Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Integrante	LU	Correo electrónico
Guberman, Diego Andrés	469/17	diego98g@hotmail.com
Ramis Folberg, Ezequiel Leonel	881/21	ezequielramis.hello@gmail.com
Sabetay, Kevin Damian	476/16	kevin.sabetay96@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Preámbulo

Antes de presentar los módulos, definimos las siguientes variables para las complejidades temporales:

- N — tamaño del tablero.
- K — cantidad de jugadores.
- $|\Sigma|$ — cantidad de letras en el alfabeto.
- F — cantidad de fichas por jugador.
- $L_{\texttt{máx}}$ — longitud de la palabra legítima más larga definida por la variante del juego de la que se trate.

2. Módulo Juego

2.1. Interfaz

se explica con: JUEGO

géneros: `juego`

usa: BOOL, NAT, COLA, LETRA, OCURRENCIA, VARIANTE

operaciones:

NUEVOJUEGO(**in** k : `nat`, **in** v : `variante`, **in** r : `pila(letra)`) $\rightarrow$ res : `juego`
Pre $\equiv$ $\{tamaño(r) \geq tamañoTablero(v) * tamañoTablero(v) + k * \#fichas(v) \wedge k > 0\}$
Post $\equiv$ $\{res =_{\text{obs}} nuevoJuego(k, v, r)\}$
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: Dada una cantidad de jugadores, una variante de juego y un repositorio de fichas, se inicia un nuevo juego con el tablero vacío.
Aliasing: ??

JUGADAVÁLIDA?(**in** j : `juego`, **in** o : `occurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} jugadaVálida?(j, o)\}$
Complejidad: $O(L^2_{\texttt{máx}})$
Descripción: Determina si una jugada es válida.
Aliasing: ??

UBICAR(**in/out** j : `juego`, **in** o : `occurrencia`)
Pre $\equiv$ $\{jugadaVálida(j, o) \wedge j =_{\text{obs}} J_0\}$
Post $\equiv$ $\{j =_{\text{obs}} ubicar(J_0, o)\}$
Complejidad: $O(m)$, donde m es el número de fichas que se ubican.
Descripción: Ubica un conjunto de fichas en el tablero.
Aliasing: ??

VARIANTE(**in** j : `juego`) $\rightarrow$ res : `variante`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} variante(j)\}$
Complejidad: $O(1)$
Descripción: Obtiene información sobre la variante del juego.
Aliasing: ??

TURNO(**in** j : `juego`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} turno(j)\}$

Complejidad: $O(1)$
Descripción: Obtiene al jugador del turno actual.
Aliasing: ??

PUNTAJE(**in** j : `juego`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} puntaje(j, i)\}$
Complejidad: $O(1 + m \cdot L_{\texttt{máx}})$, donde m es la cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación.
Descripción: Obtiene el puntaje de un jugador.
Aliasing: ??

ENTABLERO?(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} enTablero?(tablero(J), i, j)\}$
Complejidad: $O(1)$
Descripción: Determina si una coordenada (i, j) está en el rango del tablero.
Aliasing: ??

HAYLETRA?(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{enTablero?(tablero(J), i, j)\}$
Post $\equiv \{res =_{\text{obs}} hayLetra?(tablero(J), i, j)\}$
Complejidad: $O(1)$
Descripción: Determina si una celda del tablero dada una coordenada (i, j) está ocupada por una letra.
Aliasing: ??

LETRA(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `letra`
Pre $\equiv \{enTablero?(tablero(J), i, j) \wedge_{\text{L}} hayLetra?(tablero(J), i, j)\}$
Post $\equiv \{res =_{\text{obs}} letra(tablero(J), i, j)\}$
Complejidad: $O(1)$
Descripción: Obtiene el contenido de una celda del tablero dada una coordenada (i, j).
Aliasing: ??

#LETRATIENEJUGADOR(**in** j : `juego`, **in** x : `letra`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} \#(x, fichas(j, i))\}$
Complejidad: $O(1)$
Descripción: Dada una cierta letra x del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra.
Aliasing: ??

