

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

La cosa se pone compleja

Grupo d-mifflin

Integrante	LU	Correo electrónico
Guglielmino, Adrian Roberto	39/18	adrianguglielmino@gmail.com
Hajek, David	52/32	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

2. Módulo Variante

Interfaz

se explica con: VARIANTE

géneros: variante.

Operaciones básicas de Variante

CREARVARIANTE(in tamaño : nat, in cant_fichas : nat, in valor_letras : dicc(letra × nat), in palabras_validas : conj(Lista(letra))) → res : variante

Pre ≡ {n > 0 ∧ cant_fichas > 0}

Post ≡ {res =_{obs} nuevaVariante(tamaño, cant_fichas, valor_letras, palabras_validas)}

Complejidad: $\mathcal{O}(C * L_{max})$, donde C = cantidad de palabras

Descripción: crea un nueva variante

Aliasing: pasamos valor_letras y palabras_validas por referencia no modificable

TAMAÑODELTABLERO(in v : variante) → res : nat

Pre ≡ {true}

Post ≡ {res =_{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 =_{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 =_{obs} puntajeLetra(l, v)}

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

Descripción: devuelve el puntaje asociado a una determinada letra l en la variante del juego

Aliasing: no aplica

ESPALABRAPERMITIDA(in v : variante, in palabra : Lista(letra)) → res : bool

Pre ≡ {true}

Post ≡ {res =_{obs} esPalabraLegitima?(v, palabra)}

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

Descripción: determina si una palabra es válida para la variente del juego

Aliasing: no aplica

PALABRASPERMITIDAS(in v : variante, in palabras : Cola(Lista(letra))) → res : DiccTrie(Lista(letra), Nat)

Pre ≡ {true}

Post ≡ {res =_{obs} (∀palabra : Lista(letra)(esta?(palabras, palabra) ∧_L palabraPermitida(variante, palabra)))}

Complejidad: $\mathcal{O}(Tamao(palabras) * L_{max}) = \mathcal{O}(L_{max} * L_{max})$

Descripción: determina si unas palabras son válidaa para la variente del juego

Aliasing: toma por referencia a la variante y por copia a la cola de palabras

Representación

variante se representa con var

donde var es tupla(tamaño: nat, cant_fichas: nat, valor_letras: array[nat], palabras_puntos: diccTrie(palabras: Lista(letra), t))

Rep : var → bool

Rep(v) ≡ true ⇔ (v.tamaño > 0) ∧ (v.cant_fichas > 0)

Abs : var v → variante

{Rep(v)}

3. Módulo Tablero

Interfaz

se explica con: TABLERO

géneros: tablero

Operaciones básicas de Tablero

TAMAÑO(**in** $t : \text{tablero}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{tamaño}(t)\}$

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

Descripción: devuelve el tamaño de un tablero

Aliasing: no aplica

HAYUNALETRAENPOS?(**in** $t : \text{tablero}$, **in** $fila : \text{nat}$, **in** $columna : \text{nat}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{enTablero?(t, fila, columna)\}$

Post $\equiv \{res =_{\text{obs}} hayLetra?(t, fila, columna)\}$

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

Descripción: determina si un casillero indicado de un tablero está ocupado

Aliasing: no aplica

LETRAENPOS(**in** $t : \text{tablero}$, **in** $fila : \text{nat}$, **in** $columna : \text{nat}$) $\rightarrow res : \text{letra}$

Pre $\equiv \{enTablero?(t, fila, columna) \wedge_L hayLetra?(t, fila, columna)\}$

Post $\equiv \{res =_{\text{obs}} letra(t, fila, columna)\}$

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

Descripción: devuelve la letra de un casillero del tablero

Aliasing: no aplica

CREARTABLERO(**in** $n : \text{nat}$) $\rightarrow res : \text{tablero}$

Pre $\equiv \{n > 0\}$

Post $\equiv \{res =_{\text{obs}} nuevoTablero(n)\}$

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

Descripción: crea un tablero de tamaño n

Aliasing: no aplica

PONERLETRA(**in/out** $t : \text{tablero}$, **in** $fila : \text{nat}$, **in** $columna : \text{nat}$, **in** $l : \text{letra}$)

Pre $\equiv \{t = t_0 \wedge (enTablero?(t, fila, columna) \wedge_L \neg hayLetra?(t, fila, columna))\}$

Post $\equiv \{res =_{\text{obs}} ponerLetra(t_0, fila, columna, l)\}$

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

Descripción: devuelve el tablero con la letra l colocada en el casillero indicado

Aliasing: no aplica

ENTABLERO(**in** $t : \text{tablero}$, **in** $i : \text{nat}$, **in** $j : \text{nat}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} enTablero?(t, i, j)\}$

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

Descripción: indica si el casillero (i,j) está en el tablero

Aliasing: no aplica

ESTALIBRE(**in** $t : \text{tablero}$, **in** $i : \text{nat}$, **in** $j : \text{nat}$) $\rightarrow res : \text{bool}$

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** $t : \text{tablero}$, **in** $i : \text{nat}$, **in** $j : \text{nat}$) $\rightarrow res : \text{bool}$

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** $t : \text{tablero}$, **in** $ocurrencia : \text{conjunto}(\text{tupla}(\text{nat}, \text{nat}, \text{letra}))$)

Pre $\equiv \{t = t_0 \wedge celdasLibres?(t, ocurrencia) \wedge ((\forall i, j : \text{nat})(\forall l, l' : \text{letra})((\langle i, j, l \rangle \in ocurrencia \Rightarrow \langle i, j, l' \rangle \in ocurrencia) \Rightarrow l = l'))\}$

Post $\equiv \{res =_{\text{obs}} ponerLetras(t_0, ocurrencia)\}$

Complejidad: $\mathcal{O}(\text{tamaño}(ocurrencia))$

Descripción: ubica cada elemento de la ocurrencia en el tablero

Aliasing: Pasamos la ocurrencia por referencia no modificable

SONCELDASLIBRES?(**in** $t : \text{tablero}$, **in** $ocurrencia : \text{conjunto}(\text{tupla}(\text{nat}, \text{nat}, \text{letra}))$) $\rightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} celdasLibres?(t, ocurrencia)\}$

Complejidad: $\mathcal{O}(\text{tamaño}(ocurrencia))$

Descripción: chequea que las celdas de la ocurrencia esten libres
Aliasing: no aplica

