Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Juego de palabras

Grupo d-mifflin

Integrante	LU	Correo electrónico
Guglielmino, Adrian Roberto	39/18	adrianguglielmino@gmail.com
Hajek, David	52/21	hajekuba96@gmail.com
Lopez, Fernando Mariano	81/20	fmlopez@dc.uba.ar
Lopez Bianco, Macarena	268/18	maca_lopezbianco@hotmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Renombres De Tipos

Ficha es renombre de tupla<fila : nat, columna nat, letr: letra>
Ocurrencia es renombre de conj(Ficha)
Palabra es renombre de lista(letra)

2. Módulo Variante

Interfaz

usa: `conjTrie`, `dicc(letra,nat)`, `itDicc(letra,nat)`, `letra`, `palabra`

se explica con: VARIANTE

géneros: `variante`

Operaciones básicas de Variante

CREARVARIANTE(**in** *tamaño* : `nat`, **in** *cant_fichas* : `nat`, **in** *letraPuntaje* : `dicc(letra,nat)`, **in** *palabras_validas* : `conjTrie`) → *res* : `variante`
Pre ≡ $\{n > 0 \land cant\_fichas > 0 \land$
$(\forall p : palabra)(p \in palabras\_validas \rightarrow_L (\forall l : letra)(esta?(l, p) \rightarrow_L l \in letraPuntaje))\}$
Post ≡ $\{res =_{\text{obs}} nuevaVariante(tamaño, cant\_fichas, letraPuntaje, palabras\_validas)\}$
Complejidad: $\mathcal{O}(\#\text{Claves(letraPuntaje)})$
Descripción: crea un nueva variante
Aliasing: genera aliasing al conjTrie, el cual no debe ser modificado para mantener así la variante válida

TAMAÑODELTABLERO(**in** v : `variante`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} tamañoTablero(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tamaño del tablero de una variante del juego
Aliasing: no aplica

CANTFICHAS(**in** v : `variante`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} \#Fichas(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de fichas usada por cada jugador en la variante del juego
Aliasing: no aplica

PUNTAJEDELALETRA(**in** v : `variante`, **in** l : `letra`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} puntajeLetra(v, l)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el puntaje asociado a una determinada letra l en la variante del juego
Aliasing: no aplica

ESPALABRALEGÍTIMA?(**in** v : `variante`, **in** p : `Palabra`) → *res* : `bool`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}}$ palabraLegitima?(v, p)$\}$
Complejidad: $\mathcal{O}(longitud(p))$
Descripción: determina si una palabra es válida para la variante del juego
Aliasing: no aplica

SONPALABRASLEGÍTIMAS?(**in** v : `variante`, **in** *palabras* : `Lista(Palabra)`) → *res* : `bool`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} (\forall p : Palabra)(esta?(p, palabras) \rightarrow_L$ palabraLegítima?(v,p))$\}$
Complejidad: $\mathcal{O}(\sum_{p/esta?(p,palabras)} Longitud(p)) = \mathcal{O}(Lmax * Longitud(palabras))$
Descripción: determina si las palabras en la cola ingresada son válidas para la variante del juego. La complejidad está acotada por Lmax siendo el largo máximo de un elemento del conjunto de palabras validas de la variante
Aliasing: no aplica

Representación

`variante` **se representa con** `var`

donde `var` es `tupla`(*tamaño*: `nat`, *cant_fichas*: `nat`, *valor_letras*: `arreglo[nat]`, *palabras_permitidas*: `conjTrie(palabra)`)

Rep : var $\longrightarrow$ bool

Rep(v) ≡ true $\iff$ v.tamaño $> 0 \land$ v.cant_fichas $> 0 \land$ tamaño(v.valor_letras) $> 0 \land \neg$Vacío?(palabras_permitidas)

Abs : var v $\longrightarrow$ variante {Rep(v)}

Abs(v) $=_{\text{obs}}$ va: variante | (tamañoTablero(va) = v.tamaño) $\land$
(#Fichas(va) = v.cant_fichas) $\land$
$((\forall l : \text{letra})(\text{puntajeLetra}(va, l) = v.\text{valor\_letras}[ord^{-1}(l)])) \land$

$$((\forall p : \text{palabra})(\text{palabraLegitima?}(va, p) = p \in v.\text{palabras\_validas}))$$

Algoritmos

---

iCrearVariante(**in** n : `nat`, **in** f : `nat`, **in** *letraPuntaje* : `dicc(letra, nat)`, **in** *palabras_validas* : `conjTrie`) $\longrightarrow$ *res* : `var`

1: $res.tamaño \leftarrow n$ ▷ $\mathcal{O}(1)$
2: $res.cant\_fichas \leftarrow f$ ▷ $\mathcal{O}(1)$
3: $res.valor\_letras \leftarrow CrearArreglo(Nat)[\#Claves(letraPuntaje)]$ ▷ $\mathcal{O}(\#\text{Claves(letraPuntaje)})$

4: $it \leftarrow CrearIT(letraPuntaje)$ ▷ $\mathcal{O}(1)$
5: **mientras** HaySiguiente(it) **hacer** ▷ $\mathcal{O}(\#\text{Claves(letraPuntaje)})$
6: $letra0 \leftarrow Clave(Siguiente(it))$ ▷ $\mathcal{O}(1)$
7: $puntaje0 \leftarrow Significado(Siguiente(it))$ ▷ $\mathcal{O}(1)$
8: $res.valor\_letras[ord^{-1}(letra0)] \leftarrow puntaje0$ ▷ $\mathcal{O}(1)$
9: **end mientras**
10: $res.palabras\_permitidas \leftarrow palabras\_validas$ ▷ $\mathcal{O}(1)$
Complejidad: $\mathcal{O}(\#\text{Claves(letraPuntaje)})$

---

iTamañoDelTablero(**in** v : `var`) $\longrightarrow$ *res* : `nat`

1: $res \leftarrow v.tamaño$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iCantFichas(**in** v : `var`) $\longrightarrow$ *res* : `nat`

1: $res \leftarrow v.cant\_fichas$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iPuntajeDeLaLetra(**in** v : `var`, **in** l : `letra`) $\longrightarrow$ *res* : `nat`

1: $res \leftarrow v.valor\_letras[\text{ord}^{-1}(l)]$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iEsPalabraLegítima?(**in** v : `var`, **in** p : `Palabra`) $\longrightarrow$ *res* : `bool`

1: $res \leftarrow Pertenece?(v.palabras\_permitidas, p)$ ▷ $\mathcal{O}(L_{max})$

Complejidad: $\mathcal{O}(L_{max})$
siendo L_{max} el tamaño de la palabra más larga que pertenece al conjTrie palabras_permitidas

---

iSonPalabrasLegítimas?(**in** v : `var`, **in** p : `Lista(Palabra)`) $\longrightarrow$ *res* : `bool`

1: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
2: **mientras** res $\wedge_L$ $\neg$ EsVacia?(palabras) **hacer** ▷ $\mathcal{O}(\text{cantidad de palabras})$
3: $palabra \leftarrow Proximo(palabras)$ ▷ $\mathcal{O}(1)$
4: Desencolar(palabras) ▷ $\mathcal{O}(1)$
5: $res \leftarrow palabraPermitida(variante, palabra)$ ▷ $\mathcal{O}(L_{max})$
siendo L_{max} el tamaño de la palabra más larga que pertenece al conjTrie palabras_permitidas
6: **end mientras**

Complejidad: $\mathcal{O}(Longitud(p) * L_{max})$

---

3. Módulo Tablero

Extendemos TAD Tablero

TAD TABLERO EXTENDIDO

géneros tab

generadores

ponerLetra : tabt × nati × natj × letra × nat ⟶ tab {enTablero?(t, i, j) $\wedge_L hayLetra?(t, i, j)$}

observadores básicos

turno : tabt × nati × natj ⟶ nat {enTablero?(t, i, j) $\wedge_L hayLetra?(t, i, j)$}

otras operaciones

ponerLetras : tabt × ocurrenciao × nat ⟶ tab
{celdasLibres?(t, o) $\wedge(\forall i, j : nat)(\forall l, l0 : letra)((hi, j, li \in o \wedge hi, j, l0, i \in o) \rightarrow l = l0)$}

axiomas $\forall$ *t*: tab, $\forall$ $n, i, j, i0, j0$: nat, $\forall$ *l*: letra, $\forall$ *o*: ocurrencia, $\forall$ *tur*: nat

turno(ponerLetra(t, i, j, l, tur), i0, j0) ≡
if $i = i0 \wedge j = j0$ **then** *tur* **else** $turno(t, i0, j0)$ **fi**

ponerLetras(t, o, tur) ≡ **if** vacio?(o) **then**
t
else
$ponerLetra(ponerLetras(t, sinUno(o), tur), \Pi1(dameUno(o)), \Pi2(dameUno(o)),$
$\Pi3(dameUno(o)), tur)$
fi

Fin TAD

Interfaz

se explica con: TABLERO

géneros: `tablero`

Operaciones básicas de Tablero

CREARTABLERO(**in** n : `nat`) → *res* : `tablero`
Pre ≡ $\{n > 0\}$
Post ≡ $\{res =_{\text{obs}} nuevoTablero(n)\}$
Complejidad: $\mathcal{O}(n^2)$, donde n es el tamaño del tablero
Descripción: crea un tablero de tamaño n
Aliasing: no aplica

PONERLETRA(**in/out** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`, **in** l : `letra`, **in** *tur* : `nat`)
Pre ≡ $\{t = t_0 \wedge tur > 0 \wedge (enTablero?(t, fila, columna) \wedge_L \neg hayLetra?(t, fila, columna))\}$
Post ≡ $\{res =_{\text{obs}} ponerLetra(t_0, fila, columna, l, tur)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: coloca la ficha con la letra l y el turno *tur* en el tablero
Aliasing: no aplica

TAMAÑO(**in** t : `tablero`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} tamaño(t)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tamaño de un tablero
Aliasing: no aplica

HAYUNALETRAENPOS?(**in** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`) → *res* : `bool`
Pre ≡ $\{enTablero?(t, fila, columna)\}$
Post ≡ $\{res =_{\text{obs}} hayLetra?(t, fila, columna)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: determina si un casillero indicado de un tablero está ocupado
Aliasing: no aplica

LETRAENPOS(**in** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`) → *res* : `letra`
Pre ≡ $\{enTablero?(t, fila, columna) \wedge_L hayLetra?(t, fila, columna)\}$
Post ≡ $\{res =_{\text{obs}} letra(t, fila, columna)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la letra de un casillero del tablero
Aliasing: Se devuelve una referencia no modificable a la información del casillero

ENTABLERO(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) → *res* : `bool`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} enTablero?(t, i, j)\}$
Complejidad: $\mathcal{O}(1)$

Descripción: indica si el casillero (i,j) está en el tablero
Aliasing: no aplica

EstaLibre(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} libre?(t,i,j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el casillero (i,j) de un tablero está libre
Aliasing: no aplica

EstaOcupada(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} ocupada?(t,i,j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el casillero (i,j) de un tablero está ocupada
Aliasing: no aplica

ponerLetras(**in/out** t : `tablero`, **in** o : `Ocurrencia`, **in** tur : `nat`)
Pre $\equiv$ $\{t = t_o \wedge celdasLibres?(t,o) \wedge ((\forall i,j : nat)(\forall l,l' : letra)((\langle i,j,l\rangle \in o \wedge \langle i,j,l'\rangle \in o) \Rightarrow l = l'))\}$
Post $\equiv$ $\{res =_{\text{obs}} ponerLetras(t_0,o,tur)\}$
Complejidad: $\mathcal{O}(tamaño(o))$
Descripción: ubica cada elemento de la ocurrencia en el tablero
Aliasing: no aplica

sonCeldasLibres?(**in** t : `tablero`, **in** o : `Ocurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} celdasLibres?(t,o)\}$
Complejidad: $\mathcal{O}(tamaño(o))$
Descripción: chequea que las celdas de la ocurrencia esten libres
Aliasing: no aplica

turnoApoyado(**in** t : `tablero`, **in** $fila$: `nat`, **in** $columna$: `nat`) $\rightarrow$ res : `nat`
Pre $\equiv$ $\{enTablero?(t,fila,columna) \wedge_L hayLetra?(t,fila,columna)\}$
Post $\equiv$ $\{res =_{\text{obs}} turno(t,fila,columna)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el turno en el que se apoyó la ficha que está en la celda (fila, columna)
Aliasing: no aplica

Representación

`tablero` **se representa con** `tab`

donde `tab` es `tupla`($casillero$: `arreglo(arreglo` $\langle$ $ocupado$: `bool`, $letr$: `letra`, $turno$: `nat`$\rangle$)), tam: `nat`)

Rep : tab $\longrightarrow$ bool

Rep(t) $\equiv$ true $\iff$ $t.tam > 0 \wedge$
($\forall$ i,j: nat) ($0 \leq i,j <$t.tam $\Rightarrow_L$ (longitud(t.casilleros) = longitud(t.casilleros[i]) = t.tam) $\wedge$
$(\neg(t.casilleros[i][j]).ocupado) \Rightarrow_L$ ord^{-1}(1) =(t.casilleros[i][j]).letr $\wedge(t.casilleros[i][j]).turno = 0)$