2.2. Implementación

Representación

`juego` **se representa con** `juego_estr`

donde `juego_estr` es `tupla`(

tablero: `tab`
, *jugadores*: `array_dimensionable(jugador)`
, *tiempo*: `nat`
, *repositorio*: `pila(letra)`
, *variante*: `variante`
)

y `jugador` es `tupla`(

puntaje: `nat`
, *historial*: `lista(tupla(`*ocurrencia*: `ocurrencia`, *tiempo*: `nat))`
, *jugadasSinCalcularPuntaje*: `nat`
, *cantFichasPorLetra*: `array_dimensionable(nat)`
)

Invariante de Representación

Rep : juego_estr $\longrightarrow$ bool

$$
\begin{aligned}
\text{Rep}(e) \equiv\ & tamaño(e.tablero) = tamañoTablero(e.variante) \wedge \\
& (\forall i : \texttt{nat})(i < tam(e.jugadores) \Rightarrow_{\text{L}} (\\
& \quad tam(e.jugadores[i].cantFichasPorLetra) = \text{DOM}() \wedge_{\text{L}} \\
& \quad \textstyle\sum_{f<\text{DOM}()} e.jugadores[i].cantFichasPorLetra[f] = \#fichas(e.variante) \wedge \\
& \quad e.jugadores[i].jugadasSinCalcularPuntaje \leq tam(e.jugadores[i].historial) \wedge_{\text{L}} \\
& \quad tam(e.jugadores[i].historial) = \lceil e.tiempo/tam(e.jugadores) \rceil \wedge \\
& \quad (\forall h : \texttt{nat})(h < long(e.jugadores[i].historial) \Rightarrow_{\text{L}} \\
& \quad\quad e.jugadores[i].historial[h].tiempo = h * tam(e.jugadores) + i \wedge \\
& \quad\quad (\forall p, q : \texttt{nat})(\forall l, l' : \texttt{letra})(\\
& \quad\quad\quad \{\langle p, q, l\rangle, \langle p, q, l'\rangle\} \subseteq e.jugadores[i].historial[h].ocurrencia \ \Rightarrow\ l = l' \\
& \quad\quad) \\
& \quad) \\
&)) \wedge_{\text{L}} \\
& ocurrenciasVálidas?(nuevoTablero(tamaño(e.tablero)), historiales(e.jugadores, 0)) \wedge_{\text{L}} \\
& e.tablero =_{\text{obs}} ponerOcurrencias(nuevoTablero(tamaño(e.tablero)), historiales(e)) \wedge_{\text{L}} \\
& (\forall i : \texttt{nat})(i < tam(e.jugadores) \Rightarrow_{\text{L}} \\
& \quad e.jugadores[i].puntaje = \textstyle\sum_{k<tam(e.jugadores[i].historial)-e.jugadores[i].jugadasSinCalcularPuntaje} \\
& \qquad\qquad\qquad\qquad puntajeDeOcurrenciaEnTiempo(e, i, k) \\
&)
\end{aligned}
$$

donde

$historiales$: `juego_estr` $\longrightarrow$ `multiconj(ocurrencia)`
$historiales(e') \equiv historialesHastaTiempo(e'.jugadores, 0, e.tiempo)$

$historialesHastaTiempo$: `ad(jugador)` $\times$ `nat` $\times$ `nat`
$\longrightarrow$ `multiconj(ocurrencia)`
$historialesHastaTiempo(js, k, t) \equiv$

if $k \geq tam(js)$
then $\emptyset$
else $historialHastaTiempo(js, k, t)$
$\cup\ historialesHastaTiempo(js, k + 1, t)$

fi

$historialHastaTiempo$: `ad(jugador)` $\times$ `nat` $\times$ `nat`
$\longrightarrow$ `multiconj(ocurrencia)`
$historialHastaTiempo(js, k, t) \equiv historialHastaTiempo'(js[k].historial, t)$

$historialHastaTiempo'$: `lista(tupla(ocurrencia,nat))` $\times$ `nat`
$\longrightarrow$ `multiconj(ocurrencia)`
$historialHastaTiempo'(ls, t) \equiv$

if $vacía?(ls)$

then $\emptyset$
else $historialHastaTiempo'(fin(ls), t) \cup$

if $\pi_2(prim(ls)) < t$

then $prim(ls)$
else $\emptyset$

fi

fi

$ocurrenciasVálidas?$: `tab` $\times$ `multiconj(ocurrencia))` $\longrightarrow$ `bool`
$ocurrenciasVálidas?(t, os) \equiv$