Representación

tablero se representa con `tab`

donde `tab` es `arreglo(arreglo(tupla < ocupado: bool , letr: letra >))`

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

$\text{Rep}(t) \equiv \text{true} \iff (\forall i, j: \text{nat}) (0 \leq i, j < \text{longitud}(t) \Rightarrow_L (\text{longitud}(t) = \text{longitud}(t[i]) \wedge \text{longitud}(t[i]) = \text{longitud}(t[i][j])))$

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

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

Algoritmos

I TAMAÑO(in <i>t</i> : tab) \longrightarrow <i>res</i> : nat		
1: <code>rest.tamao</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		
<hr/>		
I HAYUNALETRAENPOS?(in <i>t</i> : tab , in <i>fila</i> : nat , in <i>columna</i> : nat) \longrightarrow <i>res</i> : bool		
1: <code>res(t.casilleros[fila][columna]).ocupado</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		
<hr/>		
I LETRAENPOS(in <i>t</i> : tab , in <i>fila</i> : nat , in <i>columna</i> : nat) \longrightarrow <i>res</i> : letra		
1: <code>res(t.casilleros[fila][columna]).letr</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		
<hr/>		
I CREARTABLERO(in <i>tam</i> : nat) \longrightarrow <i>res</i> : tab		
1: <code>t.tamañotam</code>		$\triangleright \mathcal{O}(1)$
2: <code>t.casillerosarreglo(arreglo(tupla<false, >)[n])[n]</code>		$\triangleright \mathcal{O}(n^2)$
3: <code>rest</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(N^2)$		
<hr/>		
I PONERLETRA(in/out <i>t</i> : tab , in <i>fila</i> : nat , in <i>columna</i> : nat , in <i>l</i> : letra)		
1: <code>t.casilleros[fila][columna].casillero.letrl</code>		$\triangleright \mathcal{O}(1)$
2: <code>t.casilleros[fila][columna].casillero.ocupadatru</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		
<hr/>		
I ENTABLERO(in <i>t</i> : tab , in <i>fila</i> : nat , in <i>columna</i> : nat) \longrightarrow <i>res</i> : bool		
1: <code>res(fila <t.tamaño) ∧(columna < t.tamao)</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		
<hr/>		
I ESTALIBRE(in <i>t</i> : tab , in <i>fila</i> : nat , in <i>columna</i> : nat) \longrightarrow <i>res</i> : bool		
1: <code>resienTablero(t, fila, columna) ∧¬t.casilleros[<i>fila</i>][<i>columna</i>].casillero.ocupada</code>		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		

IESTAOCUPADA(in <i>t</i> : tab, in <i>fila</i> : nat, in <i>columna</i> : nat) \longrightarrow <i>res</i> : bool		
1:	resienTablero(<i>t</i> , <i>fila</i> , <i>columna</i>) \wedge <i>t.casilleros</i> [<i>fila</i>][<i>columna</i>]). <i>casillero.ocupada</i>	$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		

IPONERLETRAS(in/out <i>t</i> : tab, in <i>ocurrencia</i> : conjuntoLineal(<i>tupla</i> (nat,nat,letra)))		
1:	itcrearIT(<i>o</i>)	$\triangleright \mathcal{O}(1)$
2:	mientras IHAYSIGUIENTE(IT) hacer	$\triangleright \mathcal{O}(\text{tamaño}(\text{ocurrencia}))$
3:	si IESTAOCUPADA(TAB, FICHA0.FILA, FICHA0.COLUMNNA) entonces	
4:	RESFALSE	$\triangleright \mathcal{O}(1)$
5:	end si	
6:	iavanzar(it)	$\triangleright \mathcal{O}(1)$
7:	end mientras	
<u>Complejidad:</u> $\mathcal{O}(\text{tamaño}(\text{ocurrencia}))$		

ISONCELDASLIBRES?(in <i>t</i> : tab, in <i>ocurrencia</i> : conjuntoLineal(<i>tupla</i> (nat,nat,letra))) \longrightarrow <i>res</i> : bool		
1:	itcrearIt(<i>o</i>)	$\triangleright \mathcal{O}(1)$
2:	RESTRUE	$\triangleright \mathcal{O}(1)$
3:	mientras HAYSIGUIENTE(IT) hacer	$\triangleright (\mathcal{O}(\text{tamao}(\text{ocurrencia})))$
4:	si ESTAOCUPADA(TAB, FICHA0.FILA, FICHA0.COLUMNNA) entonces	
5:	RESFALSE	$\triangleright \mathcal{O}(1)$
6:	end si	
7:	iavanzar(it)	$\triangleright \mathcal{O}(1)$
8:	end mientras	
<u>Complejidad:</u> $\mathcal{O}(\text{tamaño}(\text{ocurrencia}))$		

4. Módulo Juego

Interfaz

se explica con: JUEGO
géneros: juego.
Operaciones básicas de Juego
<div><div>CREARNUEVOJUEGO(in jugadores : nat, in varianteDelJuego : variante, in bolsaDeFichas: cola(letra)) → res : juego</div><div><div>Pre ≡ {long(bolsaDeFichas) ≤ tamañoTablero(v) * tamañoTablero(v) + k * fichas(v) ∧ k > 0}</div><div>Post≡ {res = nuevoJuego(jugadores, varianteDelJuego, bolsaDeFichas)}</div><div>Complejidad: $\mathcal{O}(N^2 + F * K + \Sigma * K)$</div><div>Descripción: crea un nuevo juego</div><div>Aliasing: No aplica</div></div></div>
<div><div>UBICARFICHAS(in/out j : juego, in ocurrencia : o, in nuevasPalabras : conj(palabra)) → res : juego</div><div><div>Pre ≡ {j = j0JugadaValida?(j,o)∧nuevasPalabras = palabrasUbicadas(ocurrenciasDePalabras(ponerLetras(tablero(j),o))}</div><div>Post ≡ {res = ubicar(j0,o)}</div><div>Complejidad: $\mathcal{O}(\text{cardinal}(o))$</div><div>Descripción: ubica las fichas en el tablero, pasa el turno, actualiza los puntos del jugador y repone la cantidad de fichas ubicadas</div><div>Aliasing: se pasa el juego por referencia modificable</div></div></div>
<div><div>VARIANTEDELJUEGO(in j : juego) → res : variante</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res = variante(j)}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve la variante del juego</div><div>Aliasing: res por referencia no modificable</div></div></div>
<div><div>CANTJUGADORES(in j : juego) → res : nat</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res = #Jugadores(j)}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve la cantidad de jugadores</div><div>Aliasing: No aplica</div></div></div>
<div><div>REPOSITORIODELJUEGO(in j : juego) → res : cola(letras)</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res = repositorio(j)}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve el repositorio del juego</div><div>Aliasing: res por referencia (modificable)</div></div></div>
<div><div>TABLERO(in j : juego) → res : tablero</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res = tablero(j)}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve el tablero</div><div>Aliasing: se pasa por referencia el tablero (modificable)</div></div></div>
<div><div>POSDETABLERO(in j : juego, in i : nat, in j : nat) → res : tupla(bool, letra)</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res =< ocupada?(tablero(j),i,j), letra(tablero(j),i,j) >}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: dada una posición del tablero, devuelve si está ocupada y la letra que lo ocupa (si está vacía, devuelve el caracter vacío)</div><div>Aliasing: se pasa por referencia el tablero (modificable)</div></div></div>
<div><div>TURNOACTUAL(in j : juego) → res : nat</div><div><div>Pre ≡ {true}</div><div>Post ≡ {res = turno(j)}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve a quien le toca jugar</div><div>Aliasing: No aplica</div></div></div>
<div><div>FICHASDEJUGADOR(in j : juego, in i : nat) → res : arreglo(nat)</div><div><div>Pre ≡ {i < cantJugadores(j)}</div><div>Post ≡ {(∀f : ficha)(f ∈ fichas(i,j) ⇒_L f ∈ π1(res) ∧ (f, fichas(i,j) = π2(res)))}</div><div>Complejidad: $\mathcal{O}(1)$</div><div>Descripción: devuelve la cantidad de fichas de cada letra que tiene el jugador</div><div>Aliasing: No aplica</div></div></div>
<div><div>PUNTAJE(in j : juego, in i : nat, out puntajePrevio : nat) → res : nat</div></div>