Abs : tab t $\longrightarrow$ tablero $\{\text{Rep}(t)\}$

Abs(t) $=_{\text{obs}}$ ta: tablero | (tamaño(ta) = t.tam) $\wedge$
($(\forall i,j$: nat) $((0 \leq i,j < t.\text{tam}) \Rightarrow_L$ ((hayLetra?$(ta,i,j) = (t.$casilleros$[i][j])$.ocupado) $\wedge$
$((t.$casilleros$[i][j])$.ocupado $\Rightarrow_L$ letra$(ta,i,j) = (t.$casilleros$[i][j])$.letr))))

Algoritmos

iCrearTablero(**in** N : `nat`) $\longrightarrow$ res : `tab`

1: $res.casilleros$ ←arreglo(arreglo(tupla $\langle false, ord^{-1}(1), 0\rangle)[N])[N]$ ▷ $\mathcal{O}(N^2)$
2: $res.tam$ ←N ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(N^2)$

iPonerLetra(**in/out** t : `tab`, **in** $fila$: `nat`, **in** $columna$: `nat`, **in** l : `letra`, **in** tur : `nat`)

1: $t.casilleros[fila][columna].letr$ ←l ▷ $\mathcal{O}(1)$
2: $t.casilleros[fila][columna].ocupada$ ←true ▷ $\mathcal{O}(1)$
3: $t.casilleros[fila][columna].turno$ ←tur ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iTamaño(**in** t : tab) $\longrightarrow$ res : nat

1: $res \leftarrow t.tam$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iHayUnaLetraEnPos?(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : bool

1: $res \leftarrow t.casilleros[fila][columna].ocupado$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iLetraEnPos(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : letra

1: $res \leftarrow t.casilleros[fila][columna].letr$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iEnTablero(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : bool

1: $res \leftarrow (fila < t.tamaño) \wedge (columna < t.tamaño)$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iestaLibre(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : bool

1: $res \leftarrow ienTablero(t, fila, columna) \wedge \neg t.casilleros[fila][columna]).ocupada$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iestaOcupada(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : bool

1: $res \leftarrow ienTablero(t, fila, columna) \wedge t.casilleros[fila][columna]).ocupada$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iponerLetras(**in/out** t : tab, **in** o : Ocurrencia), **in** tur : nat)

1: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
2: **mientras** haySiguiente(it) **hacer** ▷ $\mathcal{O}$(tamaño(o))
3: $ficha_0 \leftarrow siguiente(o)$ ▷ $\mathcal{O}(1)$
4: $ponerLetra(t, ficha_0.fila, ficha_0.columna, ficha.letr, tur)$ ▷ $\mathcal{O}(1)$
5: Avanzar(it) ▷ $\mathcal{O}(1)$
6: **end mientras**

Complejidad: $\mathcal{O}(tamaño(o))$

---

iSonCeldasLibres?(**in** t : tab, **in** o : Ocurrencia) $\longrightarrow$ res : bool

1: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
2: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
3: **mientras** haySiguiente(it) **hacer** ▷ $\mathcal{O}$(tamaño(o))
4: $ficha_0 \leftarrow siguiente(o)$ ▷ $\mathcal{O}(1)$
5: **si** estaOcupada(tab, $ficha_0.fila, ficha_0.columna$) **entonces**
6: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
7: **end si**
8: Avanzar(it) ▷ $\mathcal{O}(1)$
9: **end mientras**

Complejidad: $\mathcal{O}(tamaño(o))$

---

iturnoApoyado(**in** t : tab, **in** *fila* : nat, **in** *columna* : nat) $\longrightarrow$ res : nat

1: $res \leftarrow t.casilleros[fila][columna].turno$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

4. Módulo Juego

Interfaz

se explica con: JUEGO

géneros: `juego`.

Operaciones básicas de Juego

CREARNUEVOJUEGO(**in** *jugadores* : `nat`, **in** *varianteDelJuego* : `variante`, **in** *bolsaDeFichas*: `cola(letra)`) → *res* : `juego`
Pre ≡ {long(bolsaDeFichas) ≤ $tamañoTablero(v) * tamañoTablero(v) + k * fichas(v) \land k > 0$}
Post≡ {$res = nuevoJuego(jugadores, varianteDelJuego, bolsaDeFichas)$}
Complejidad: $\mathcal{O}(N^2 + F * K + \Sigma * K)$
Descripción: crea un nuevo juego
Aliasing: No aplica

UBICARFICHAS(**in/out** j : `juego`, **in** *ocurrencia* : `o`) → *res* : `juego`
Pre ≡ {$j = j0 \land JugadaValida?(j, o)$}
Post ≡ {$res = ubicar(j0, o)$}
Complejidad: $\mathcal{O}(cardinal(o))$
Descripción: ubica las fichas en el tablero, pasa el turno, actualiza los puntos del jugador y repone la cantidad de fichas ubicadas
Aliasing: no aplica

VARIANTEDELJUEGO(**in** j : `juego`) → *res* : `variante`
Pre ≡ {$true$}
Post ≡ {$res = variante(j)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la variante del juego
Aliasing: Se devuelve una referencia no modificable a la variante del juego

CANTJUGADORES(**in** j : `juego`) → *res* : `nat`
Pre ≡ {$true$}
Post ≡ {$res = \#Jugadores(j)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de jugadores
Aliasing: no aplica

REPOSITORIODELJUEGO(**in** j : `juego`) → *res* : `cola(letras)`
Pre ≡ {$true$}
Post ≡ {$res = repositorio(j)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el repositorio del juego
Aliasing: Se devuelve una referencia no modificable al repositorio del juego

TABLERO(**in** j : `juego`) → *res* : `tablero`
Pre ≡ {$true$}
Post ≡ {$res = tablero(j)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tablero del juego
Aliasing: Se devuelve una referencia no modificable al tablero del juego

POSDETABLERO(**in** ju : `juego`, **in** i : `nat`, **in** j : `nat`) → *res* : `tupla(bool, letra, nat)`
Pre ≡ {$true$}
Post ≡ {$res =< ocupada?(tablero(ju), i, j), letra(tablero(ju), i, j), turno(tablero(ju), i, j) >$}
Complejidad: $\mathcal{O}(1)$
Descripción: dada una posición del tablero, devuelve si está ocupada, la letra que lo ocupa y el turno en el que fue ubicada la ficha (si está vacía, devuelve la primera letra del alfabeto y el turno 0)
Aliasing: Se devuelve una referencia no modificable al casillero del tablero del juego

TURNOACTUAL(**in** j : `juego`) → *res* : `nat`
Pre ≡ {$true$}
Post ≡ {$res = turno(j)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve a quien le toca jugar
Aliasing: no aplica

FICHASDEJUGADOR(**in** j : `juego`, **in** i : `nat`) → *res* : `arreglo(nat)`
Pre ≡ {$i < \#Jugadores(j)$}
Post ≡ {$(\forall f : ficha)(f \in fichas(i, j) \Rightarrow_L f \in \pi1(res) \land (f, fichas(i, j) = \pi2(res))$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de fichas de cada letra que tiene el jugador
Aliasing: No aplica

PUNTAJE(**in** j : `juego`, **in** i : `nat`, **out** *puntajePrevio* : `nat`) → *res* : `nat`

Pre $\equiv \{i < \#Jugadores(j)\}$
Post $\equiv \{res = puntaje(j)\}$
Complejidad: $\mathcal{O}(1$ + cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación $*$ longitud de la palabra mas larga$)$
Descripción: devuelve el puntaje del jugador
Aliasing: no aplica

CantidadFichasDeLetra(**in** j : `juego`, **in** l : `letra`, **in** i : `nat`) $\rightarrow$ res : nat
Pre $\equiv \{i < \#Jugadores(j)\}$
Post $\equiv \{res = \#(l, fichas(j, i))\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuleve la cantidad de fichas con una letra que tiene un jugador
Aliasing: no aplica

jugadaValida(**in** j : `juego`, **in** o : `ocurrencia`) $\rightarrow$ res : $bool$
Pre $\equiv \{i < \#Jugadores(j)\}$
Post $\equiv \{res = jugadaValida?(j, o)\}$
Complejidad: $\mathcal{O}(L_{max})$
Descripción: dice si la jugada es válida
Aliasing: No aplica

Representación

`juego` **se representa con** `jueg`

donde `jueg` es `tupla`(*variante*: `var`, *repositorio*: `cola(letra)`, *tablero*: `tab`, *jugadores*: `arreglo(jugador)`, *turnoDe*: `nat`, *cantidadDeTurnos*: `nat`)

donde `jugador` es `tupla`(*fichas*: `arreglo(nat)`, *puntos*: `tupla`⟨*puntaje* : `nat`, *ocurrencias* : `cola(palabra)` ⟩)

Rep : jueg $\longrightarrow$ bool

Rep(j) $\equiv$ true $\Longleftrightarrow$ long(j.repositorio) >0 $\wedge$
tamaño(j.tablero) = tamañoTablero(j.variante) $\wedge$
0 $\leq$ j.turnoDe <Longitud(j.jugadores) $\wedge$
j.cantidadDeTurnos $\geq$ 0 $\wedge$
($\forall$ i : nat)(0$\leq$ i <Longitud(j.jugadores) $\Rightarrow_L$ cantFichas(j.variante) = cantFichasTiene(j,i))

Abs : jueg j $\longrightarrow$ juego {Rep(j)}

Abs(j) $=_{\text{obs}}$ ju: juego | variante(ju) = j.variante $\wedge$
#jugadores(ju) = Longitud(j.jugadores) $\wedge$
repositorio(ju) = j.repositorio $\wedge$
tablero(ju) = j.tablero $\wedge$
turno(ju) = j.turnoDe $\wedge$
($\forall i$: nat)(0 $\leq$ i <Longitud(j.jugadores)(j.jugadores) $\Rightarrow_{\text{L}}$ puntaje(ju, i) =
j.jugador[i].puntos.puntaje + calcularPuntosPalabrasJugadas(j, j.jugador[i].puntos.ocurrencias)
$\wedge_L$ ($\forall$ L : letra)(L $\in$ fichas(ju, i) $\Rightarrow_{\text{L}}$ #(L, fichas(ju,i)) = j.jugadores[i].fichas[ord^{-1}(L)]))

Algoritmos

---

iCrearNuevoJuego(**in** *cantJugadores* : `nat`, **in** *varianteDelJuego* : `variante`, **in** *bolsaDeFichas* : `cola(letra)`) $\longrightarrow$ *res* : `jueg`

1: $jueg.var \leftarrow varianteDelJuego$ ▷ $\mathcal{O}(1)$
2: $jueg.repositorio \leftarrow bolsaDeFichas$ ▷ $\mathcal{O}(1)$
3: $tam \leftarrow tamañoDelTablero(var)$ ▷ $\mathcal{O}(1)$
4: $jueg.tablero \leftarrow crearTablero(tam)$ ▷ $\mathcal{O}((\text{tamaño del tablero})^2)$

5: $\#fichas \leftarrow CantFichas(var)$ ▷ $\mathcal{O}(1)$
6: $jueg.jugadores \leftarrow CrearArreglo[cantJugadores]$ ▷ $\mathcal{O}(\text{cantidad de jugadores})$
7: $jueg.cantidadDeTurnos \leftarrow 0$ ▷ $\mathcal{O}(1)$
8: $i \leftarrow 0$ ▷ $\mathcal{O}(1)$

9: **mientras** i <cantJugadores **hacer** ▷ $\mathcal{O}(\text{cantidad de fichas por jugador} * \text{cantidad de jugadores} + \text{cantidad de letras del alfabeto} * \text{cantidad de jugadores})$
10: $jueg.jugadores[i] \leftarrow jugador$ ▷ $\mathcal{O}(1)$
11: $repositorioDelJugador \leftarrow CrearArreglo(fichas)[\#fichas]$ ▷ $\mathcal{O}(\text{cantidad de fichas por jugador})$
12: $fichasPorJugador \leftarrow CrearArreglo[nat](\#letrasDelAbecedario)$ ▷ $\mathcal{O}(\text{cantidad de letras del abecedario})$
13: $jugador.puntos.puntaje \leftarrow 0$ ▷ $\mathcal{O}(1)$
14: $jugador.puntos.ocurrencia \leftarrow cola < palabras >$ ▷ $\mathcal{O}(1)$
15: **para** j = 0... #fichas-1 **hacer** ▷ $\mathcal{O}(|\Sigma|)$
16: $fichaDelJugador \leftarrow proximo(jueg.repositorio)$ ▷ $\mathcal{O}(1)$
17: $repositorioDelJugador[j] \leftarrow fichaDelJugador$ ▷ $\mathcal{O}(1)$
18: $jueg.repositorio \leftarrow desencolar(jueg.repositorio)$ ▷ $\mathcal{O}(1)$
19: $letraDeLaFicha \leftarrow fichaDelJugador.letra$ ▷ $\mathcal{O}(1)$
20: $fichasPorJugador[ord^{-}1(letraDeLaFicha)] \leftarrow fichasPorJugador[ord^{-}1(letraDeLaFicha)] + 1$ ▷ $\mathcal{O}(1)$
21: **end para**
22: $jugador.fichas \leftarrow fichasPorJugador$ ▷ $\mathcal{O}(1)$
23: $i \leftarrow i + 1$ ▷ $\mathcal{O}(1)$
24: **end mientras**
25: $jueg.turnoDe \leftarrow 0$ ▷ $\mathcal{O}(1)$