if $vacía?(os)$

then $true$
else $celdasLibres?(t, dameUno(os)) \wedge_{\text{L}}$
$ocurrenciasVálidas?(ponerLetras(t, dameUno(os)), sinUno(os))$

fi

$ponerOcurrencias$: `tab` $\times$ `multiconj(ocurrencia)` $\longrightarrow$ `tab`
$ponerOcurrencias(t, os) \equiv$

if $vacía?(os)$

then t
else $ponerOcurrencias(ponerLetras(t, dameUno(os)), sinUno(os))$

fi

$puntajeDeOcurrenciaEnTiempo$: `estr_juego` $\times$ `nat` $\times$ `nat` $\longrightarrow$ `nat`
$puntajeDeOcurrenciaEnTiempo(e, i, k) \equiv$

$puntajePalabrasEstr(e.variante, t',$
$palabrasUbicadas(ocurrenciasDePalabras(t'), e.jugadores[i].historial[k].ocurrencia))$

donde

$tiempo \equiv e.jugadores[i].historial[k].tiempo$
$t' \equiv ponerOcurrencias(nuevoTablero(tamaño(e.tablero)),$

if $tiempo = 0?$

then $\emptyset$
else $historialesHastaTiempo(e.jugadores, 0, tiempo - 1)$

fi

$\cup\ historialHastaTiempo(e.jugadores, i, tiempo)$
$\cup\ \{e.jugadores[i].historial[k].ocurrencia\})$

$puntajePalabrasEstr$: `variante` $\times$ `tab` $\times$ `conj(ocurrencia)` $\longrightarrow$ `nat`
$puntajePalabrasEstr(v, t, os) \equiv$

if $vacío?(os)$

then 0
else $puntajePalabraEstr(v, t, dameUno(os))$
$+ puntajePalabras(v, t, sinUno(os))$

fi

$puntajePalabraEstr$: `variante` $\times$ `tab` $\times$ `ocurrencia` $\longrightarrow$ `nat`
$puntajePalabraEstr(v, t, o) \equiv$

if $vacía?(o)$

then 0
else $puntajeLetra(v, \pi_3(dameUno(o)))$
$+ puntajePalabra(v, t, sinUno(o))$

fi

Función de Abstracción

Abs : juego_estr e $\longrightarrow$ juego {Rep(e)}

Abs(e) $=_{obs}$ J: juego | $e.variante =_{obs} variante(J) \wedge$
$e.repositorio =_{obs} repositorio(J) \wedge$
$e.tiempo \equiv turno(J)\ (mod\ \#jugadores(J)) \wedge$
$e.tablero =_{obs} tablero(J) \wedge$
$(tam(e.jugadores) =_{obs} \#jugadores(J) \wedge_{L}$
$(\forall i : \texttt{nat})(i < tam(e.jugadores) \Rightarrow_{L} ($
$e.jugadores[i].puntaje =_{obs} puntaje(J, i) \wedge$
$(\forall l : \texttt{letra})(e.cantFichasPorLetra[\text{ORD}(l)] = \#(l, fichas(J, i)))$
$))$

Algoritmos

```
iNuevoJuego(in k : nat, in v : variante, in r : cola(letra)) ⟶ juego_estr
1:  res.variante ← v
2:  res.repositorio ← r
3:  res.tiempo ← 0
4:  res.tablero ← nuevoTablero(tamañoTablero(v))                    ▷ O(N^2)
5:  res.jugadores ← CrearArreglo(k)
6:  for jugador ∈ res.jugadores do                                  ▷ O(K)
7:      jugador.puntaje ← 0
8:      jugador.historial ← Vacía()
9:      jugador.jugadasSinCalcularPuntaje ← 0
10:     // Por cada jugador le damos su primer mazo de fichas del repositorio
11:     jugador.cantFichasPorLetra ← CrearArreglo(dom())
12:     for cant ∈ jugador.cantFichasPorLetra do                    ▷ O(|Σ|)
13:         cant ← 0
14:     for 1 ... fichasPorJugador(v) do                            ▷ O(F)
15:         ficha ← Desapilar(res.repositorio)
16:         jugador.cantFichasPorLetra[ord(ficha)] + +
17: return res
```