$j : \text{juego} \wedge (\text{variante}(ju) = j.\text{variante}) \wedge (\#jugadores(ju) = \text{Longitud}(j.\text{jugadores}) \wedge (\text{repositorio}(ju) = j.\text{repositorio}) \wedge (\text{tablero}(ju) = j.\text{tablero}) \wedge (\text{turno}(ju) = j.\text{turno}_{de}) \wedge ((\forall i : \text{nat})(0 \leq i < \text{longitud}(j.\text{jugadores}) \Rightarrow \text{puntaje}(j, i, j.\text{jugadores}[i].\text{puntos.puntaje} = \text{puntaje}(ju, i) \wedge_L (\forall l : \text{letra})(l \in \text{fichas}(ju, i) \Rightarrow_L (l, \text{fichas}(ju, i)) = j.\text{jugadores}[i].\text{fichas}[\text{ord}^{-1}(l)]))))))$

Algoritmos

ICREARNUEVOJUEGO(in <i>cantJugadores</i> : nat, in <i>varianteDelJuego</i> : variante, in <i>bolsaDeFichas</i> : cola(letra)) \longrightarrow <i>res</i> : jueg			
1:	jueg.varvarianteDelJuego		$\triangleright \mathcal{O}(1)$
2:	jueg.repositoriobolsaDeFichas		$\triangleright \mathcal{O}(1)$
3:	tamitamaoDelTablero(var)		$\triangleright \mathcal{O}(1)$
4:	jueg.tableroicrearTablero(tam)		$\triangleright \mathcal{O}(N*N)$
5:	fichasiCantFichas(var)		$\triangleright \mathcal{O}(1)$
6:	jueg.jugadoresarray[(cantJugadores)		$\triangleright \mathcal{O}(k)$
7:	i0		$\triangleright \mathcal{O}(1)$
8:	mientras <i>i</i> < <i>cantJugadores(j)</i> hacer		$\triangleright \mathcal{O}(\text{tamao}(\text{palabra}))$
9:	jueg.jugadores[i]jugador		$\triangleright \mathcal{O}(1)$
10:	repositorioDelJugadorarrayfichas		$\triangleright \mathcal{O}(F)$
11:	fichasPorJugadorarray[nat](letrasDelAbecedario)	$\triangleright \mathcal{O}(\sigma)$	<i>jugador.puntos.puntaje</i> 0 $\triangleright \mathcal{O}(1)$
13:	jugador.puntos.ocurrenciacola<palabras		$\triangleright \mathcal{O}(1)$
14:	j0		$\triangleright \mathcal{O}(1)$
15:	mientras		
16:	hacer fichaDelJugadoriproximo(jueg.repositorio)		$\triangleright \mathcal{O}(1)$
17:	repositorioDelJugador[j]fichaDelJugador		$\triangleright \mathcal{O}(1)$
18:	jueg.repositorioidesencolar(jueg.repositorio)		$\triangleright \mathcal{O}(1)$
19:	letraDeLaFichafichaDelJugador.letra		$\triangleright \mathcal{O}(1)$
20:	fichasPorJugador[ord ⁻¹ (letraDeLaFicha)]fichasPorJugador[ord ⁻¹ (letraDeLaFicha)] + 1		$\triangleright \mathcal{O}(1)$
21:	jj+1		$\triangleright \mathcal{O}(1)$
22:	end mientras		
23:	jugador.fichasfichasPorJugador		$\triangleright \mathcal{O}(1)$
24:	ii+1		$\triangleright \mathcal{O}(1)$
25:	end mientras		
26:	jueg.turnoDe0		$\triangleright \mathcal{O}(1)$
27:	resjueg		$\triangleright \mathcal{O}(1)$
Complejidad: $\mathcal{O}(N^2 + F * K + \Sigma * K)$			

IUBICARFICHAS(in/out <i>j</i> : juego, in <i>o</i> : ocurrencia, in <i>nuevasPalabras</i> : cola(lista(letra))) \longrightarrow <i>puntajePrevio</i> : nat			
H			
1:	jugador0 j.jugadores[turno _{de} (j)]		$\triangleright \mathcal{O}(1)$
2:	itcrearIT(o)		$\triangleright \mathcal{O}(1)$
3:	ponerLetras(t, o)		$\triangleright \mathcal{O}(\text{cardinal}(\text{o}))$
4:	nroNuevaLetraord-1(iproxima(juego.repositorio))		$\triangleright \mathcal{O}(1)$
5:	jugado0[nroNuevaLetra]jugado0[nroNuevaLetra] + 1		$\triangleright \mathcal{O}(1)$
6:	mientras $\neg EsVaca?(nuevasPalabras)$ hacer	$\triangleright \mathcal{O}(m+1) = \mathcal{O}(m+1) = \mathcal{O}(Lm)$	
7:	palabra0Proximo(nuevasPalabras)		$\triangleright \mathcal{O}(1)$
8:	Encolar(jugador0.puntos.ocurrencias, palabra0)		$\triangleright \mathcal{O}(1)$
9:	Desencolar(nuevasPalabras)		$\triangleright \mathcal{O}(1)$
10:	end mientras		
11:	turnoj.turnoDe +1		$\triangleright \mathcal{O}(1)$
12:	si CANTJUGADORES(J) == TURNO entonces		
13:	TURNOTURNO - CANTJUGADORES(J)		$\triangleright \mathcal{O}(1)$
14:	end si		
15:	puntajePreviojugador0.puntos.puntaje		
Complejidad: $\mathcal{O}(\text{cardinal}(\text{o}))$			