26: $res \leftarrow jueg$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}((\text{tamaño del tablero})^2 + \text{cantidad de fichas por jugador} * \text{cantidad de jugadores} + \text{cantidad de letras del alfabeto} * \text{cantidad de jugadores})$

---

iubicarfichas(**in/out** *j* : `jueg`, **in** *o* : `ocurrencia`)

1: $jugador_0 \leftarrow j.jugadores[j.turnoDe]$ ▷ $\mathcal{O}(1)$
2: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
3: $ponerLetras(t, o, j.cantidadDeTurnos)$ ▷ $\mathcal{O}(\text{cardinal(o)}) = \mathcal{O}(\text{cantidad de fichas que se ubican})$
4: $j.cantidadDeTurnos++$ ▷ $\mathcal{O}(1)$

5: **mientras** haySiguiente(it) **hacer** ▷ $\mathcal{O}(\text{cardinal(o)})$
6: $ficha \leftarrow Siguiente(it)$ ▷ $\mathcal{O}(1)$
7: $jugador_0.fichas[ord(ficha.letra)]--$ ▷ $\mathcal{O}(1)$
8: $nroNuevaLetra \leftarrow ord(proxima(j.repositorio))$ ▷ $\mathcal{O}(1)$
9: $jugador_0.fichas[nroNuevaLetra]++$ ▷ $\mathcal{O}(1)$
10: $desencolar(j.repositorio)$ ▷ $\mathcal{O}(1)$
11: $Avanzar(it)$ ▷ $\mathcal{O}(1)$
12: **end mientras**

13: $encolar(jugador0.puntos.ocurrencias, o)$ ▷ $\mathcal{O}(1)$
14: $turno \leftarrow j.turnoDe$ ▷ $\mathcal{O}(1)$
15: $turno++$ ▷ $\mathcal{O}(1)$
16: **si** cantJugadores(j) == turno **entonces**
17: $turno \leftarrow 0$ ▷ $\mathcal{O}(1)$
18: **end si**

Complejidad: $\mathcal{O}(\text{cantidad de fichas que se ubican})$

---

iVarianteDelJuego(**in** *j* : `jueg`) $\longrightarrow$ *res* : `variante`

1: $res \leftarrow j.variante$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

---

iRepositorioDelJuego(**in** j : `jueg`) $\longrightarrow$ res : `cola(letras)`

1: $res \leftarrow j.repositorio$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iTablero(**in** j : `jueg`) $\longrightarrow$ res : `tab`

1: $res \leftarrow j.tablero$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iCantJugadores(**in** j : `jueg`) $\longrightarrow$ res : `nat`

1: $res \leftarrow Longitud(j.jugadores)$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iPosDeTablero(**in** j : `juego`, **in** i : `nat` , **in** j : `nat`) $\longrightarrow$ res : `tupla(bool, letra, nat)`

1: $res \leftarrow j.tablero[i][j]$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iTurnoActual(**in** j : `jueg`) $\longrightarrow$ res : `nat`

1: $res \leftarrow j.turnoDe$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iFichasDeJugador(**in** j : `jueg`, **in** i : `nat`) $\longrightarrow$ res : `arraglo(nat)`

1: $res \leftarrow j.jugador[i].fichas$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iPuntaje(**in** j : `jueg`, **in** i : `nat`) $\longrightarrow$ res : `nat`

1: $ocus \leftarrow j.jugadores[i]).puntos.ocurrencias$ ▷ $\mathcal{O}(1)$
2: $puntosACalcular \leftarrow calcularPuntosPalabrasJugadas(j, ocus)$ ▷ $\mathcal{O}$(cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación $*$ longitud de la palabra mas larga)
3: $j.jugadores[i]).puntos.puntaje \leftarrow j.jugadores[i]).puntos.puntaje + puntosACalcular$ ▷ $\mathcal{O}(1)$
4: $res \leftarrow j.jugadores[i].puntos.puntaje$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1$ + cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación $*$ longitud de la palabra mas larga)

iCantidadFichasDeLetra(**in** j : `juego`, **in** l : `letra`, **in** i : `nat`) $\longrightarrow$ res : `nat`

1: $res \leftarrow j.jugador[i].fichas[ord^{-1}(l)]$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

IJUGADAVALIDA(**in** j : jueg, **in** o : ocurrencia) $\longrightarrow$ res : bool

1: **si** CARDINAL(O) $= 0$ **entonces** ▷ $\mathcal{O}(1)$
2: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
3: **else**
4: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
5: $varJ \leftarrow VarianteDelJuego(j)$ ▷ $\mathcal{O}(1)$
6: $largoMax \leftarrow LargoDeClaveMax(palabras\_validas(varJ))$ ▷ $\mathcal{O}(1)$

7: **si** RES $\wedge_L$ (CANTFICHAS(VARJ) $<$ CARDINAL(O) $\vee_L$ LARGOMAX $<$ CARDINAL(O)) **entonces** ▷ $\mathcal{O}(1)$
8: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
9: **end si**

10: $fichasJugador \leftarrow Fichas(juego.jugadores[turno\_de(j)])$ ▷ $\mathcal{O}(1)$
11: **si** RES $\wedge_L \neg tieneLasFichas(fichasJugador, o)$ **entonces** ▷ $\mathcal{O}(Cardinal(o)^2)$ que acotado es $\mathcal{O}(L_{max}^2)$
12: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
13: **end si**

14: **si** RES $\wedge_L \neg$CELDASLIBRES?(TABLERO(JUEGO), O) **entonces** ▷ $\mathcal{O}(L_{max})$
15: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
16: **end si**

17: $esHorizontal \leftarrow false$ ▷ $\mathcal{O}(1)$
18: $esVertical \leftarrow false$ ▷ $\mathcal{O}(1)$

19: **si** RES **entonces**
20: $esHorizontal, esVertical \leftarrow HorizontalOVertical(o)$ ▷ $\mathcal{O}(L_{max})$
21: **si** $\neg$(ESHORIZONTAL $\vee_L$ ESVERTICAL) **entonces** ▷ $\mathcal{O}(1)$
22: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
23: **end si**
24: **end si**

25: **si** RES $\wedge_L \neg$ OCURRENCIASINPOSICIONESREPETIDAS(O) **entonces** ▷ $\mathcal{O}(L_{max}^2)$
26: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
27: **end si**

28: $sentido \leftarrow true$ ▷ $\mathcal{O}(1)$
29: **si** RES $\wedge_L$ ESVERTICAL **entonces**
30: $sentido \leftarrow false$ ▷ $\mathcal{O}(1)$
31: **end si**

32: **si** RES $\wedge_L$ $2 \leq$ #CARDINAL(O) $\wedge_L$ $\neg$DISTANCIANOMAYORALONGMAX(O, SENTIDO, LARGOMAX) **entonces** ▷ En este contexto, la complejidad es $\mathcal{O}(L_{max})$, porque en la línea 7 acotamos el cardinal(o)
33: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
34: **end si**

35: **si** RES $\wedge_L$ CARDINAL(O) $= 1$ **entonces**
36: $iter \leftarrow CreatIT(o)$ ▷ $\mathcal{O}(1)$
37: $ficha0 \leftarrow Siguiente(iter)$ ▷ $\mathcal{O}(1)$
38: $palabraVertical \leftarrow PalabraTransversalVertical(tablero(j), ficha0)$ ▷ $\mathcal{O}(L_{max})$
39: **si** $\neg$ESPALABRALEGÍTIMA?(VARJ, PALABRAVERTICAL) **entonces** ▷ $\mathcal{O}(L_{max})$
40: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
41: **end si**
42: **end si**

43: $palabraHorizontal \leftarrow PalabraTransversalHorizontal(tablero(j), ficha0)$ ▷ $\mathcal{O}(L_{max})$
44: **si** RES $\wedge_L \neg$ESPALABRALEGÍTIMA?(VARJ, PALABRAHORIZONTAL) **entonces** ▷ $\mathcal{O}(L_{max})$
45: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
46: **end si**

47: **si** RES $\wedge_L \neg$CARDINAL(O) $!= 1$ **entonces** ▷ $\mathcal{O}(1)$
48: $palabraPrincipal \leftarrow formarPalabraPrincipal(tablero(j), o, sentido)$ ▷ $\mathcal{O}(L_{max}{}^2)$
49: **si** ESVACIA?(PALABRAPRINCIPAL) $\vee_L \neg$ ESPALABRALEGÍTIMA?(VARJ, PALABRAPRINCIPAL) **entonces** ▷ $\mathcal{O}(L_{max})$
50: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
51: **else**
52: $palabras \leftarrow PalabrasTransversales(tablero(j), o, sentido)$ ▷ $\mathcal{O}(L_{max}{}^2)$
53: **si** $\neg$SONPALABRASLEGÍTIMAS?(VARJ, PALABRAS) **entonces** ▷ $\mathcal{O}(Longitud(palabras) * L_{max}) = \mathcal{O}(L_{max}{}^2)$, porque en este contexto longitud(palabras)= L_{max}+1
54: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
55: **end si**
56: **end si**
57: **end si**
58: **end si**

Complejidad: $\mathcal{O}(L_{max}{}^2)$ donde L_{max} es la longitud de la palabra más larga del alfabeto

4.1. Funciones Auxiliares

4.1.1. Auxiliares Puntaje

calcularPuntosPalabrasJugadas(**in** j : `juego`, **in** $ocus$: `cola(ocurrencias)`) $\longrightarrow$ res : `nat`

1: $res \leftarrow 0$ ▷ $\mathcal{O}(1)$
2: **mientras** ¬ esVacia?(ocus) **hacer** ▷ $\mathcal{O}(\sum_{\forall o \in ocus} long(o) * L_{max})$
3: $p \leftarrow proximo(ocus)$ ▷ $\mathcal{O}(1)$
4: $palabras \leftarrow palabrasFormadas(j, p)$ ▷ $\mathcal{O}$(longitud(o) * L_{max})
5: $desencolar(ocus)$ ▷ $\mathcal{O}(1)$
6: $puntosDeLasPalabras \leftarrow calcularPuntos(j, palabras)$ ▷ $\mathcal{O}$((longitud(o)+1) * L_{max})
7: $res \leftarrow res + puntosDeLasPalabras$ ▷ $\mathcal{O}(1)$
8: **end mientras**

Complejidad: $\mathcal{O}(\sum_{\forall o \in ocus} long(o)$*L$_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto)
▷ Pre{$\forall$ o : ocurrencia en ocus vale jugadaValida(j, o)}
▷ Post{calculamos el puntaje de las palabras que se formaron con las fichas de cada ocurrencia en la cola ocu: la palabra principal y las palabras transversales}

iPalabrasFormadas(**in** j : `juego`, **in** o : `ocurrencia`) $\longrightarrow$ res : `cola(palabra)`

1: **si** Cardinal(o) ≥ 1 **entonces**
2: $tablero \leftarrow tablero(juego)$ ▷ $\mathcal{O}(1)$
3: $esHorizontal, esVertical \leftarrow HorizontalOVertical(o)$ ▷ $\mathcal{O}$(longitud(o))
4: $sentido \leftarrow true$ ▷ $\mathcal{O}(1)$
5: **si** esVertical **entonces**
6: $sentido \leftarrow false$ ▷ $\mathcal{O}(1)$
7: **end si**
8: $res \leftarrow PalabrasFormadasTransversales(tablero, o, sentido)$ ▷ $\mathcal{O}$(longitud(o) $*$ L_{max})
9: $palabraPrincipal \leftarrow formarPalabraJugadaPrincipal(tablero, o, sentido)$ ▷ $\mathcal{O}(L_{max})$
10: $encolar(res, palabraPrincipal)$ ▷ $\mathcal{O}(1)$
11: **si** cardinal(o) $= 1$ **entonces**
12: $palabraHorizontal \leftarrow formarPalabraJugadaPrincipal(tablero, o, sentido)$ ▷ $\mathcal{O}(L_{max})$
13: **si** Longitud(palabraPrincipal) $= 1 \wedge$ Longitud(palabraHorizontal) $= 1$ **entonces**
14: $desencolar(res)$ ▷ $\mathcal{O}(1)$
15: **end si**
16: **end si**
17: **else**
18: $res \leftarrow crearCola(palabras)$ ▷ $\mathcal{O}(1)$
19: **end si**

Complejidad: $\mathcal{O}$(longitud(o) $*$ L_{max}), donde L_{max} es la longitud de la palabra más larga del alfabeto)
▷ Pre{jugadaValida(j,o)}
▷ Post{calculamos el puntaje de las palabras que se formaron con las fichas de cada ocurrencia en la cola ocu: la palabra principal y las palabras transversales}

iHorizontalOVertical(**in** o : `ocurrencia`) $\longrightarrow$ `tupla` $\langle$ `horizontal : bool, vertical : bool` $\rangle$