Justificación de complejidad:

$$\begin{aligned} O(N^2) + O(K) * (O(|\Sigma|) + O(F)) &= O(N^2) + O(K) * O(|\Sigma| + F) \\ &= O(N^2) + O(K * (|\Sigma| + F)) \\ &= O(N^2) + O(|\Sigma|K + FK) \\ &= O(N^2 + |\Sigma|K + FK) \end{aligned}$$

```
iJugadaVálida?(in j : estr_juego, in o : ocurrencia) ⟶ bool
1:  if Cardinal(o) > longPalabraMásLarga(j.variante) then           ▷ Con este if evitamos acotar por m
2:      return false
3:  for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do        ▷ O(L_máx)
4:      ficha ← Siguiente(oIt)
5:      if ¬enTablero?(j.tablero, π1(ficha), π2(ficha)) ∨L hayLetra?(j.tablero, π1(ficha), π2(ficha)) then
6:          return false
7:  if haySuperpuestas?(o) ∨ ¬(esHorizontal?(o) ∨ esVertical?(o)) then   ▷ O(L_máx^2)
8:      return false
9:  // Ponemos las fichas de la ocurrencia para validar
10: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do        ▷ O(L_máx)
11:     ficha ← Siguiente(oIt)
12:     ponerLetra(e.tablero, ficha)                                ▷ O(1)
13:     ??
14: // Sacamos las fichas de la ocurrencia para no modificar el tablero
15: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do        ▷ O(L_máx)
16:     ficha ← Siguiente(oIt)
17:     sacarLetra(e.tablero, π1(ficha), π2(ficha))                 ▷ O(1)
18: return true
```

Justificación de complejidad:

iUbicar(**in/out** j : estr_juego, **in** o : ocurrencia)

```
1: jugador ← j.jugadores[turno(j)]
2: j.tiempo++
3: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                ▷ O(m)
4:     ficha ← Siguiente(oIt)
5:     ponerLetra(j.tablero, π1(ficha), π2(ficha), π3(ficha), π4(ficha))   ▷ O(1)
6:     jugador.cantFichasPorLetra[ord(π3(ficha))] − −
7:     jugador.cantFichasPorLetra[ord(Desapilar(j.repositorio))] + +       ▷ O(1)
```

iVariante(**in** j : estr_juego) $\longrightarrow$ variante

```
1: return j.variante
```

iTurno(**in** j : estr_juego) $\longrightarrow$ nat

```
1: return j.tiempo % tam(j.jugadores)
```

iPuntaje(**in** j : estr_juego, **in** i : nat) $\longrightarrow$ nat

```
1: ??
```

iEnTablero?(**in** j : estr_juego, **in** i : nat, **in** j : nat) $\longrightarrow$ bool

```
1: return enTablero?(j.tablero, i, j)
```

iHayLetra?(**in** j : estr_juego, **in** i : nat, **in** j : nat) $\longrightarrow$ bool

```
1: return hayLetra?(j.tablero, i, j)
```

iLetra(**in** j : estr_juego, **in** i : nat, **in** j : nat) $\longrightarrow$ letra

```
1: return letra(j.tablero, i, j)
```

i#letraTieneJugador(**in** j : estr_juego, **in** l : letra, **in** i : nat) $\longrightarrow$ nat

```
1: return j.jugadores[i].cantFichasPorLetra[ord(l)]
```

2.3. Servicios usados

3. Módulo Servidor

3.1. Interfaz

se explica con: SERVIDOR

géneros: `servidor`

usa: NAT, JUEGO, OCURRENCIA, VARIANTE

operaciones:

NUEVOSERVIDOR(**in** k : `nat`, **in** v : `variante`) $\rightarrow$ res : `servidor`
Pre $\equiv$ {true}
Post $\equiv$ {$\exists r$: `cola(letra)` | $res =_{\text{obs}} nuevoServidor(k, v, r)$}
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: Dada una cantidad de jugadores y una variante de juego, se inicia un nuevo servidor y una nueva partida de juego.
Aliasing: ??

CONECTAR(**in/out** s : `servidor`)
Pre $\equiv$ {$\neg empezó?(s) \land s =_{\text{obs}} S_0$}
Post $\equiv$ {$s =_{\text{obs}} conectarCliente(S_0)$}
Complejidad: $O(1)$
Descripción: Conecta un cliente a un servidor.
Aliasing: ??