IVARIANTEDELJUEGO(in <i>j</i> : juego) \longrightarrow <i>res</i> : variante	
resj.variante	$\triangleright \mathcal{O}(1)$

IREPOSITORIODELJUEGO(in <i>j</i> : juego) \longrightarrow <i>res</i> : cola(letras)	
resj.repositorio	$\triangleright \mathcal{O}(1)$

iTABLERO(in j : juego) → res : tab	
resj.tablero	▷ O(1)
iCANTJUGADORES(in j : juego) → res : nat	
resLongitud(j.jugadores)	▷ O(1)
iPOSDETABLERO(in j : juego, in i : nat , in j : nat) → res : tupla(bool, letra)	
resj.tablero[i][j]	▷ O(1)
iTURNOACTUAL(in j : juego) → res : nat	
resj.turnoDe	▷ O(1)
iFICHASDEJUGADOR(in j : juego, in i : nat) → res : array(nat)	
resj.jugador[i].fichas	▷ O(1)
IPUNTAJE(in j : juego, in i : nat) → res : nat	
puntosACalcular0	▷ O(1)
cj.jugadores[i].puntos.ocurrencias	▷ O(1)
mientras ¬iesVacia?(c) hacer piproximo(c)	▷ O(1)
cidesencolar(c)	▷ O(1)
puntosDeLaPalabra0	
j0	▷ O(1)
mientras j <ilong(p)-1 hacer	
puntajeLetraipuntajeLetra(p[i])	▷ O(1)
puntosACalcularpuntosACalcular + puntajeLetra	▷ O(1)
end mientras	
 puntaje j.jugadores[i].puntos.puntaje	▷ O(1)
puntajepuntaje + puntosACalcular	▷ O(1)
respuntaje =0	
IPALABRASJUGADAS(in j : juego, in o : ocurrencia) → res : cola(lista(letra))	
tablerotablero(juego)	▷ O(1)
esHorizontal, esVerticalHorizontalOVertical(o)	▷ O(o.cardinal)
sentidotrue	▷ O(1)
si esVertical entonces	
sentidofalse	▷ O(1)
end si	
resPalabrasTransversales(tablero,o,sentido)	▷ O(????)
palabraPrincipalformarPalabraPrincipal(tablero, o, sentido)	▷ O(???????)
Encolar(res,palabraPrincipal)	▷ O(1)
si Cardinal(o) = 1 entonces	
palabraHorizontalformarPalabraPrincipal(tablero, o, sentido)	▷ O(???????)
si longitud(palabraPrincipal) = 1 ∧longitud(palabraHorizontal = 1) entonces Desencolar(res)	
si	
end si entonces	

5. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: servidor.

Operaciones básicas de Servidor

#JUGESPERADOS(**in** $s : \text{servidor}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \#esperados(s)\}$

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

Descripción: indica la cantidad de jugadores esperados en el servidor

Aliasing: no aplica

#JUGCONECTADOS(**in** $s : \text{servidor}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \#conectados(s)\}$

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

Descripción: indica la cantidad de jugadores conectados en el servidor

Aliasing: no aplica

CONFIGURACIONDELSERVIDOR(**in** $s : \text{servidor}$) $\rightarrow res : \text{tupla}(\text{variante}, \text{cola}(\text{letra}))$

Pre $\equiv \{\neg \text{empezo?}(s)\}$

Post $\equiv \{res =_{\text{obs}} \text{configuracion}(s)\}$

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

Descripción: indica el juego que se está jugando en el servidor

Aliasing: no aplica

JUEGODELSERVIDOR(**in** $s : \text{servidor}$) $\rightarrow res : \text{juego}$

Pre $\equiv \{\text{empezo?}(s)\}$

Post $\equiv \{res =_{\text{obs}} \text{juego}(s)\}$

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

Descripción: devuelve la variante y el repositorio de un servidor

Aliasing: no aplica

NOTISDELJUGADOR(**in** $s : \text{servidor}$, **in** $\text{jugador} : \text{nat}$) $\rightarrow res : \text{cola}(\text{notif})$

Pre $\equiv \{\text{jugador} < \#conectados(s)\}$

Post $\equiv \{res =_{\text{obs}} \text{notificaciones}(s, \text{jugador})\}$

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

Descripción: devuelve la cola de notificaciones de un jugador

Aliasing: no aplica

CREARNUEVOSESRVIDOR(**in** $\text{cantJugadores} : \text{nat}$, **in** $\text{variante} : \text{var}$, **in** $\text{bolsaDeFichas} : \text{cola}(\text{letras})$) $\rightarrow res : \text{s}$

Pre $\equiv \{\text{long}(\text{bolsaDeFichas}) \geq \text{tamavaroTablero}(\text{variante}) * \text{tamaoTablero}(\text{variante}) + \text{cantJugadores} * \#fichas(\text{variante})\}$

Post $\equiv \{s = \text{nuevoServidor}(\text{cantJugadores}, \text{variante}, \text{bolsaDeFichas})\}$

Complejidad: $\mathcal{O}(N^2 + |\Sigma| + \text{FK})$

Descripción: creaunnuevoservidor

Aliasing: se pasantodoslosinputs por referencia

CONECTARCLIENTEALSERVIDOR(**in/out** $s : \text{servidor}$)

Pre $\equiv \{s = s_0 \wedge \neg \text{empezo?}(s)\}$

Post $\equiv \{s = \text{conectarCliente}(s)\}$

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

Descripción: devuelve un servidor con un cliente que se conecta

Aliasing: pasamos el servidor por referencia

CONSULTARNOTIS(**in/out** $s : \text{servidor}$, **in** $\text{jugador} : \text{nat}$)