1: $horizontal \leftarrow true$ ▷ $\mathcal{O}(1)$
2: $vertical \leftarrow true$ ▷ $\mathcal{O}(1)$
3: **si** Cardinal(o) != 0 **entonces**
4: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
5: $fila0 \leftarrow Siguiente(o).fila$ ▷ $\mathcal{O}(1)$
6: $col0 \leftarrow Siguiente(o).columna$ ▷ $\mathcal{O}(1)$
7: $it \leftarrow Avanzar(it)$ ▷ $\mathcal{O}(1)$
8: **mientras** HaySiguiente(it) $\wedge$ (horizontal $\vee$ vertical) **hacer** ▷ $\mathcal{O}$(cardinal(o)-1)
9: $fila1 \leftarrow Siguiente(o).fila$ ▷ $\mathcal{O}(1)$
10: $col1 \leftarrow Siguiente(o).columna$ ▷ $\mathcal{O}(1)$
11: **si** horizontal **entonces**
12: $horizontal \leftarrow fila1 == fila0$ ▷ $\mathcal{O}(1)$
13: **end si**
14: **si** vertical **entonces**
15: $vertical \leftarrow col0! = col1$ ▷ $\mathcal{O}(1)$
16: **end si**
17: $fila0 \leftarrow fila1$ ▷ $\mathcal{O}(1)$
18: $col0 \leftarrow col1$ ▷ $\mathcal{O}(1)$
19: $it \leftarrow avanzar(it)$ ▷ $\mathcal{O}(1)$
20: **end mientras**
21: **end si**

Complejidad: $\mathcal{O}$(cardinal(o))
▷ Pre{¬vacio(o)}
▷ Post{horizontal = true si y solo si todas las fichas en la ocurrencia tienen mismo valor en el campo columna vertical = true si y solo si todas las fichas en o tienen mismo valor en el campo fila}

iFormarPalabraJugadaPrincipal(**in** *tab* : `tablero`, **in** *o* : `ocurrencia` , **in** *sentido* : `bool`)) $\longrightarrow$ *res* : `palabra`

1: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
2: $f \leftarrow siguente(it)$ ▷ $\mathcal{O}(1)$
3: $res \leftarrow crearLista(letra)$ ▷ $\mathcal{O}(1)$
4: **si** sentido **entonces**
5: $principal \leftarrow iPalabraJugadaPrincipalHorizontal(tablero, f)$ ▷ $\mathcal{O}(L_{max})$
6: **else**
7: $principal \leftarrow iPalabraJugadaPrincipalVertical(tablero, f)$ ▷ $\mathcal{O}(L_{max})$
8: **end si**

Complejidad: $\mathcal{O}(L_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{longitud(o)>0 $\wedge$ la ocurrencia es una jugada válida}
▷ Post{res es la palabra principal que forma la ocurrencia cuando la apoyamos en el tablero}

iPalabraJugadaPrincipalHorizontal (**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

1: $res \leftarrow crearLista(palabra)$ ▷ $\mathcal{O}(1)$
2: $l \leftarrow f.letra$ ▷ $\mathcal{O}(1)$
3: $AgregarAdelante(res, l)$ ▷ $\mathcal{O}(1)$
4: $fila \leftarrow f.fila$ ▷ $\mathcal{O}(1)$
5: $columna \leftarrow f.columna$ ▷ $\mathcal{O}(1)$
6: $izquierda \leftarrow f.columna - 1$ ▷ $\mathcal{O}(1)$
7: $derecha \leftarrow f.columna + 1$ ▷ $\mathcal{O}(1)$
8: $turno \leftarrow turnoApoyado(tablero, fila, columna)$ ▷ $\mathcal{O}(1)$
9: **mientras** (derecha <tamaño(tablero) $\wedge$ Ocupada(tablero,fila, derecha) $\wedge$
10: turno >turnoApoyado(tablero, fila, derecha)) **hacer** ▷ $\mathcal{O}(L_{max})$
11: $l_0 \leftarrow LetraEnPos(tablero, fila, derecha)$ ▷ $\mathcal{O}(1)$
12: $AgregarAtras(res, l_0)$ ▷ $\mathcal{O}(1)$
13: $derecha++$ ▷ $\mathcal{O}(1)$
14: **end mientras**
15: **mientras** $0 \leq$ izquierda $\wedge$ Ocupada(tablero,fila, izquierda) $\wedge$
16: turno>turnoApoyado(tablero, fila, izquierda) **hacer** ▷ $\mathcal{O}(L_{max})$
17: $l_0 \leftarrow LetraEnPos(tablero, fila, izquierda)$ ▷ $\mathcal{O}(1)$
18: $AgregarAdelante(res, l_0)$ ▷ $\mathcal{O}(1)$
19: $izquierda--$ ▷ $\mathcal{O}(1)$
20: **end mientras**

Complejidad: $\mathcal{O}(L_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{el sentido de la palabra principal formada por la ocurrencia es horizontal $\wedge$ la ficha se encuentra en el tablero}
▷ Post{res es la ocurrencia que se forma en el tablero con la ficha horizontalmente}

iPalabraJugadaPrincipalVertical(**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

1: $res \leftarrow crearLista(palabra)$ ▷ $\mathcal{O}(1)$
2: $l \leftarrow f.letra$ ▷ $\mathcal{O}(1)$
3: $AgregarAdelante(res, l)$ ▷ $\mathcal{O}(1)$
4: $fila \leftarrow f.fila$ ▷ $\mathcal{O}(1)$
5: $columna \leftarrow f.columna$ ▷ $\mathcal{O}(1)$
6: $arriba \leftarrow f.fila - 1$ ▷ $\mathcal{O}(1)$
7: $abajo \leftarrow f.fila + 1$ ▷ $\mathcal{O}(1)$
8: $turno \leftarrow turnoApoyado(tablero, fila, columna)$ ▷ $\mathcal{O}(1)$
9: **mientras** $0 \leq$ arriba $\wedge$ Ocupada(tablero,arriba,columna) $\wedge$
10: turno>turnoApoyado(tablero, fila, izquierda)) **hacer** ▷ $\mathcal{O}(L_{max})$
11: $l_0 \leftarrow LetraEnPos(tablero, arriba, columna)$ ▷ $\mathcal{O}(1)$
12: $AgregarAdelante(res, l_0)$ ▷ $\mathcal{O}(1)$
13: $arriba--$ ▷ $\mathcal{O}(1)$
14: **end mientras**
15: **mientras** abajo <tamaño(tablero) $\wedge$ Ocupada(tablero,fila, abajo) $\wedge$
16: turno>turnoApoyado(tablero, fila, izquierda) **hacer** ▷ $\mathcal{O}(L_{max})$
17: $l_0 \leftarrow LetraEnPos(tablero, abajo, columna)$ ▷ $\mathcal{O}(1)$
18: $AgregarAtras(res, l_0)$ ▷ $\mathcal{O}(1)$
19: $abajo++$ ▷ $\mathcal{O}(1)$
20: **end mientras**

Complejidad: $\mathcal{O}(L_{max})$, , donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{el sentido de la palabra principal formada por la ocurrencia es vertical $\wedge$ la ficha se encuentra en el tablero}
▷ Post{res es la ocurrencia que se forma en el tablero con la ficha verticalmente}

iPalabrasFormadasTransversales(**in** *tab* : `tablero`, **in** *o* : `ocurrencia`, **in** *sentido* : `bool`) $\longrightarrow$ *res* : `cola(palabra)`

```
1:  it ← crearIT(o)                                                          ▷ O(1)
2:  palabras ← Cola(palabra)                                                 ▷ O(1)
3:  mientras haySiguiente(it) hacer                                          ▷ O(longitud(o) ∗L_max)
4:      ficha ← Siguiente(it)                                                ▷ O(1)
5:      palabra ← CrearLista(letra)                                          ▷ O(1)
6:      si sentido entonces
7:          pal ← palabraFormadaTransversalVertical(tablero, ficha)          ▷ O(L_max)
8:      else
9:          pal ← palabraFormadaTransversalHorizontal(tablero, ficha)        ▷ O(L_max)
10:     end si
11:     encolar(palabras, pal)                                               ▷ O(1)
12:     avanzar(it)                                                          ▷ O(1)
13: end mientras
```

Complejidad: $\mathcal{O}$(longitud(o)∗L_{max}), donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{longitud(o)>0 ∧ la ocurrencia es una jugada válida}
▷ Post{res es la cola de palabras transversales que se forman en el tablero con cada letra de la ocurrencia }

iPalabraFormadaTransversalHorizontal(**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                                  ▷ O(1)
2:  fila ← f.fila                                                            ▷ O(1)
3:  columna ← f.columna                                                      ▷ O(1)
4:  derecha ← f.colunna + 1                                                  ▷ O(1)
5:  izquierda ← f.colunna − 1                                                ▷ O(1)
6:  AgregarAdelante(res, f.letra)                                            ▷ O(1)
7:  turno ← turnoApoyado(tablero, fila, columna)
8:  mientras (derecha <tamaño(tablero) ∧ Ocupada(tablero,fila, derecha) ∧
9:  turno>turnoApoyado(tablero, fila, derecha)) hacer                        ▷ O(L_max)
10:     letra_0 ← LetraEnPos(tablero, fila, derecha)                         ▷ O(1)
11:     AgregarAtras(res, letra_0)                                           ▷ O(1)
12:     derecha + +                                                          ▷ O(1)
13: end mientras
14: mientras ( 0 ≤ izquierda ∧ Ocupada(tablero,fila, izquierda) ∧
15: turnoApoyado>turnoApoyado(tablero, fila, izquierda)) hacer               ▷ O(L_max)
16:     letra_0 ← LetraEnPos(tablero, fila, izquierda)                       ▷ O(1)
17:     AgregarAdelante(res, letra_0)                                        ▷ O(1)
18:     izquierda − −                                                        ▷ O(1)
19: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{la ocurrencia es una jugada válida ∧ la ficha se encuentra en el tablero}
▷ Post{res es la palabra que se forma horizontalmente en el tablero con la ficha}

iPalabraFormadaTransversalVertical(**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                                  ▷ O(1)
2:  fila ← f.fila                                                            ▷ O(1)
3:  columna ← f.columna                                                      ▷ O(1)
4:  arriba ← f.fila − 1                                                      ▷ O(1)
5:  abajo ← f.fila + 1                                                       ▷ O(1)
6:  AgregarAdelante(res, f.letra)                                            ▷ O(1)
7:  turno ← turnoApoyado(tablero, fila, columna)
8:  mientras ( 0 ≤ arriba ∧ Ocupada(tablero,arriba,columna) ∧
9:  turno>turnoApoyado(tablero, arriba, columna)) hacer                      ▷ O(L_max)
10:     letra_0 ← LetraEnPos(tablero, arriba, columna)                       ▷ O(1)
11:     AgregarAdelante(res, letra_0)                                        ▷ O(1)
12:     arriba − −                                                           ▷ O(1)
13: end mientras
14: mientras (abajo <tamaño(tablero) ∧ Ocupada(tablero,abajo,columna) ∧
15: turno>turnoApoyado(tablero, abajo, columna)) hacer                       ▷ O(L_max)
16:     letra_0 ← LetraEnPos(tablero, abajo, columna)                        ▷ O(1)
17:     AgregarAtras(res, letra_0)                                           ▷ O(1)
18:     abajo + +                                                            ▷ O(1)
19: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{la ocurrencia es una jugada válida ∧ la ficha se encuentra en el tablero}
▷ Post{res es la palabra que se forma verticalmente en el tablero con la ficha}

calcularPuntos (**in** j : `juego`, **in** *palabras* : `cola(palabra)`) $\longrightarrow$ *res* : `nat`

1: $res \leftarrow 0$ ▷ $\mathcal{O}(1)$
2: **mientras** ($\neg$ vacia(palabras)) **hacer** ▷ $\mathcal{O}$(cantidad(palabras)$*L_{max}$)
3: $pal \leftarrow proximo(palabras)$ ▷ $\mathcal{O}(1)$
4: $it \leftarrow crearIT(pal)$ ▷ $\mathcal{O}(1)$
5: **mientras** haySiguiente(it) **hacer** ▷ $\mathcal{O}(L_{max})$
6: $l \leftarrow Siguiente(it)$ ▷ $\mathcal{O}(1)$
7: $res \leftarrow res + PuntajeDeLaLetra(varianteDeljuego(j), l)$ ▷ $\mathcal{O}(1)$
8: $Avanzar(it)$ ▷ $\mathcal{O}(1)$
9: **end mientras**
10: $desencolar(palabras)$ ▷ $\mathcal{O}(1)$
11: **end mientras**

Complejidad: $\mathcal{O}(cantidad(palabras) * L_{max})$, donde L_{max} es la longitud de la palabra más larga del alfabeto
▷ Pre{todas las palabras de la cola se encuentran en el tablero $\wedge$ todas forman una jugada válida}
▷ Post{calculamos el puntaje de las palabras}

4.1.2. Auxiliares Jugada válida

iTieneLasFichas(**in** *fichas* : `arreglo(nat)`, **in** o : `ocurrencia`) $\longrightarrow$ *res* : `bool`