CONSULTAR(**in/out** s : `servidor`, **in** cid : `nat`)
Pre $\equiv$ {$cid \leq \#conectados(s) \land s =_{\text{obs}} S_0$}
Post $\equiv$ {$s =_{\text{obs}} consultar(S_0, cid)$}
Complejidad: $O(n)$, donde n es la cantidad de mensajes en la cola de dicho cliente.
Descripción: Consulta la cola de notificaciones de un cliente (lo cual vacía dicha cola).
Aliasing: ??

RECIBIR(**in/out** s : `servidor`, **in** cid : `nat`, **in** o : `ocurrencia`)
Pre $\equiv$ {$cid \leq \#conectados(s) \land s =_{\text{obs}} S_0$}
Post $\equiv$ {$s =_{\text{obs}} recibirMensaje(S_0, cid, o)$}
Complejidad: ??
Descripción: Recibe un mensaje de un cliente.
Aliasing: ??

CLIENTESESPERADOS(**in** s : `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} \#esperados(s)$}
Complejidad: $O(1)$
Descripción: Obtiene el número de clientes esperados.
Aliasing: ??

CLIENTESCONECTADOS(**in** s : `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} \#conectados(s)$}
Complejidad: $O(1)$
Descripción: Obtiene el número de clientes conectados.
Aliasing: ??

PARTIDA(**in** s : `servidor`) $\rightarrow$ res : `juego`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} juego(s)$}
Complejidad: $O(1)$
Descripción: Obtiene el juego que se está jugando en el servidor.
Aliasing: ??

3.2. Implementación

Representación

`servidor` **se representa con** `servidor_estr`

donde `servidor_estr` es `tupla`(

juego: `juego`
, *jugadoresConectados*: `nat`
, *jugadoresEsperados*: `nat`
, *notificaciones*: `array_dimensionable(cola(notif))`

)

Invariante de Representación

Función de Abstracción

Algoritmos

3.3. Servicios usados

4. Módulos auxiliares

4.1. Módulo Letra

Se asume una implementación acorde[1] al módulo de género `letra` con las siguientes operaciones en la interfaz (todas con órden de complejidad $O(1)$):

- DOM : $\rightarrow$ `nat` — Tamaño del dominio del tipo `letra`. Corresponde con la variable A de su especificación.
- ORD : `letra` $\rightarrow$ `nat` — Dada una letra, devuelve su correspondiente índice.
- ORD^{-1} : `nat` n $\rightarrow$ `letra` $\{n < A\}$ — Dado un índice, devuelve su correspondiente letra.

4.2. Módulo Variante (Trie)

4.2.1. Interfaz

se explica con: VARIANTE

géneros: `variante`

usa: ??

operaciones:

NUEVAVARIANTE(

in n : `nat`,
in f : `nat`,
in *puntajes* : `dicc(letra, nat)`,
in *legítimas* : `conj(secu(letra))`

) $\rightarrow$ *res* : `variante`
Pre $\equiv \{n > 0 \land f > 0\}$
Post $\equiv \{res =_{\text{obs}} nuevaVariante(n, f, puntajes, legítimas)\}$
Complejidad: $O(\#legítimas \cdot L_{\text{máx}})$
Descripción: Genera una variante de juego.
Aliasing: ??

TAMAÑOTABLERO(**in** v : `variante`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} tamañoTablero(v)\}$
Complejidad: $O(1)$
Descripción: Devuelve el tamaño del tablero.
Aliasing: ??

FICHASPORJUGADOR(**in** v : `variante`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \#fichas(v)\}$
Complejidad: $O(1)$
Descripción: Devuelve la cantidad de fichas que debe de tener cada jugador.
Aliasing: ??

PUNTAJELETRA(**in** v : `variante`, **in** l : `letra`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} puntajeLetra(v, l)\}$
Complejidad: $O(1)$
Descripción: Devuelve el puntaje de una letra.

[1] Una buena opción es usar un Enumerado.

Aliasing: ??

PALABRALEGÍTIMA?(**in** v : `variante`, **in** l : `secu(letra)`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} palabraLegítima(v, l)\}$
Complejidad: $O(L_{\text{máx}})$
Descripción: Determina si una palabra es legítima dentro de la variante de juego.
Aliasing: ??

LONGPALABRAMÁSLARGA(**in** v : `variante`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {

$$(\exists p : \texttt{secu(letra)})(res =_{\text{obs}} long(p) \land palabraLegítima?(v, p) \land$$
$$(\forall p_2 : \texttt{secu(letra)})(palabraLegítima?(v, p_2) \Rightarrow res \geq long(p_2)))$$

}
Complejidad: $O(1)$
Descripción: Obtiene la longitud de la palabra legítima más larga de la variante.
Aliasing: ??