Pre $\equiv \{s = s_0 \wedge \text{jugador} < \#conectados(s)\}$

Post $\equiv \{\text{esVacia?}(\text{notificaciones}(s, \text{jugador})) \wedge s = \text{consultar}(s_0, \text{jugador})\}$

Complejidad: $\mathcal{O}(n)$, donde n es la cantidad de mensajes en la cola del jugador

Descripción: devuelve el servidor una vez que un jugador consulta sus notificaciones

Aliasing: pasamos al servidor por referencia

RECIBIRMENSAJE(**in/out** $s : \text{servidor}$, **in** $\text{cid} : \text{nat}$, **in** $\text{o} : \text{ocurrencia}$)

Pre $\equiv \{(s = s_0) \wedge (0 \leq \text{cid}) \wedge (\text{cid} < \#conectados(s))\}$

Post $\equiv \{s = \text{recibirMensaje}(s_0, \text{cid}, \text{o})\}$

Complejidad: $\mathcal{O}(\text{fpj} * \ln * K)$

Descripción: devuelve un servidor con un nuevo mensaje al jugador cid tras realizar una ocurrencia

Aliasing: no aplica

EMPEZOJUEGO(**in** $s : \text{servidor}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{\#conectados(s) =_{\text{obs}} \#esperados(s)\}$

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

IJUGADAVALIDA (in j : juego, in o : ocurrencia) \longrightarrow res : bool	
si Cardinal(o) == 0 entonces	$\triangleright \mathcal{O}(1)$
devolver false	$\triangleright \mathcal{O}(1)$
end si	
restrue	$\triangleright \mathcal{O}(1)$
palabrasFormadascola(lista(letra))Vacía	$\triangleright \mathcal{O}(1)$
varJ VarianteDelJuego(jueg)	$\triangleright \mathcal{O}(1)$
largoMax longitudMax(palabras _{p} untos($varJ$))	$\triangleright \mathcal{O}(1)$
si $res \wedge_L (\text{cantFichas}(\text{varJ}) < \text{cardinal}(o) \vee_L \text{largoMax} < \text{cardinal}(o))$ entonces	$\triangleright \mathcal{O}(1)$
resfalse	$\triangleright \mathcal{O}(1)$
end si	
fichasJugadorarreglo(nat) Copia(Fichas(juego.jugadores[turno_de(juego)]))	$\triangleright \mathcal{O}(\Sigma)$
si $res \wedge_L \neg \text{tieneLasFichas}(\text{fichasJugador}, o)$ entonces	$\triangleright \mathcal{O}(\text{cardinal}(o))$
resfalse	
end si	
si $res \wedge_L \neg \text{celdasLibres?}(\text{Tablero}(\text{juego}), o)$ entonces	$\triangleright \mathcal{O}(\text{cardinal}(o))$
resfalse	
end si	
esHorizontalfalse	
esVerticalfalse	
si res entonces esHorizontal, esVerticalHorizontalOVertical(o)	$\triangleright \mathcal{O}(\text{cardinal}(o))$
si $\neg(\text{esHorizontal} \vee_L \text{esVertical})$ entonces	
resfalse	$\triangleright \mathcal{O}(1)$
end si	
end si	
si $res \wedge_L \neg \text{ocurrenciaSinPosicionesRepetidas}(o)$ entonces	$\triangleright \mathcal{O}(Lm^2)$
resfalse	
end si	
sentidotrue	
si $res \wedge_L \text{esVertical}$ entonces	
sentidofalse	
end si	
si $res \wedge_L \neg \text{DistanciaNoMayorALongMax}(o, \text{sentido}, \text{largoMax})$ entonces	
resfalse	
si $res \wedge_L \text{Cardinal}(o) == 1$ entonces	
iterCreatIT(o)	
ficha0Siguiente(iter)	
palabraVertical PalabraTransversalVertical(tablero(juego), ficha0)	
si $\neg \text{palabraLegitima}(\text{varJ}, \text{palabraVertical})$ entonces	resfalse
si	
entonces	
end si	
palabraHorizontalPalabraTransversalHorizontal(tablero(juego), ficha0)	
si $res \wedge_L \neg \text{palabraLegitima}(\text{varJ}, \text{palabraHorizontal})$ entonces	resfalse
si	
end si entonces	
si $res \wedge_L \neg \text{Cardinal}(o) != 1$ entonces	
palabraPrincipal formarPalabraPrincipal(tablero, o , sentido)	
si EsVacía?(palabraPrincipal) $\vee_L \neg$ palabraLegitima?(varJ, palabraPrincipal) entonces	
resfalse	
else	
palabrasPalabrasTransversales(tablero(juego), o , sentido)	
si $\neg \text{PalabrasLegitimas?}(\text{varJ}, \text{palabras})$ entonces	
resfalse	

ICANTFICHASTIENE(in $j : \text{juego}$, in $i : \text{nat}$, in $l : \text{letra}$) $\longrightarrow res : \text{nat}$		
res0		$\triangleright \mathcal{O}(1)$
para j in longitud(j.jugadores[i].fichas) hacer		$\triangleright \mathcal{O}(\ \Sigma\)$
$resres + jugadores[i].fichas[j]$		$\triangleright \mathcal{O}(1)$
 $i++$		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(\ \Sigma\) = 0$		
Pre{ $i < \text{cantJugadores}(j)$ }		
Post{ res es la cantidad de fichas en total que tiene el jugador i)}		

ITIENELASFICHAS(in $fichas : \text{arreglo}(\text{nat})$, in $o : \text{ocurrencia}$) $\longrightarrow res : \text{nat}$		
restrue		
itcrearIT(o)		
mientras HaySiguiente(it) $\wedge res$ hacer $ordenord^{-1}(\text{Siguiente}(o).letra)$		
mientras hacer fichas[orden] fichas[orden] - 1		
si fichas[orden] <0 entonces		
resfalse		
end si		
itavanzar(it)		

Descripción: indica si el juego de un servido empezó
Aliasing: no aplica

Representación

servidor se representa con serv

donde **serv** es $\text{tupla}(\#jugadoresConectados: \text{nat}, \#jugadoresEsperados: \text{nat}, \text{notificacionesPersonales: arreglo}(\text{cola}(\text{notif})), \text{notificacionesParaTodos: tupla} \langle \text{cantidadVistos: arreglo}(\text{nat}), \text{notis: vector}(\text{notif}) \rangle)$

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

$\text{Rep}(s) \equiv \text{true} \iff (\text{long}(s.\text{juego.repositorio}) \geq \text{tamañoTablero}(s.\text{juego.variante}) * \text{tamañoTablero}(s.\text{juego.variante}) + s.\#jugadoresEsperados * \#fichas(s.\text{juego.variante})) \wedge (s.\#jugadoresEsperados > 0)$