1: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
2: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
3: $apariciones \leftarrow Lista(Tupla(letr : letra, cant : nat))Vacia$ ▷ $\mathcal{O}(1)$
4: **mientras** HaySiguiente(it) **hacer** ▷ $\mathcal{O}$(Cardinal(o)* Longitud(apariciones))
5: $letra0 \leftarrow Siguiente(it).letr$ ▷ $\mathcal{O}(1)$
6: $SumarAparicion(apariciones, letra0)$ ▷ $\mathcal{O}$(Longitud(apariciones)
7: $Avanzar(it)$ ▷ $\mathcal{O}(1)$
8: **end mientras**
9: $i \leftarrow 0$ ▷ $\mathcal{O}(1)$
10: **mientras** res $\wedge_L$ i <Longitud(apariciones) **hacer** ▷ $\mathcal{O}$(Longitud(apariciones)
11: $letra0 \leftarrow apariciones[i].letr$ ▷ $\mathcal{O}(1)$
12: $cant \leftarrow apariciones[i].cant$ ▷ $\mathcal{O}(1)$
13: **si** fichas[ord(letra0)] <cant **entonces** ▷ $\mathcal{O}(1)$
14: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
15: **end si** ▷ $\mathcal{O}(1)$
16: $i++$
17: **end mientras**
18: *return res*

Complejidad: $\mathcal{O}$(Cardinal(o)* Longitud(apariciones))= $\mathcal{O}((\text{Cardinal(o)})^2)$
La longitud de apariciones es menor o igual que el cardinal de la ocurrencia o
▷ Pre{para toda letra en la ocurrencia ord(letra) <fichas}
▷ Post{res es verdadero si y solo si la cantidad de apariciones de cada letra en la ocurrencia es menor o igual que fichas[ord(letra)]}

SumarAparicion(**in/out** *apariciones* : `Lista(Tupla(letra,nat))`, **in** l : `letra`)

1: $noEsta \leftarrow true$ ▷ $\mathcal{O}(1)$
2: $i \leftarrow 0$ ▷ $\mathcal{O}(1)$
3: **mientras** noEsta $\wedge_L$ i <Longitud(apariciones) **hacer** ▷ $\mathcal{O}$(Longitud(apariciones))
4: **si** apariciones[i].ltr == l **entonces** ▷ $\mathcal{O}(1)$
5: $noEsta \leftarrow false$ ▷ $\mathcal{O}(1)$
6: $apariciones[i].cant \leftarrow apariciones[i].cant$ ▷ $\mathcal{O}(1)$
7: **end si**
8: **end mientras**
9: **si** noEsta **entonces** ▷ $\mathcal{O}(1)$
10: $AgregarAtras(apariciones, Tupla(l, 1))$ ▷ $\mathcal{O}(1)$
11: **end si**

Complejidad: $\mathcal{O}$(Longitud(apariciones))
▷ Pre{true}
▷ Post{rapariciones se modifica de forma tal que si había una tupla con la letra l esta se modifica en el valor cant y si no hay una tupla con dicha letra, se agrega un una tupla con valores l y 1}

IOCURRENCIASINPOSICIONESREPETIDAS(**in** *o* : `ocurrencia`) $\longrightarrow$ *res* : `bool`

```
1:  res ← true                                                                  ▷ O(1)
2:  it0 ← crearIT(o)                                                            ▷ O(1)
3:  mientras HAYSIGUIENTE(IT0) ∧ RES hacer                                      ▷ O(Cardinal(o)^2)
4:      ficha0 ← Siguiente(it0)                                                 ▷ O(1)
5:      it1 ← Avanzar(it0)                                                      ▷ O(1)
6:      mientras HAYSIGUIENTE(IT1) ∧ RES hacer                                  ▷ O(Cardinal(o))
7:          ficha1 ← Siguiente(it1)                                             ▷ O(1)
8:          si (FICHA0.FILA == FICHA1.FILA ) ∧ (FICHA0.COLUMNA == FICHA1.COLUMNA) entonces  ▷ O(1)
9:              res ← false                                                     ▷ O(1)
10:         end si
11:         it1 ← Avanzar(it1)                                                  ▷ O(1)
12:     end mientras
13:     it0 ← Avanzar(it0)                                                      ▷ O(1)
14: end mientras
```

Complejidad: $\mathcal{O}(\text{Cardinal(o)}^2)$

▷ Pre{true}

▷ Post{para toda ficha en la ocurrencia o no existe una ficha en o tal que los campos fila y columna sean iguales y el campo letra no lo sea}

IMASCHICO(**in** *t1* : `ficha`, **in** *t2* : `ficha` **in** *sentido* : `bool`) $\longrightarrow$ *res* : `ficha`

```
1:  res ← t1                                                                    ▷ O(1)
2:  si SENTIDO entonces                                                         ▷ O(1)
3:      si T2.FILA < T1.FILA entonces                                           ▷ O(1)
4:          res ← t2                                                            ▷ O(1)
5:      end si
6:  else
7:      si T2.COLUMNA < T1.COLUMNA entonces                                     ▷ O(1)
8:          res ← t2                                                            ▷ O(1)
9:      end si
10: end si
```

Complejidad: $\mathcal{O}(1)$

▷ Pre{true}

▷ Post{si sentido es true res devuelve la ficha con el menor valor en el campo fila, de lo contrario devuelve la que tenga el menor valor en el campo columna}

IMASGRANDE(**in** *t1* : `ficha`, **in** *t2* : `ficha` **in** *sentido* : `bool`) $\longrightarrow$ *res* : `ficha`

```
1:  res ← t1                                                                    ▷ O(1)
2:  si SENTIDO entonces                                                         ▷ O(1)
3:      si T2.FILA > T1.FILA entonces                                           ▷ O(1)
4:          res ← t2                                                            ▷ O(1)
5:      end si
6:  else
7:      si T2.COLUMNA > T1.COLUMNA entonces                                     ▷ O(1)
8:          res ← t2                                                            ▷ O(1)
9:      end si
10: end si
```

Complejidad: $\mathcal{O}(1)$

▷ Pre{true}

▷ Post{si sentido es true res devuelve la ficha con el mayor valor en el campo fila, de lo contrario devuelve la que tenga el mayor valor en el campo}

IDISTANCIAENTREFICHAS(**in** *t1* : `ficha`, **in** *t2* : `ficha` **in** *sentido* : `bool`) $\longrightarrow$ *res* : `nat`

```
1:  res ← t2.columna − t1.columna                                               ▷ O(1)
2:  si SENTIDO entonces                                                         ▷ O(1)
3:      res ← t2.fila − t1.fila                                                 ▷ O(1)
4:  end si
```

Complejidad: $\mathcal{O}(1)$

▷ Pre{si sentido es true el valor del campo fila de t1 es menor que el de t2, si es false el campo columna de t1 es menor que el de t2}

▷ Post{si sentido es true devuelve la diferencia entre los campos de las filas, si es false, devuelve la diferencia entre los valores de las columnas}

---

iDistanciaNoMayorALongMax(**in** *o* : `ocurrencia`, **in** *sentido* : `bool`, **in** *largoMax* : `nat`) $\longrightarrow$ *res* : `bool`

```
1:  res ← true                                                          ▷ O(1)
2:  it ← crearIt(o)                                                     ▷ O(1)
3:  min ← Siguiente(it)                                                 ▷ O(1)
4:  max ← Siguiente(it)                                                 ▷ O(1)
5:  it ← Avanzar(it)                                                    ▷ O(1)
6:  mientras haySiguiente(it) hacer                                     ▷ O(cardinal(o))
7:      ficha0(it)                                                      ▷ O(1)
8:      min ← masChico(min, ficha0, sentido)                            ▷ O(1)
9:      max ← masGrande(max, ficha0, sentido)                           ▷ O(1)
10: end mientras
11: si largoMax < DistanciaEntreFichas(min, max , sentido) entonces     ▷ O(1)
12:     res ← false                                                     ▷ O(1)
13: end si
```

Complejidad: $\mathcal{O}$(cardinal(o))

▷ Pre{2 ≤ cardinal(o) y sentido es true si todas las fichas de o tienen mismo campo fila y es false si tienen mismo campo columna}

▷ Post{res es true si y solo siendo sentido true (false) la distancia entre las fichas con la menor y mayor fila (columna) de la ocurrencia o es menor o igual que largoMax}

iPalabrasTransversales(**in** *tab* : `tablero`, **in** *o* : `ocurrencia`, **in** *sentido* : `bool`) $\longrightarrow$ *res* : `cola(palabra)`

```
1:  it ← crearIT(o)                                                     ▷ O(1)
2:  palabras ← Cola(palabra)                                            ▷ O(1)
3:  mientras haySiguiente(it) hacer                                     ▷ O(Cardinal(o) ∗L_max)
4:      ficha ← Siguiente(it)                                           ▷ O(1)
5:      palabra ← CrearLista(letra)                                     ▷ O(1)
6:      si sentido entonces
7:          pal ← palabraTransversalVertical(tablero, ficha)            ▷ O(L_max)
8:      else
9:          pal ← palabraTransversalHorizontal(tablero, ficha)          ▷ O(L_max)
10:     end si
11:     encolar(palabras, pal)                                          ▷ O(1)
12:     avanzar(it)                                                     ▷ O(1)
13: end mientras
```

Complejidad: $\mathcal{O}$(longitud(o)∗L_{max}) donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante

▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}

▷ Post{con sentido true (false) res es igual a una cola de palabras cuyas palabras son las que se forman al recorrer verticalmente (horizontalmente) el tablero para cada ficha en la ocurrencia o }

iPalabraTransversalHorizontal(**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                             ▷ O(1)
2:  fila ← f.fila                                                       ▷ O(1)
3:  derecha ← f.columna + 1                                             ▷ O(1)
4:  izquierda ← f.columna − 1                                           ▷ O(1)
5:  AgregarAdelante(res, f.letra)                                       ▷ O(1)
6:  mientras (derecha <tamaño(tablero) ∧ Ocupada(tablero,fila, derecha)) hacer   ▷ O(L_max)
7:      letra_0 ← LetraEnPos(tablero, fila, derecha)                    ▷ O(1)
8:      AgregarAtras(res, letra_0)                                      ▷ O(1)
9:      derecha + +                                                     ▷ O(1)
10: end mientras
11: mientras ( 0 ≤ izquierda ∧ Ocupada(tablero,fila, izquierda)) hacer  ▷ O(L_max)
12:     letra_0 ← LetraEnPos(tablero, fila, izquierda)                  ▷ O(1)
13:     AgregarAdelante(res, letra_0)                                   ▷ O(1)
14:     izquierda − −                                                   ▷ O(1)
15: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$ donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante

▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}

▷ Post{res es igual a la palabra maximal que se forma recorriendo el tablero a partir de la ficha t0 variando la coordenada de columna}

iPalabraTransversalVertical(**in** *tab* : `tablero`, **in** *f* : `ficha`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                                    ▷ O(1)
2:  fila ← f.fila                                                              ▷ O(1)
3:  columna ← f.columna                                                        ▷ O(1)
4:  arriba ← f.fila − 1                                                        ▷ O(1)
5:  abajo ← f.fila + 1                                                         ▷ O(1)
6:  AgregarAdelante(res, f.letra)                                              ▷ O(1)
7:  mientras ( 0 ≤ ARRIBA ∧ OCUPADA(TABLERO,ARRIBA,COLUMNA) ∧
8:  TURNO>TURNOAPOYADO(TABLERO, FILA, IZQUIERDA)) hacer                        ▷ O(L_max)
9:      letra_0 ← LetraEnPos(tablero, arriba, columna)                         ▷ O(1)
10:     AgregarAdelante(res, letra_0)                                          ▷ O(1)
11:     arriba − −                                                             ▷ O(1)
12: end mientras
13: mientras (ABAJO <TAMAÑO(TABLERO) ∧ OCUPADA(TABLERO,ABAJO,COLUMNA)) hacer   ▷ O(L_max)
14:     letra_0 ← LetraEnPos(tablero, abajo, columna)                          ▷ O(1)
15:     AgregarAtras(res, letra_0)                                             ▷ O(1)
16:     abajo + +                                                              ▷ O(1)
17: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$ donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante

▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}

▷ Post{res es igual a la palabra maximal que se forma recorriendo el tablero a partir de la ficha t0 variando la coordenada de fila}

iocurrenciaAvector(**in** *o* : `ocurrencia`) $\longrightarrow$ *res* : `vector(ficha`

```
1:  res ← crearVector(ficha)                                                   ▷ O(1)
2:  it ← crearIT(o)                                                            ▷ O(1)
3:  mientras HAYSIGUIENTE(IT) hacer                                            ▷ O(longitud(o))
4:      ficha ← Siguiente(it)                                                  ▷ O(1)
5:      AgregarAtras(res, ficha)                                               ▷ O(1)
6:      it ← Avanzar(it)                                                       ▷ O(1)
7:  end mientras
```

Complejidad: $\mathcal{O}$(longitud(o))