4.2.2. Implementación

4.3. Módulo Tablero

4.3.1. Interfaz

se explica con: TABLERO

géneros: `tab`

usa: ??

operaciones:

NUEVOTABLERO(**in** n : `nat`) $\rightarrow$ res : `tab`
Pre $\equiv$ $\{n > 0\}$
Post $\equiv$ $\{res =_{\text{obs}} nuevoTablero(n)\}$
Complejidad: $O(N^2)$
Descripción: Genera un tablero de tamaño n.
Aliasing: ??

TAMAÑO(**in** t : `tab`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} tamaño(t)\}$
Complejidad: $O(1)$
Descripción: Devuelve tamaño del tablero.
Aliasing: ??

PONERLETRA(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`, **in** l : `letra`, **in** tm : `nat`) $\rightarrow$ res : `tab`
Pre $\equiv$ $\{enTablero(t, i, j) \land_{\text{L}} \neg hayLetra?(t, i, j)\}$
Post $\equiv$ $\{res =_{\text{obs}} ponerLetra(t, i, j, l)\}$
Complejidad: $O(1)$
Descripción: Pone una letra en la coordenada (i, j).
Aliasing: ??

SACARLETRA(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `tab`
Pre $\equiv$ $\{enTablero(t,i,j) \wedge_{\text{L}} hayLetra?(t,i,j)\}$
Post $\equiv$ $\{res$ es igual al tablero antes de haber puesto una letra en la coordenada $(i,j)\}$
Complejidad: $O(1)$
Descripción: Saca una letra en la coordenada (i,j).
Aliasing: ??

ENTABLERO?(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{\text{true}\}$
Post $\equiv$ $\{res =_{\text{obs}} enTablero?(t,i,j)\}$
Complejidad: $O(1)$
Descripción: Determina si una coordenada (i,j) está en el rango del tablero.
Aliasing: ??

HAYLETRA?(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{enTablero?(t,i,j)\}$
Post $\equiv$ $\{res =_{\text{obs}} hayLetra?(t,i,j)\}$
Complejidad: $O(1)$
Descripción: Determina si una celda del tablero dada una coordenada (i,j) está ocupada por una letra.
Aliasing: ??

LETRA(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `letra`
Pre $\equiv$ $\{enTablero?(t,i,j) \wedge_{\text{L}} hayLetra?(t,i,j)\}$
Post $\equiv$ $\{res =_{\text{obs}} letra(t,i,j)\}$
Complejidad: $O(1)$
Descripción: Obtiene la letra de una ficha del tablero dada una coordenada (i,j).
Aliasing: ??

TIEMPO(**in** t : `tab`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv$ $\{enTablero?(t,i,j) \wedge_{\text{L}} hayLetra?(t,i,j)\}$
Post $\equiv$ $\{res =_{\text{obs}} letra(t,i,j)\}$
Complejidad: $O(1)$
Descripción: Obtiene el momento en que una ficha del tablero fue puesta dada una coordenada (i,j).
Aliasing: ??

4.3.2. Implementación

Representación

`tab` **se representa con** `tab_estr`

donde `tab_estr` es `array_dimensionable(array_dimensionable(puntero(tupla(letra,nat))))`

Invariante de Representación

Rep : tab_estr $\longrightarrow$ bool
Rep(e) $\equiv$ $(\forall i : \texttt{nat})(i < tam(tab\_estr) \Rightarrow_{\text{L}} tam(tab\_estr[i]) = tam(tab\_estr))$

Función de Abstracción

Abs : tab_estr e $\longrightarrow$ tab $\{\text{Rep}(e)\}$

$$\begin{aligned}\text{Abs}(e) =_{\text{obs}} \text{T: tab} \mid\ & tam(e) =_{\text{obs}} tamaño(T) \wedge_{\text{L}} \\ & (\forall i, j : \texttt{nat})((enTablero?(T, i, j) \wedge_{\text{L}} hayLetra?(T, i, j)) \Rightarrow_{\text{L}} \\ & (e[i][j] \neq NULL \wedge_{\text{L}} letra(T, i, j) =_{\text{obs}} \pi_1(*e[i][j])))\end{aligned}$$