$\text{Abs} : \text{serv } s \longrightarrow \text{servidor} \qquad \{ \text{Rep}(s) \}$

$\text{Abs}(s) \equiv se / \#esperados(se) = s.\text{jugadoresEsperados} \wedge \#conectados(se) = s.\#jugadoresConectador \wedge (\forall cid : \text{nat})(0 < cid \leq \#conectados(s) \Rightarrow_L \text{notificaciones}(se) = \text{encolarN}(s.\text{notificacionesPersonales}[cid-1], s.\text{notificacionesParaTodos.cantidadVistos}, \text{longitud}(\text{cola}), s.\text{notificacionesParaTodos.notis}))$

Algoritmos

I#JUGESPERADOS(in $s : \text{serv}$) $\longrightarrow res : \text{nat}$		
1: $ress.\#jugadoresEsperados$		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1) = 0$		

I#JUGCONECTADOS(in $s : \text{serv}$) $\longrightarrow res : \text{nat}$		
1: $ress.\#jugadoresConectados$		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		

ICONFIGURACIONDELSERVIDOR(in $s : \text{serv}$) $\longrightarrow res : \text{tupla}(\text{variante}, \text{cola}(\text{letra}))$		
1: $resupla<ivarianteDelJuego(s.\text{juego}), irepositorioDelJuego(s.\text{juego})$		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		

IJUEGODELSERVIDOR(in $s : \text{serv}$) $\longrightarrow res : \text{juego}$		
1: $ress.\text{juego}$		$\triangleright \mathcal{O}(1)$
<u>Complejidad:</u> $\mathcal{O}(1)$		

INOTIDELJUGADOR(in <i>s</i> : serv, in <i>cid</i> : nat) \longrightarrow <i>res</i> : vector<notif>		
1: notisPers s.notificacionesPersonales[<i>cid</i> -1]		$\triangleright \mathcal{O}(1)$
2: notisTodoss.notificacionesParaTodos.notis		$\triangleright \mathcal{O}(1)$
3: ns.notificacionesParaTodos.cantidadVistos[<i>cid</i> -1]		$\triangleright \mathcal{O}(1)$
4: mientras \neg iVacía(notisPers) hacer		$\triangleright \mathcal{O}(\text{longitud}(\text{notisPers}))$
5: AgregarAtras(res, proximo(notisPers))		$\triangleright \mathcal{O}(\text{longitud}(\text{res}))$
6: Desencolar(notisPers)		$\triangleright \mathcal{O}(1)$
7: end mientras		
8: l ilongitud(notisTodos)-1		$\triangleright \mathcal{O}(1)$
9: si $N \leq l$ entonces		$\triangleright \mathcal{O}(l-n)$
10: para i = n .. l hacer		$\triangleright \mathcal{O}(\text{longitud}(\text{res}))$
11: AgregarAtras(res, notisTodos[i])		
12: end para		
13: end si		
<u>Complejidad:</u> $\mathcal{O}(\text{longitud}(\text{notisPers}) + \text{cantidaddemensajesnovistosdelasnotificacionesGenerales})$		

ICREARNUEVOSEVIDOR(in <i>cantJugadores</i> : nat, in <i>variante</i> : var, in <i>bolsaDeFichas</i> : cola(letra)) \longrightarrow <i>res</i> : servidor		
1: res.#jugadoresConectados 0		$\triangleright \mathcal{O}(1)$
2: res.#jugadoresEsperados cantJugadores		$\triangleright \mathcal{O}(1)$
3: res.notificacionesPersonales array[cantJugadores]		$\triangleright \mathcal{O}(K)$
4: res.notificacionesParaTodos.cantidadVistos array[cantJugadores]		$\triangleright \mathcal{O}(K)$
5: res.notificacionesParaTodos.notis ivacio()		$\triangleright \mathcal{O}(1)$
6: res.juego icrearNuevoJuego(cantJugadores, variante, bolsaDeFichas)		$\triangleright \mathcal{O}(N^2 + \Sigma + FK)$
<u>Complejidad:</u> $\mathcal{O}(N^2 + \Sigma + FK)$		

ICONECTARCLIENTEALSERVIDOR(in/out <i>s</i> : serv)		
1: #conectadoss.#jugadoresConectados		$\triangleright \mathcal{O}(1)$
2: #esperadoss.#jugadoresEsperados		$\triangleright \mathcal{O}(1)$
3: #conectados++		$\triangleright \mathcal{O}(1)$
4: iencolar(s.notificacionesPersonales[#conectados-1],iIDCliente[#conectados])		$\triangleright \mathcal{O}(1)$
5: si #CONECTADOS < #ESPERADOS entonces		$\triangleright \mathcal{O}(1)$
6: iAgregarAtras(s.notificacionesParaTodos.notis,iEmpezar(s.juego.variante.tamaño))		$\triangleright \mathcal{O}(1)$
7: iAgregarAtras(s.notificacionesParaTodos.notis, iTurnoDe(0))		$\triangleright \mathcal{O}(1)$
8: end si		
<u>Complejidad:</u> $\mathcal{O}(1)$		

ICONSULTARNOTIS(in/out <i>s</i> : serv, in <i>cid</i> : nat)		
1: notisPers s.notificacionesPersonales[<i>cid</i> -1]		$\triangleright \mathcal{O}(1)$
2: notisTodos s.notificacionesParaTodos.notis		$\triangleright \mathcal{O}(1)$
3: ns.notificacionesParaTodos.cantidadVistos[<i>cid</i> -1]		$\triangleright \mathcal{O}(1)$
4: mientras \neg iVacía(notisPers) hacer		$\triangleright \mathcal{O}(\# \text{notificacionesPersonales})$
5: desencolar(notisPers)		$\triangleright \mathcal{O}(1)$
6: end mientras		
7: para i = n .. ilongitud(notisTodos)-1 hacer		$\triangleright \mathcal{O}(\# \text{notificaciones para todos} - \# \text{notificaciones de todos ya vistas por el jugador})$
8: s.notificacionesParaTodos.cantidadVistos[<i>cid</i> -1]++		
9: end para		
<u>Complejidad:</u> $\mathcal{O}(A + B)$, donde <i>A</i> es #notificacionesPersonales y <i>B</i> es #notificaciones para todos - #notificaciones de todos ya vistas por el jugador		