▷ Pre{true}

▷ Post{para toda ficha f: f pertenece a la ocurrencia o si y solo si está en el vector res}

iordenarVectorDeFichas(**in/out** *vect* : `vector(ocurrencia)`)

```
1:  swapped ← true                                                             ▷ O(1)
2:  i ← longitud(vect)                                                         ▷ O(1)
3:  mientras SWAPPED hacer                                                     ▷ O(longitud(vect)^2)
4:      swapped ← false                                                        ▷ O(1)
5:      j ← 0                                                                  ▷ O(1)
6:      mientras J <I hacer                                                    ▷ O(i-j)
7:          mayorFila ← vect[j].fila > vect[j + 1].fila                        ▷ O(1)
8:          mayorCol ← vect[j].columna > vect[j + 1].columna                   ▷ O(1)
9:          si MAYORFILA ∨ MAYORCOL entonces
10:             temp ← vect[j]                                                 ▷ O(1)
11:             vect[j] ← vect[j + 1]                                          ▷ O(1)
12:             vect[j + 1] ← temp                                             ▷ O(1)
13:             swapped ← true                                                 ▷ O(1)
14:         end si
15:         j ← j + 1                                                          ▷ O(1)
16:     end mientras
17:     i ← i − 1                                                              ▷ O(1)
18: end mientras
```

Complejidad: $\mathcal{O}$(longitud(vect)^2)

▷ Pre{todas las fichas en la ocurrencia o tienen o bien igual valor en el campo fila, o bien, todas tienen igual valor en el campo columna}

▷ Post{si las fichas tienen todas igual fila (columna) devuelve el vector con éstas ordenadas segun su columna(fila)}

iFormarPalabraPrincipal(**in** *tab* : `tablero`, **in** *o* : `ocurrencia` , **in** *sentido* : `bool`)) $\longrightarrow$ *res* : `palabra`

```
1:  fichas ← ocurrenciaAVector(o)                                   ▷ O(longitud(o))
2:  ordenarVectorDeFichas(fichas)                                   ▷ O(longitud(o)^2)
3:  res ← crearLista(letra)                                         ▷ O(1)
4:  si SENTIDO entonces                                             ▷ O(L_max)
5:      res ← iPrincipalHorizontal(tablero, fichas)                 ▷ O(L_max)
6:  else
7:      res ← iPrincipalVertical(tablero, fichas)                   ▷ O(L_max)
8:  end si
```

Complejidad: $\mathcal{O}(L_{max}+$ longitud(o)$^2))$ donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante
▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}
▷ Post{con sentido con valor true (false) res es igual a la palabra maximal horizontal (vertical) que se forma en el tablero poniendo las fichas de la ocurrencia o}

iPrincipalHorizontal (**in** *tab* : `tablero`, **in** *fichas* : `vector(ficha)`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                                 ▷ O(1)
2:  AgregarAtras(res, fichas[0].letra)                                      ▷ O(1)
3:  fila ← fichas[0].fila                                                   ▷ O(1)
4:  columna ← fichas[0].columna                                             ▷ O(1)
5:  izquierda ← columna − 1                                                 ▷ O(1)
6:  derecha ← columna + 1                                                   ▷ O(1)
7:  i ← 1                                                                   ▷ O(1)
8:  esContigua ← true                                                       ▷ O(1)
9:  termino ← false                                                         ▷ O(1)
10: mientras 0 ≤ IZQUIERDA ∧ hayLetra?(TABLERO, FILA, IZQUIERDA) hacer      ▷ O(L_max)
11:     letraIzquierda ← LetraEnPos(tablero, fila, izquierda)               ▷ O(1)
12:     AgregarAdelante(res, letraIzquierda)                                ▷ O(1)
13:     izquierda − −                                                       ▷ O(1)
14: end mientras
15: mientras (DERECHA <TAMAÑO(TABLERO) ∧ /TERMINO ∧ esContigua hacer        ▷ O(L_max)
16:     si hayLetra?(TABLERO,FILA, DERECHA) entonces                        ▷ O(1)
17:         letraDerecha ← LetraEnPos(tablero, fila, derecha)               ▷ O(1)
18:         AgregarAtras(res, letraDerecha)                                 ▷ O(1)
19:     else
20:         si I= LONGITUD(FICHAS) entonces                                 ▷ O(1)
21:             termino ← true                                              ▷ O(1)
22:         else
23:             si FICHAS[I].COLUMNA = DERECHA entonces                     ▷ O(1)
24:                 LetraNueva ← fichas[i].letra                            ▷ O(1)
25:                 AgregarAtras(res, LetraNueva)                           ▷ O(1)
26:                 i + +                                                   ▷ O(1)
27:             else
28:                 res ← CrearLista(letras)                                ▷ O(1)
29:                 esContigua ← false                                      ▷ O(1)
30:             end si
31:         end si
32:     end si
33:     derecha + +                                                         ▷ O(1)
34: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$ donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante
▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}
▷ Post{res es igual a la palabra maximal que se forma recorriendo el tablero a partir de la ficha t0 variando el valor de la coordenada columna}

iPrincipalVertical(**in** *tab* : `tablero`, **in** *fichas* : `vector(ficha)`) $\longrightarrow$ *res* : `palabra`

```
1:  res ← crearLista(letra)                                                    ▷ O(1)
2:  AgregarAtras(res, fichas[0].letra)                                         ▷ O(1)
3:  fila ← fichas[0].fila                                                      ▷ O(1)
4:  columna ← fichas[0].columna                                                ▷ O(1)
5:  arriba ← fila − 1                                                          ▷ O(1)
6:  abajo ← fila + 1                                                           ▷ O(1)
7:  i ← 1                                                                      ▷ O(1)
8:  esContigua ← true                                                          ▷ O(1)
9:  termino ← false                                                            ▷ O(1)
10: mientras (0 ≤ ARRIBA ∧ hayLetra?(TABLERO,ARRIBA,COLUMNA)) hacer            ▷ O(L_max)
11:     letraArriba ← LetraEnPos(tablero, arriba, columna)                     ▷ O(1)
12:     AgregarAdelante(res, letraArriba)                                      ▷ O(1)
13:     arriba − −                                                             ▷ O(1)
14: end mientras
15: mientras (ABAJO < TAMAÑO(TABLERO) ∧_L ¬ TERMINO ∧_L esContigua) hacer       ▷ O(L_max)
16:     si hayLetra?(TABLERO,ABAJO,COLUMNA) entonces                           ▷ O(1)
17:         letraAbajo ← LetraEnPos(tablero, abajo, columna)                   ▷ O(1)
18:         AgregarAtras(res, letraAbajo)                                      ▷ O(1)
19:     else
20:         si I= LONGITUD(FICHAS) entonces                                    ▷ O(1)
21:             termino ← true                                                 ▷ O(1)
22:         else
23:             si FICHAS[I].FILA = ABAJO entonces                             ▷ O(1)
24:                 LetraNueva ← fichas[i].letra                               ▷ O(1)
25:                 AgregarAtras(res, LetraNueva)                              ▷ O(1)
26:                 i + +                                                      ▷ O(1)
27:             else
28:                 res ← CrearLista(letras)                                   ▷ O(1)
29:                 esContigua ← false                                         ▷ O(1)
30:             end si
31:         end si
32:     end si
33:     abajo + +                                                              ▷ O(1)
34: end mientras
```

Complejidad: $\mathcal{O}(L_{max})$ donde L_{max} es la máxima longitud de una palabra en el conjTrie de la variante

▷ Pre{en la ocurrencia: no hay fichas con misma posicion y distinta letra, para todas las fichas sus valores fila y columna son posiciones vacías en el tablero}

▷ Post{res es igual a la palabra maximal que se forma recorriendo el tablero a partir de la ficha t0 variando el valor de la coordenada columna}

5. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `servidor`.

Operaciones básicas de Servidor

CREARNUEVOSERVIDOR(**in** *cantJugadores* : `nat`, **in** *variante* : `var`, **in** *bolsaDeFichas* : `cola(letras)`) → *res* : `serv`
Pre ≡ {$long(bolsaDeFichas) \geq tamañoTablero(variante) * tamañoTablero(variante) + cantJugadores * \#fichas(variante)$}
Post ≡ {$s =_{\text{obs}} nuevoServidor(cantJugadores, variante, bolsaDeFichas)$}
Complejidad: $\mathcal{O}$(tamaño del tablero del juego)^2 + cantidad de letras en el alfabeto de la variante del juego * cantidad de jugadores + cantidad de fichas por jugador * cantidad de jugadores)
Descripción: crea un nuevo servidor
Aliasing: No aplica

CONECTARCLIENTEALSERVIDOR(**in/out** s : `serv`)
Pre ≡ {$s = s_0 \wedge \neg empezo?(s)$}
Post ≡ {$s =_{\text{obs}} conectarCliente(s)$}
Complejidad: $\mathcal{O}(1)$
Descripción: conecta un nuevo cliente al servidor
Aliasing: No aplica

NOTIFICACIONES(**in/out** s : `servidor`) → *res* : `cola(notif)`
Pre ≡ {$s = s_0 \wedge jugador < \#conectados(s)$}
Post ≡ {$res =_{\text{obs}} secuACola(notificaciones(s_0, jugador)) \wedge s =_{\text{obs}} consultar(s_0, jugador)$}
Complejidad: $\mathcal{O}(n)$, donde n es la cantidad de notificaciones en la cola del jugador desde la ultima vez que se llamó a la función
Descripción: devuelve la cola de notificaciones del jugador y la vacía
Aliasing: Se devuelve una referencia no modificable a algunas notificaciones del jugador (las notificaciones para todos)

RECIBIRMENSAJE(**in/out** s : `serv`, **in** *cid* : `nat`, **in** o : `ocurrencia`)
Pre ≡ {$s = s_0 \wedge (0 \leq cid < \#conectados(s))$}
Post ≡ {$s =_{\text{obs}} recibirMensaje(s_0, cid, o)$}
Complejidad: $\mathcal{O}$((longitud de la palabra mas larga)^2 + cantidad de letras del alfabeto + cantidad de fichas apoyadas)
Descripción: recibimos un nuevo mensaje del jugador cid
Aliasing: No aplica

#JUGESPERADOS(**in** s : `servidor`) → *res* : `nat`
Pre ≡ {$true$}
Post ≡ {$res =_{\text{obs}} \#esperados(s)$}
Complejidad: $\mathcal{O}(1)$
Descripción: indica la cantidad de jugadores esperados en el servidor
Aliasing: No aplica

#JUGCONECTADOS(**in** s : `servidor`) → *res* : `nat`
Pre ≡ {$true$}
Post ≡ {$res =_{\text{obs}} \#conectados(s)$}
Complejidad: $\mathcal{O}(1)$
Descripción: indica la cantidad de jugadores conectados en el servidor
Aliasing: No aplica

JUEGODELSERVIDOR(**in** s : `servidor`) → *res* : `juego`
Pre ≡ {$empezo?(s)$}
Post ≡ {$res =_{\text{obs}} juego(s)$}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el juego que se está jugando en el servidor servidor
Aliasing: Se devuelve una referencia no modificable al juego

EMPEZOJUEGO(**in** s : `serv`) → *res* : `bool`
Pre ≡ {$true$}
Post ≡ {$\#conectados(s) =_{\text{obs}} \#esperados(s)$}
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el juego de un servido empezó

Aliasing: No aplica

Representación

`servidor` se representa con `serv`

donde `serv` es `tupla`(*#jugadoresConectados*: `nat`, *#jugadoresEsperados*: `nat`, *notificacionesPersonales*: `Arreglo(Cola(tupla`⟨ *notificaciones* : `notif`, *contador* : `nat` ⟩, *notificacionesParaTodos*: `tupla` ⟨ *cantidadVistos* : `arreglo(nat)`, *notis* : `vector(tupla` ⟨ *notificacion* : `notif`, *contador* : `nat` ⟩⟩⟩ , *cantidadDeNotificaciones*: `nat`, *jueg*: `juego`)

Rep : serv ⟶ bool

Rep(s) ≡ true ⟺ (long(repositorio(s.juego)) >0) ∧
(s.#jugadoresEsperados >0) ∧
(s.#jugadoresEsperados = long(s.notificacionesPersonales) = long(s.notificacionesParaTodos.cantidadVistos) = long(s.ordenDeNotificaciones)) ∧
(cantDeNotis(s.notificacionesPersonales) + long(s.notificacionesParaTodos.notis) = long(s.cantidadDeNotificaciones) ∧
empezoJuego(s) ⇒$_L$ (cantJugadores(s.juego) = s.#jugadoresEsperados)

Abs : serv s ⟶ servidor {Rep(s)}

Abs(s) =$_{\text{obs}}$ se: servidor | #esperados(se) = s.#jugadoresEsperados ∧
#conectados(se) = s.#jugadoresConectador ∧
(empezo?(se) ⇒$_L$ juego(se) = s.juego) ∧
(∀i:nat)(0 ≤ i <#conectados(se) ⇒$_L$ notificaciones(se, i) =
vectorASecu(ordenarNotificaciones(s.notificacionesPersonales[i-1],
s.notificacionesParaTodos.notis,
s.notificacionesParaTodos.cantidadVistos, s.cantidadDeNotificaciones)))

Algoritmos

ICREARNUEVOSERVIDOR(**in** *cantJugadores* : `nat`, **in** *variante* : `var`, **in** *bolsaDeFichas* : `cola(letra)`) ⟶ *res* : `serv`

