ a \in con \iff \text{En}(a, c)$

Algoritmos

Algoritmos de Agentes

Lista de algoritmos

iVacio() → res: lista(α)
begin
| res ← Vacia() //**O(1)**
end
Complejidad: O(1)

Algoritmo 1: Vacio

---

iAgregar(**in** a: α, **in/out** c: `lista(`α`)`)
begin
| AgregarAtras(c, a) //**O(1)**
end
Complejidad: O(1)

Algoritmo 2: Agregar

---

iVacio?(**in** c: `lista(`α`)`) → res: bool
begin
| Vacia?(c) //**O(1)**
end
Complejidad: O(1)

Algoritmo 3: Vacio?

---

*i*En(**in** a: α, **in** c: `lista(`α`)`) → res: bool
begin
  var
  iterador : itLista
  iterador ← CrearIt(l) //**O(1)**
  res ← false //**O(1)**
  while *HaySiguiente(iterador)* **do** //**O(1)**
    if *Siguiente(iterador)* == a **then** //**O(1)**
      res ← true //**O(1)**
    end
    Avanzar(iterador) //**O(1)**
  end
  //**While: O(N)**
end
Complejidad: O(N)

Algoritmo 4: En

---

*i*DameUno(**in** c: `lista(`α`)`) → res: α
begin
  Primero(c) //**O(1)**
end
Complejidad: O(1)

Algoritmo 5: DameUno

---

*i*SinUno(**in/out** c: `lista(`α`)`)
begin
  Borrar(DameUno(c), c) //**O(N)**
end
Complejidad: O(N)

Algoritmo 6: SinUno

---

*i*Cantidad(**in** c: `lista(`α`)`) → res: nat
begin
  var
  iterador : itLista(α)
  copiaLista : lista(α)
  cantidad : nat
  cantidad ← 0 //**O(1)**
  iterador ← CrearIt(c) //**O(1)**
  res ← false //**O(1)**
  copiaLista ← Copiar(c) //**O(N)**
  while *HaySiguiente(iterador)* **do** //**O(1)**
    Fin(copiaLista) //**O(1)**
    if ¬*En(Siguiente(iterador), copiaLista)* **then** //**O(N)**
      cantidad ← cantidad + 1 //**O(1)**
    end
    Avanzar(iterador) //**O(1)**
  end
  //**While: O($\sum_{i=1}^{n} n$)**
end
Complejidad: O(N^2)
Comentarios: O($\sum_{i=1}^{n} n$) = O($\frac{1}{2}n(n+1)$) = O($\frac{1}{2}n^2 + \frac{1}{2}n$) = O($n^2$)

Algoritmo 7: Cantidad

```
iBorrar(in a: α, in/out c: lista(α))
begin
    var
    iterador : itLista(α)
    iterador ← CrearIt(c)                                          //O(1)
    while HaySiguiente(iterador) do                                //O(1)
        if Siguiente(iterador) == a then                           //O(1)
            EliminarSiguiente(iterador)                            //O(1)
        else
            Avanzar(iterador)                                      //O(1)
        end
    end
                                                                   //While: O(N)
end
```

Complejidad: O(N)

Algoritmo 8: Borrar