Algoritmos

iNuevoTablero(**in** n : nat) $\longrightarrow$ tab_estr

```
1: filas ← CrearArreglo(n)
2: for columnas ∈ filas do                                   ▷ O(N)
3:     columnas ← CrearArreglo(n)
4:     for celda ∈ columnas do                               ▷ O(N)
5:         celda ← NULL
6: return res
```

iTamaño(**in** t : tab_estr) $\longrightarrow$ nat

```
1: return tam(t)
```

iPonerLetra(**in/out** t : tab_estr, **in** i : nat, **in** j : nat, **in** l : letra, **in** tm : nat)

```
1: t[i][j] ← &⟨l, tm⟩
```

iSacarLetra(**in/out** t : tab_estr, **in** i : nat, **in** j : nat)

```
1: t[i][j] ← NULL
```

iEnTablero(**in** t : tab_estr, **in** i : nat, **in** j : nat) $\longrightarrow$ bool

```
1: return i < tam(t) ∧ j < tam(t)
```

iHayLetra(**in** t : tab_estr, **in** i : nat, **in** j : nat) $\longrightarrow$ bool

```
1: return t[i][j] ≠ NULL
```

iLetra(**in** t : tab_estr, **in** i : nat, **in** j : nat) $\longrightarrow$ letra

```
1: return π1(*t[i][j])
```

iTiempo(**in** t : tab_estr, **in** i : nat, **in** j : nat) $\longrightarrow$ nat

```
1: return π2(*t[i][j])
```

4.4. Módulo Ocurrencia

Es renombre de `conj(tupla(nat,nat,letra))` con las siguientes operaciones auxiliares.

operaciones:

esHorizontal?(**in** o : ocurrencia) $\rightarrow$ res : bool
Pre $\equiv$ {true}

Post $\equiv \{res =_{\text{obs}} true \iff (\forall f, f' : \texttt{tupla(nat,nat,letra)})(f, f' \in o \wedge f \neq f' \Rightarrow \pi_2(f) = \pi_2(f'))\}$
Complejidad: $O(\#o^2)$
Descripción: Determina si una ocurrencia está alineada horizontalmente.
Aliasing: ??

esVertical?(**in** o : `ocurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} true \iff (\forall f, f' : \texttt{tupla(nat,nat,letra)})(f, f' \in o \wedge f \neq f' \Rightarrow \pi_1(f) = \pi_1(f'))\}$
Complejidad: $O(\#o^2)$
Descripción: Determina si una ocurrencia está alineada verticalmente.
Aliasing: ??

haySuperpuestas?(**in** o : `ocurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} false \iff (\forall p, q : \texttt{nat})(\forall l, l' : \texttt{letra})(\{\langle p, q, l\rangle, \langle p, q, l'\rangle\} \subseteq o \Rightarrow l = l')\}$
Complejidad: $O(\#o^2)$
Descripción: Determina si existen fichas distintas en una misma coordenada.
Aliasing: ??

Algoritmos

iEsHorizontal?(**in** o : `ocurrencia`) $\longrightarrow$ `bool`

```
1: colns ← Vacío()                                              ▷ Conjunto Lineal
2: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do     ▷ O(#o)
3:     ficha ← Siguiente(oIt)
4:     Agregar(colns, π2(ficha)))                               ▷ O(#o)
5: return Cardinal(colns)=1
```

iEsVertical?(**in** o : `ocurrencia`) $\longrightarrow$ `bool`

```
1: filas ← Vacío()                                              ▷ Conjunto Lineal
2: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do     ▷ O(#o)
3:     ficha ← Siguiente(oIt)
4:     Agregar(filas, π1(ficha)))                               ▷ O(#o)
5: return Cardinal(filas)=1
```

iHaySuperpuestas?(**in** o : `ocurrencia`) $\longrightarrow$ `bool`

```
1: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do     ▷ O(#o)
2:     ficha ← Siguiente(oIt)
3:     for oItSig ← Avanzar(copy(oIt)); HaySiguiente(oItSig); Avanzar(oItSig) do   ▷ O(#o)
4:         ficha′ ← Siguiente(oItSig)
5:         if π1(ficha) = π1(fichaSig) ∧ π2(ficha) = π2(fichaSig) then
6:             return true
7: return false
```

4.5. Módulo Notificación

Asumimos que existe el tipo notif