1: $res.\#jugadoresConectados \leftarrow 0$ ▷ $\mathcal{O}(1)$
2: $res.\#jugadoresEsperados \leftarrow$cantJugadores ▷ $\mathcal{O}(1)$
3: $res.notificacionesPersonales \leftarrow$CrearArreglo[cantJugadores] ▷ $\mathcal{O}$(cantidad de jugadores)
4: $res.notificacionesParaTodos.cantidadVistos \leftarrow$CrearArreglo[cantJugadores] ▷ $\mathcal{O}$(cantidad de jugadores)
5: $res.notificacionesParaTodos.notis \leftarrow$CrearVector() ▷ $\mathcal{O}(1)$
6: $res.cantidadDeNotificaciones \leftarrow 0$ ▷ $\mathcal{O}(1)$
7: $res.juego \leftarrow$crearNuevoJuego(cantJugadores, variante, bolsaDeFichas) ▷ complejidad de crear nuevo juego

Complejidad: $\mathcal{O}$((tamaño del tablero del juego)^2+cantidad de letras en el alfabeto de la variante del juego * cantidad de jugadores+cantidad de fichas por jugador * cantidad de jugadores)

ICONECTARCLIENTEALSERVIDOR(**in/out** s : `serv`)

1: $\#conectados \leftarrow s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$
2: $\#esperados \leftarrow s.\#jugadoresEsperados$ ▷ $\mathcal{O}(1)$
3: $conectados++$ ▷ $\mathcal{O}(1)$
4: $num \leftarrow s.cantidadDeNotificaciones$ ▷ $\mathcal{O}(1)$
5: $enviarNotiPers(s.notificacionesPersonales[conectados-1], IDCliente[conectados], num)$ ▷ $\mathcal{O}(1)$
6: **si** #CONECTADOS<#ESPERADOS **entonces** ▷ $\mathcal{O}(1)$ amortizado =($\mathcal{O}$(cantidad de notificaciones en el vector notificaciones para todos))
7: $enviarNotiTodos(s.notificacionesParaTodos.notis, Empezar(s.juego.variante.tamaño), num)$ ▷ $\mathcal{O}(1)$ amortizado
8: $enviarNotiTodos(s.notificacionesParaTodos.notis, TurnoDe(0), num)$ ▷ $\mathcal{O}(1)$
9: **end si**

Complejidad: $\mathcal{O}(1)$

INOTIFICACIONES(**in/out** s : `serv`, **in** *cid* : `nat`) ⟶ *res* : `cola(notif)`

1: $res \leftarrow$iOrdenarNotificaciones(s, cid) ▷ $\mathcal{O}$(cantidad de notificaciones del jugador)
2: $s.notificacionesPersonales[cid-1] \leftarrow$crearCola(tupla<notif, nat>) ▷ $\mathcal{O}(1)$
3: $notisTodos \leftarrow$s.notificacionesParaTodos.notis ▷ $\mathcal{O}(1)$
4: $s.notificacionesParaTodos.cantidadVistos[cid-1] \leftarrow$longitud(notisTodos) ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}$(cantidad de notificaciones del jugador)

IRECIBIRMENSAJENUEVO(in/out s : serv, in cid : nat, in o : nat)

1: $esValida$ ←iempezó?(s)∧cid = turno(s.juego) ▷ $\mathcal{O}(1)$
2: $palabrasFormadas$ ←CrearCola(lista(letra)) ▷ $\mathcal{O}(1)$

3: **si** ESVALIDA **entonces**
4: $jugValida$ ←jugadaValida(s.jueg, o) ▷ $\mathcal{O}((L_{max})^2 + cantdeletrasdelalfabeto)$
5: **end si**
6: **si** JUGVALIDA **entonces**
7: $fichasAnteriores$ ←ifichasDeJugador(s.juego, cid) ▷ $\mathcal{O}(1)$
8: $puntajePrevio$ ←puntaje(s.juego, cid) ▷ En este contexto, esto es equivalente a $\mathcal{O}(1)$, porque después de realizar una jugada, siempre se llama a la función

9: $ubicarfichas(s.juego, o)$ ▷ $\mathcal{O}$(cantidad de fichas apoyadas en la jugada)
10: $fichasPostJugada$ ←fichasDeJugador (s.juego, cid) ▷ $\mathcal{O}(1)$
11: num ←s.cantidadDeNotificaciones ▷ $\mathcal{O}(1)$

12: $ienviarNotiTodos(s.notificacionesParaTodos.notis, Ubicar(cid, o), num)$ ▷ $\mathcal{O}(1)$ amortizada

13: $puntajePostJugada$ ←puntaje(s.juego, cid) ▷ En este contexto, es $\mathcal{O}$(cantidad de fichas apoyadas), porque previo a realizar la jugada siempre se llama a la funcion, entonces solo tiene que calcular los puntos que suman las palabras generadas por las fichas apoyadas en esta jugada

14: $ienviarNotiTodos(s.notificacionesParaTodos.notis, SumarPuntos(cid, puntajePostJugada - puntajePrevio), num)$ ▷ $\mathcal{O}(1)$ amortizada
15: $ienviarNotiTodos(s.notificacionesParaTodos.notis, TurnoDe(s.j.turno_de), num)$ ▷ $\mathcal{O}(1)$ amortizada

16: $fichasRepuestas$ ←CrearMulticonjunto(letra) ▷ $\mathcal{O}(1)$
17: **para** I = 0 . . . LONGITUD(FICHASPOSTJUGADA)-1 **hacer** ▷ $\mathcal{O}$(cantidad de fichas apoyadas)
18: **si** FICHASANTERIORES[I] <FICHASPOSTJUGADA[I] **entonces**
19: $cant$ ←fichasPostJugada[i] - fichasAnteriores[i] ▷ $\mathcal{O}(1)$
20: **mientras** CANT >0 **hacer**
21: $Agregar(ord^{-1}(i), fichasRepuestas)$ ▷ $\mathcal{O}(1)$
22: $cant--$ ▷ $\mathcal{O}(1)$
23: **end mientras**
24: **end si**
25: **end para**

26: $ienviarNotiPers(s.notificacionesPersonales[cid-1], Reponer(fichasRepuestas), num)$ ▷ $\mathcal{O}(1)$
27: **else**
28: $ienviarNotiPers(s.notificacionesPersonales[cid-1], Mal, num)$ ▷ $\mathcal{O}(1)$
29: **end si**

Complejidad: $\mathcal{O}(1)$

I#JUGESPERADOS(in s : serv) ⟶ res : nat

1: $res \leftarrow s.\#jugadoresEsperados$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

I#JUGCONECTADOS(in s : serv) ⟶ res : nat

1: $res \leftarrow s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

IJUEGODELSERVIDOR(in s : serv) ⟶ res : juego

1: res ←s.juego ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

IEMPEZOJUEGO(in s : serv) ⟶ res : bool

1: $res \leftarrow s.\ \#jugadoresEsperados = s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

5.1. Funciones Auxiliares

iOrdenarNotificaciones(**in** *s* : `serv`, **in** *cid* : `nat`) $\longrightarrow$ *res* : `cola(notif)`

1: $notisPers \leftarrow colaAVector(s.notificacionesPersonales[cid-1])$ ▷ $\mathcal{O}$(#notificaciones personales del jugador)
2: $prim \leftarrow s.notificacionesParaTodos.cantidadVistos[cid-1]$ ▷ $\mathcal{O}(1)$
3: **si** prim<longitud(s.notificacionesParaTodos.notis **entonces**
4: $notisTodos \leftarrow CrearVector(tupla < notificacion : notif, contador : nat >)$ ▷ $\mathcal{O}(1)$
5: **para** i=prim ... longitud(s.notificacionesParaTodos.notis) **hacer** ▷ $\mathcal{O}$(#notificaciones de todos que no vio el jugador)
6: $Agregar(notisTodos, s.notificacionesParaTodos.notis[i])$ ▷ $\mathcal{O}(1)$ amortizado
7: **end para**
8: $res \leftarrow crearCola(notif)$ ▷ $\mathcal{O}(1)$
9: $Merge(res, notisPers, notisTodos)$ ▷ $\mathcal{O}$(#notificaciones totales del jugador)
10: **else**
11: $res \leftarrow s.notificacionesPersonales[cid-1]$ ▷ $\mathcal{O}(1)$
12: **end si**

Complejidad: $\mathcal{O}$(cantidad de notificaciones del jugador)

Pre {cid <#conectados(s)}
Post{res =$_{obs}$ secuACola(notificaciones(s, cid))}

Merge(**in** *pers* : `Vector(tupla<notif, nat>)`, **in** *todos* : `vector(tupla<notif, nat>)`) $\longrightarrow$ *res* : `cola(notif)`

1: $iP \leftarrow 0$ ▷ $\mathcal{O}(1)$
2: $iT \leftarrow 0$ ▷ $\mathcal{O}(1)$
3: $res \leftarrow crearCola(notif)$ ▷ $\mathcal{O}(1)$

4: **si** iP <tamaño(pers) $\wedge$(iT $\geq$ tamaño(todos) $\vee$ pers[iP].second < todos[iT].second) **entonces** ▷ $\mathcal{O}$(longitud(pers)+longitud(todos)
5: $encolar(res, pers[iP].first)$ ▷ $\mathcal{O}(1)$
6: $iP++$ ▷ $\mathcal{O}(1)$
7: **else**
8: $encolar(res, todos[iP].first)$ ▷ $\mathcal{O}(1)$
9: $iT++$ ▷ $\mathcal{O}(1)$
10: **end si**

Complejidad: $\mathcal{O}$(longitud(pers)+longitud(todos))

Pre {ordenado?(pers) $\wedge$ ordenado?(todos)}
Post{ordenado?(res) $\wedge$ esPermutación(res,concatenar(pers,todos))}

---

ienviarNotiPers(**in/out** *notisP* : `Cola(tupla<notif, nat>)`, **in** *notificacion* : `notif`, **in/out** *num* : `nat`)

1: $encolar(notisP, tupla < notificacion, num >)$ ▷ $\mathcal{O}(1)$
2: $num++$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Pre {notisP = notisP0 $\wedge num = num0$}
$Post\{notisP = encolar(notificacion, notisP0) \wedge num = num0 + 1\}$

---

ienviarNotiTodos(**in/out** *notisT* : `Vector(tupla<notif, nat>)`, **in** *notificacion* : `notif`, **in/out** *num* : `nat`)

1: $agregarAtras(notisT, tupla < notificacion, num >)$ ▷ $\mathcal{O}(1)$ amortizada
2: $num++$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

Pre {notisT = notisT0 $\wedge num = num0$}
$Post\{notisT = agregarAtras(notificacion, notisP0) \wedge num = num0 + 1\}$

---

icantDeNotis(**in** *notis* : `Arreglo(Cola(tupla<notif, nat>))` $\longrightarrow$ *res* : `nat`

```
1: res ← 0                                                          ▷ O(1)
2: para i = 0 ... longitud(notis) hacer                             ▷ O(longitud(notis))
3:     res ← res + tamaño(notis[i])                                 ▷ O(1)
4: end para
```

Complejidad: $\mathcal{O}$(longitud(notis))

Pre {true}
Post{res es la suma de la cantidad de notificaciones personales de cada jugador}

6. Módulo Conjunto Trie

El módulo Conjunto Trie implementa un conjunto de palabra (lista de letra) sobre Trie. Las palabras se podrán formar solo con una cantidad K de letras distintas definida a la hora de iniciar un conjunto

Para esto suponemos que existe un módulo letra que esta dotado de dos operaciones, que llamaremos ord y ord^{-1} que establecen una biyección entre valores del módulo Nat y el módulo letra, y viceversa, respectivamente, siendo además una la inversa de la otra.

Es una implementación parcial por lo que provee un conjunto vacío en el que se puede insertar, y testear pertenencia, no existiendo las operaciones de borrado entre otras. La complejidad en el caso del testeo de pertenencia se realiza en un tiempo acotado dependiente del largo de la palabra más larga del conjunto. La inserción de un elemento siempre tomará en peor caso O(Longitud(palabra)).

Interfaz

usa: `conj(Palabra)`, `itConj(Palabra)`, `itLista(letra)`, `lista(letra)`, `letra`.

se explica con: Conj(Palabra)

géneros: `conjTrie`.

Operaciones básicas de conjunto

Vacío(**in** n : `Nat`) → res : `conjTrie`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} \emptyset$ }
Complejidad: $\Theta(n)$
Descripción: genera un conjunto de palabras vacío. El n ingresado corresponde al cardinal del conjunto de letras del que va a disponer el árbol para formar sus elementos (que son del tipo lista(letra) renombrado palabra).
Aliasing: devuelve una referencia modificable a un conjTrie vacío.

Agregar(**in/out** c: `conjTrie, in` p : `Palabra`)
Pre ≡ {$c =_{\text{obs}} c_0$}
Post ≡ {$c =_{\text{obs}} Ag(p, c_0)$}
Complejidad: $\Theta(K * Longitud(p))$ donde k es el natural que se pasó a Vacío para crear c (tamaño del alfabeto)
Descripción: se recorre el arbol agregando nodos al mismo de ser necesarios hasta llegar a una hoja correspondiente a la última letra de la palabra a agregar, en cuyo nodo se cambia el valor de definida a True, en caso de ser la palabra vacía se cambia el valor de dicho campo del nodo raíz.