IRECIBIRMENSAJENUEVO (in/out $s : \text{serv}$, in $cid : \text{nat}$, in $o : \text{ocurrencia}$)		
1:	fichasAnterioresifichasDeJugador($s.\text{juego}, cid$)	$\triangleright \mathcal{O}(1)$
2:	esValida iempezó?(s) \wedge ($cid == s.\text{juego}.\text{turno_de}$)	$\triangleright \mathcal{O}(1)$
3:	palabrasFormadas Cola($\text{lista}(\text{letra})$) Vacía()	$\triangleright \mathcal{O}(1)$
4:	si ESVALIDA entonces	
5:	jugValida.palabrasFormadasjugadaValida($s.\text{juego}, o$)	
6:	fichasAnterioress.juego.jugadores.fichas	$\triangleright \mathcal{O}(1)$
7:	puntajePrevio ibucarFichas($s.\text{juego}, o, \text{palabrasFormadas}$)	$\triangleright \mathcal{O}(K)$
8:	fichasPostJugadas.juego.jugadores.fichas	$\triangleright \mathcal{O}(1)$
9:	iAgregarAtras($s.\text{notificacionesParaTodos}.\text{notis}, i\text{Ubicar}(cid, o)$)	$\triangleright \mathcal{O}(\ln)$, donde $\ln =$
10:	long($s.\text{notificacionesParaTodos}.\text{noti}$)	
11:	iAgregarAtras($s.\text{notificacionesParaTodos}.\text{notis}, i\text{TurnoDe}(s.j.\text{turno_de})$)	$\triangleright \mathcal{O}(\ln)$
12:	fichasRepuestasivacio()	$\triangleright \mathcal{O}(1)$
13:	para $i = 0 \dots \text{longitud}(\text{fichasPostJugada})-1$ hacer	$\triangleright \mathcal{O}(\text{fpj})$, donde $\text{longitud}(\text{ fichasPostJugada})$
14:	si FICHASANTERIORES[i] < FICHASPOSTJUGADA[i] entonces	
15:	cantfichasPostJugada[i] - fichasAnteriores[i]	$\triangleright \mathcal{O}(1)$
16:	iAgregarRapido($\text{ord}^{-1}(i)$, cant)	$\triangleright \mathcal{O}(1)$
17:	end si	
18:	end para	
19:	iencolar($s.\text{notificacionesPersonales}[\#conectados-1]$, iReponer(fichasRepuestas))	$\triangleright \mathcal{O}(1)$
20:	else iencolar($s.\text{notificacionesPersonales}[cid-1]$, iMal)	$\triangleright \mathcal{O}(1)$
21:	end si	
Complejidad: $\mathcal{O}(\text{fpj} * \ln * K)$		

IEMPEZOJUEGO(in $s : \text{serv}$) $\longrightarrow res : \text{bool}$		
1:	ress.#jugadoresEsperados = s.#jugadoresConectados	$\triangleright \mathcal{O}(1)$
Complejidad: $\mathcal{O}(1)$		

JUGADA VÁLIDA

(el servidor el que tiene que verificar si el IDC = juego.turno_de)

```
ijugadaValida (in juego : jueg, in o : ocurrencia) -> res : bool {  
  
    if( cardinal(o) == 0){  
        return true  
    }  
  
    res <- true  
    palabrasFormadas <- cola(lista(letra)) vacía  
  
    varJ <- VarianteDelJuego(jueg)  
    largoMax <- longitudMax(palabras_puntos(varJ))    O(1)    <-(fn's de trie y var a  
    implementar)  
  
    if( res && cantFichas(varJ) < cardinal(o) || largoMax < cardinal(o) ){  
        O(1)  
        res <- false  
    }  
  
    fichasJugador <- arreglo(nat) Copia( Fichas(juego.jugadores[turno_de(juego)]) )  
    O(|alfabeto|)  
  
    if( res && ¬ tieneLasFichas(fichasJugador(juego.turno_de, o) ){  
        O(cardinal(o))  
        res <- false  
    }  
  
    (checks si las posiciones son  
    válidas -> libre?)  
    if( res && ¬ celdasLibres?( Tablero(juego), o) ){  
        O(cardinal(o))  
        res <- false  
    }  
  
    esHorizontal <- bool false  
    esVertical <- bool false  
    if(res){  
        esHorizontal, esVertical <- HorizontalOVertical(o)    O(  
        cardinal(o))  
        if (¬ ( esHorizontal OR esVertical) {  
            O(1)  
            res <- false  
        }  
    }  
}
```

```

    if( res && ¬ ocurrenciaSinPosicionesRepetidas(o) ) { //O(cardinal(o)^2)
= O(Lmax^2)
        res <- false
    }

```

// sentido es true si es horizontal o false si es vertical o ambos (1 ficha)

```

bool sentido <- true
if(res && esVertical){
    sentido <- false
}

```

```

if( res && ¬ DistanciaNoMayorALongMax(o, sentido, largoMax) ){
    res <- false
}

```

```

if( res && Cardinal(o) == 1 ){ // juega una sola ficha
    iter <- crearIT(o)
    ficha0 <- Siguiente(iter)

```

```

    palabraVertical <- PalabraTransversalVertical(tablero(juego), ficha0)

```

```

    if(¬palabraLegitima(varJ, palabraVertical) ){
        res <- false
    }

```

```

    palabraHorizontal <- PalabraTransversalHorizontal(tablero(juego), ficha0)

```

```

    if(¬palabraLegitima(varJ, palabraHorizontal)){
        res <- false
    }

```

elif(res) // caso en que hay más de una ficha

```

    palabraPrincipal <- formarPalabraPrincipal( tablero, o, sentido)

```

// coment formaPalabraPrincipal devuelve 1 lista vacía si no se forma una palabra contigua

```

    if ( EsVacía?(palabraPrincipal) OR ¬palabraLegitima?(varJ, palabraPrincipal)
){

```

```

        res <- false

```

```

    }else{

```

```

        palabras <- PalabrasTransversales(tablero(juego), o, sentido)

```

```

        if( ¬ PalabrasLegítimas?(varJ, palabras) { // palabras pasado por
copia

```

```

//
O(longitud(palabras)*Lm) = O(Lm^2)
    res <- false
  }
}
}

return res
}

```

//////////////////AUXILIARES DE MODULO JUEGO //////////////////

```

iTieneLasFichas(fichas : arreglo(nat) , o : ocurrencia){
  res <- true
  it <- crearIT(o)
  while(HaySiguiete(it) && res){
    orden <- ord-1(Siguiete(o).letra)
    fichas[orden] <- fichas[orden] - 1
    if( fichas[orden] < 0 ){
      res = false
    }
    it <- avanzar(it)
  }
  return res
}