EsVacío?(**in** c: `conjTrie`) → res : `bool`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} \emptyset?(c)$}
Complejidad: $\Theta(K)$ donde k es el natural que se pasó a Vacío para crear c (tamaño del alfabeto)
Descripción: devuelve `true` si y sólo si c esta vacío.

Pertenece?(**in** c: `conjTrie`, **in** p: `Palabra`) → res : `bool`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} p \in c$}
Complejidad: $\mathcal{O}(Longitud(p))$
Descripción: devuelve `true` si y sólo a pertenece al conjunto.

LargoDeClaveMax(**in** c: `estr`) → res : `Nat`
Pre ≡ {$\neg\emptyset?$(e) }
Post ≡ {$(\forall p : palabra)(p \in c \rightarrow_L Long(p) \leq res) \wedge (\exists p : palabra)(p \in c \wedge Long(p) = res)$ }
Complejidad: $\mathcal{O}(1)$
Descripción: para un conjunto no vacío devuelve la longitud de la palabra más grande en el mismo.

AgregarPalabras(**in/out** c: `conjTrie, in` ps : `conj(Palabra)`)
Pre ≡ {$c =_{\text{obs}} c_0$}
Post ≡ {$c =_{\text{obs}} (c_0 \cup ps)$}
Complejidad: $\Theta\left(K * \sum_{p \in ps} Longitud(p)\right)$ donde k es el natural que se pasó a Vacío para crear c (tamaño del alfabeto)
Descripción: agrega palabras al conjunto trie desde un conjunto lineal.

6.1. Especificación de las operaciones auxiliares utilizadas

TAD Conjunto Extendido Trie(palabra)

Otras operaciones

alturaAcotadaPorK : nodo×nat×nat ⟶ bool

nodosAcotados : secu(puntero(nodo))×nat×nat ⟶ bool

tamañoHijos : nodo ⟶ nat

definePalabra? : nodo ⟶ bool

esHoja? : nodo ⟶ bool

esHojaAux : secu(puntero(nodo)) $\longrightarrow$ bool

NodoSuelto? : nodo $\longrightarrow$ bool

noHayNodosSueltos : secu(puntero(nodo)) $\longrightarrow$ bool

nodosMismoTamaño : secu(puntero(nodo))×nat $\longrightarrow$ bool

definida? : palabra×nodo $\longrightarrow$ bool

Axiomas

alturaAcotadaPorK$(n, k, long) \equiv long < k \wedge nodosAcotados(n.hijos, k, long + 1)$

nodosAcotados$(nds, k, long) \equiv (prim(nds) = null \vee_L alturaAcotadaPorK(prim(nds) \rightarrow dato, k, long + 1)) \wedge_L nodosAcotados(fin(nds), k, long)$

tamañoHijos$(n) \equiv long(n.hijos)$

definePalabra?$(n) \equiv n.definida$

esHoja?$(n) \equiv esHojaAux(n.hijos)$

esHojaAux$(hs) \equiv vacia?(hs) \vee_L (prim(hs) = null \wedge_L esHojaAux(fin(hs)))$

NodoSuelto?$(n) \equiv esHoja(n) \wedge_L \neg\, definePalabra(n)$

noHayNodosSueltos$(punteros_nodos) \equiv$ *vacia?(punteros_nodos)* $\wedge_L$ ((*punteroPrim = Null*) $\vee_L$ ($\neg$ *NodoSuelto?(nodoPrim)* $\wedge_L$ *noHayNodosSueltos(nodoPrim.hijos)*)) $\wedge_L$ *noHayNodosSueltos(fin(punteros_nodos))*

nodosMismoTamaño$(punteros_nodos, k) \equiv$ **if** $vacia?(punteros_nodos)$ **then** $True$ **else** $((punteroPrim = null) \vee_L (TamaoHijos(nodoPrim.hijos) = k \wedge_L nodosMismoTamao(nodoPrim.hijos, k))) \wedge_L nodosMismoTamao(fin(punteros_nodos))$ **fi**

definida?$(p, n) \equiv (esVacia?(p) \wedge n.definida) \vee_L (\neg esVacia?(p) \wedge_L \neg(puntero\_num\_sig\_letra = null) \wedge_L definida?(fin(p), nodo\_siguiente))$

.

Reemplazos sintácticos:
punteroPrim ≡ prim(punteros_nodos)
nodoPrim ≡ punteroPrim→dato
puntero_num_sig_letra ≡ n.hijos[ord^{-1}($prim(p)$)]
nodo_siguiente≡ puntero_num_sig_letra → dato

Fin TAD

Representación

Representación del Conjunto

En este módulo vamos a implementar un conjunto sobre Trie utilizando para ello un módulo auxiliar llamado nodo, que va a contar con dos campos: "definida"de tipo booleano e "hijos"de tipo arreglo dimensionable de punteros a nodo del tamaño del alfabeto que se esté empleando para formar los elementos del conjunto que corresponden a palabras (representadas por el tipo Lista de letras). El conjunto entonces se representa con un puntero a nodo llamado raíz", correspondiendo el valor de su campo 'definida' a la pertenencia o no de la palabra vacía, y un natural 'maxlong' que representará, para un conjunto no vacío, la longitud de la palabra más larga en el mismo. Luego mediante el arreglo de punteros 'hijos' de la raíz se podrá recorrer el arbol tanto para verificar la pertenencia o no de una palabra como para agregarla.
El invariante va a restringir de forma tal que nunca existan nodos hoja que no correspondan a la última letra de un elementro del conjunto, además de evitar que los hijos de un nodo apunten a nodos de una altura menor en la estructura, buscando así que no se formen bucles en nuestra estructura; esto lo hacemos con un predicado lógico que verifica que la altura del árbol está acotada. Esta es una modularización parcial, por lo tanto con las operaciones brindadas una vez ingresada una palabra ésta no podrá borrarse. La principal ventaja de ésta implementación es la complejidad temporal del algoritmo de búsqueda que depende exclusivamente del tamaño de la palabra más grande en el conjunto y no del tamaño del elemento del que se busca testear pertenencia.

`conjTrie` **se representa con** `estr`

donde `estr` es `tupla`(*raíz*: `puntero(nodo)`, *longmax*: `nat`)

donde `nodo` es `tupla`(*hijos*: `arreglo_dimensionable(puntero(nodo))`, *definida*: `bool`)

Rep : estr $\longrightarrow$ bool

Rep(c) ≡ True $\Longleftrightarrow$ nodosMismoTamaño(hijos_raíz, tamaño_raíz) $\wedge_L$
noHayNodosSueltos(hijos_raíz) $\wedge_L$ ($\exists$ k : nat)(alturaAcotadaPorK(c.raíz → dato, k, 0)) $\wedge_L$ longMaxCorrecta

.

Reemplazos sintácticos:
hijos_raiz ≡ (c.raiz → dato).hijos
tamaño_raiz ≡ tamañoHijos(c.raiz → dato)
longMaxCorrecta ≡ ($\forall$ p : palabra)(definida?(p, e.raiz → dato) $\Rightarrow_L$ Long(p) $\leq$ res) $\wedge$ ($\exists$ p : palabra)(definida?(p, e.raiz → dato) $\wedge$ Long(p) = res)

Abs : estr e $\longrightarrow$ conj(palabra) {Rep(e)}

Abs(e) $=_{\text{obs}}$ c: conj(palabra) | ($\forall$ p : palabra)(p $\in$ c $\Leftrightarrow$ definida?(p, c.raiz $\rightarrow$ dato))

Algoritmos

---

iVacío(**in** n : `nat`) $\longrightarrow$ res : `estr`

```
1: res.raiz ← &new nuevoNodo(n)                                   ▷ O(n)
2: res.longmax ← 0                                                ▷ O(1)
```

Complejidad: $\mathcal{O}(n)$

---

iAgregar(**in/out** c : `estr`, **in** p : `palabra`)

```
1:
2: si c.longmax < Longitud(p) entonces                            ▷ O(1)
3:     c.longmax ← Longitud(p)                                    ▷ O(1)
4: end si
5: ptr_temp ← c.raiz                                              ▷ O(1)
6: i ← 0                                                          ▷ O(1)
7: mientras i < Longitud(p) hacer                                 ▷ O(K*Longitud(p))
8:     num_sig_letra ← ord^-1(p[i])                               ▷ O(1)
9:     si hijoNULL?(*ptr_temp,num_sig_letra) entonces             ▷ O(1)
10:        ptr_temp ← AgregarHijo(*ptr_temp, num_sig_letra)       ▷ O(K)
11:    else
12:        ptr_temp ←*ptr_temp.hijos[num_sig_letra]               ▷ O(1)
13:    end si
14:    i ← i + 1                                                  ▷ O(1)
15: end mientras
16: definir(*ptr_temp)                                            ▷ O(1)
```

Complejidad: $\mathcal{O}(K * Longitud(p))$, donde $K = Longitud((*c.raiz).hijos)$, el natural que se le pasó a Vacío para crear el conjunto.

---

iEsVacío?(**in** c : `estr`) $\longrightarrow$ res : `bool`

```
1: res ← ¬(*c.raiz).definida                                      ▷ O(1)
2: i ← 0                                                          ▷ O(1)
3: long ← Longitud((*c.raiz).hijos)                               ▷ O(1)
4: mientras res ∧_L i < long hacer                                ▷ O(K)
5:     res ← hijoNULL?(*c.raiz, i)                                ▷ O(1)
6:     i ← i + 1                                                  ▷ O(1)
7: end mientras
```

Complejidad: $\mathcal{O}(K)$

---

iPertenece?(**in** c : `estr`, **in** p : `palabra`) $\longrightarrow$ res : `bool`

```
1: res ← true                                                     ▷ O(1)
2: ptr_temp ← c.raiz                                              ▷ O(1)
3: i ← 0                                                          ▷ O(1)
4: si c.longmax < Longitud(p) entonces                            ▷ O(1)
5:     res ← false                                                ▷ O(1)
6: end si
7: mientras res ∧_L i < Longitud(p) hacer                         ▷ O(Longitud(p))
8:     num_sig_letra ← ord^-1(p[i])                               ▷ O(1)
9:     res ← ¬hijoNULL?(*ptr_temp, num_sig_letra)                 ▷ O(1)
10:    si res entonces                                            ▷ O(1)
11:        ptr_temp ← (*ptr_temp).hijos[num_sig_letra]            ▷ O(1)
12:    end si
13:    i ← i + 1                                                  ▷ O(1)
14: end mientras
15: si res entonces                                               ▷ O(1)
16:    res ← (*ptr_temp).definida                                 ▷ O(1)
17: end si
```

Complejidad: $\mathcal{O}(Longitud(p))$

iLargoDeClaveMax(**in** c : `estr`) $\longrightarrow$ res : `Nat`

1: $res \leftarrow c.longmax$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$

iAgregarPalabras(**in/out** c : `estr`, **in** ps : `conj(palabra)`

1: $it \leftarrow iterCreatIT(ps)$ ▷ $\mathcal{O}(1)$
2: **mientras** HaySiguiente(it) **hacer** ▷ $\mathcal{O}(K * (\sum p \in ps) Longitud(p))$
3: $Agregar(c, Siguiente(it))$ ▷ $\mathcal{O}(K * Longitud(Siguiente(it)))$
4: $it \leftarrow Avanzar(it)$ ▷ $\mathcal{O}(1)$
5: **end mientras**

Complejidad: $\mathcal{O}(K * (\sum p \in ps) Longitud(p))$

6.2. Funciones Auxiliares

nuevoNodo(**in** n : `nat`) $\longrightarrow$ res : `nodo`

1: $res.definida \leftarrow false$ ▷ $\mathcal{O}(1)$
2: $res.hijos \leftarrow Arreglo\_dimensionable(NULL)[n]$ ▷ $\mathcal{O}$(n)

Complejidad: $\mathcal{O}(n)$
Aliasing: devuelve un nodo por referencia modificable

hijoNULL?(**in** n : `nodo`, **in** k : `nat`) $\longrightarrow$ res : `bool`

1: $res \leftarrow n.hijos[k] = NULL$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$
Aliasing: no genera

AgregarHijo(**in/out** n : `nodo`, **in** k : `nat`) $\longrightarrow$ res : `puntero(nodo)`

1: $n.hijos[k] \leftarrow \&new\ nuevoNodo(Longitud(n.hijos))$ ▷ $\mathcal{O}$(longitud(n.hijos))
2: $res \leftarrow n.hijos[k]$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}$(longitud(n.hijos))
Aliasing: devuelve un puntero al nodo agregado como k-esimo hijo del nodo ingresado

Pre $\{n.hijos[k] = NULL \wedge_L n =_{\text{obs}} n_0\}$
Post$\{res =_{\text{obs}} n.hijos[k] \wedge_L Longitud(n.hijos) = Longitud(n_0.hijos) \wedge_L (\forall i : nat)((0 \leq i < Longitud(n.hijos) \wedge i \neq k) \Rightarrow_L n.hijos[i] =_{\text{obs}} n.hijos[i])\ \}$

definir(**in/out** n : `nodo`)

1: $n.definida \leftarrow True$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1)$
Aliasing: no genera