```

	Complejidad
iHorizontalOVertical(in o : ocurrencia)-> horizontal : bool, vertical : bool {	
O(o.cardinal)	
horizontal <- true	O(1)
vertical <- true	O(1)
if(Cardinal(o) != 0){	O(1)
it <- crearIT(o)	O(1)
fila0 <- Siguiete(o).fila	O(1)
col0 <- Siguiete(o).columna	O(1)
it <- Avanzar(it)	O(1)
while(HaySiguiete(it) && (horizontal vertical)){	O(o.cardinal -1)
fila1 <- Siguiete(o).fila	O(1)
col1 <- Siguiete(o).columna	O(1)
if(horizontal){	O(1)
horizontal <- fila1 == fila0	O(1)
}	

```

        if(vertical){
            vertical <- col0 != col1
        }

        fila0 <- fila1
        col0 <- col1
        it <- avanzar(it)
    }
}

return horizontal, vertical
}

iocurrenciaSinPosicionesRepetidas(in o : ocurrencia ) -> res :bool {
    bool res= true;
    it0 <- creatIT(o) // primer posicion
    while ( HaySiguiente(it0) && res){
        ficha0 <- Siguiente(it0)
        it1 <- Avanzar(it0)
        while( HaySiguiente(it1) && res){
            ficha1 <- Siguiente(it1)
            if ( ficha0.fila == ficha1.fila ) && ficha0.columna == ficha1.columna ) )
            {
                // las letras son distintas o 2 elm iguales pero como es conj no existen
                2 iguales
                res <- false;
                it1 <- Avanzar(it1)
            }
            it0 <- Avanzar(it0)
        }
    }
    return res
}

```

comentario // sentido true -> horizontal otherwise es vertical

```

imasChico( in t1 : tupla(nat,nat,letra) in t2 : tupla(nat,nat,letra), sentido : bool ) -> in res :
tupla(nat,nat,letra){
    res <- t1
    if (sentido){
        res <- t2
    }
} else{
    if(t2.columna < t1.columna){
        res <- t2
    }
}
}
return res

```

```
}
```

```
imasGrande( in t1 : tupla(nat,nat,) in t2 : tupla(nat,nat,), sentido : bool ) -> in res :  
tupla(nat,nat,){  
    res <- t1  
    if (sentido){  
        if( t2.fila > t1.fila ){  
            res <- t2  
        }  
    }else{  
        if(t2.columna > t1.columna){  
            res <- t2  
        }  
    }  
    return res  
}
```

```
iDistanciaEntreFichas( in t1 : tupla(nat,nat,) in t2 : tupla(nat,nat,), sentido : bool ) -> in res :  
nat {  
    res <- res <- t2.columna - t1.columna  
    if(sentido){  
        res <- t2.fila - t1.fila  
    }  
    return res  
}
```

// comentario sentido = true -> horizontal otherwise vertical

```
iDistanciaNoMayorALongMax(in o : ocurrencia, sentido : bool , largoMax : Nat ) -> res :bool{  
    res <- true  
    it <- crearIt(o) O(1)  
    min <- Siguiente(it) O(1)  
    max <- Siguiente(it) O(1)  
    it <- Avanzar(it) O(1)  
    while( haySiguiente(it){ O(cardinal(o)) peor caso  
O(Lmax)  
        ficha0 <- Siguiente(it) O(1)  
        min <- masChico(min, ficha0, sentido) O(1)  
        max <- masGrande(max, ficha0, sentido) O(1)  
    }  
  
    if( largoMax < DistanciaEntreFichas(min, max , sentido) ){  
        res <- false  
    }  
    return res  
}
```

// funcion que te le das una tupla y el tablero

```
iPalabraTransversalHorizontal(in tablero : tab, t0 : tupla(nat,nat,letra)) -> res : lista(letra){
  res <- lista.vacia()
  AgregarAdelante(res, t0.letra)

  fila <- t0.fila
  derecha <- t0.columna +1
  izquierda <- t0.columna -1

  while(derecha < tamaño(tablero) && Ocupada(tablero,fila, derecha)) {
O(Lmax)
    letra0 <- letra(tablero[fila][derecha])
    AgregarAtras(res, letra0 )
    derecha ++
  }

  while( 0 <= izquierda  && Ocupada(tablero,fila, izquierda) ){
O(Lmax)
    letra0 <- letra(tablero[fila][izquierda])
    AgregarAdelante(res, letra0)
    izquierda- -
  }

  return res
}
```

```
iPalabraTransversalVertical(in tablero : tab, t0 : tupla(nat,nat,letra) ) -> res : lista(letra){
  res <- lista(letra).vacia()
  AgregarAdelante(res, t0.letra)

  columna <- t0.columna
  arriba <- t0.fila-1
  abajo <- t0.fila+1

  while( 0 <= arriba && Ocupada(tablero,arriba,columna)) {
    letra0 <- letra(tablero[arriba][columna])
    AgregarAdelante(res, letra0)
    arriba - -
  }

  while( abajo < tamaño(tablero) && Ocupada(tablero,fila, abajo) ){
    letra0 <- letra(tablero[abajo][columna])
    AgregarAtras(res, letra0)
    abajo ++
  }
}
```

```

    }
    return res
}

```

iPalabrasTransversales(in tablero : tab, o : ocurrencia, sentido : bool) -> palabras :

```

Cola(Lista(letras)){
    it <- crearIT(o)
    palabras <- Cola(Lista(letras)) Vacía
    while( haySiguiente(it) ){
        ficha0 <- Siguiente(it)
        palabra0 <- Lista(letra).vacía()
        if(sentido){
            palabra0 <- palabraTransversalVertical(tablero, ficha0)
        }else{
            palabra0 <- palabraTransversalHorizontal(tablero, ficha0)
        }
        Encolar(palabras, palabra0)
        it <- Avanzar(it)
    }
    return palabras
}

```

iocurrenciaAvector(in o : ocurrencia) -> res : vector(ficha){ // O(cardinal(ocurrencia)) = O(Lmax)

```

    res <- vector(ficha) vacío
    it <- crearIT(o)
    while(HaySiguiente(it)){
        ficha <- Siguiente(it)
        AgregarAtras(res, ficha)
        it <- Avanzar(it)
    }
    return res
}

```

// Pre (ForAll t1,t2 : tupla(nat,nat,letra)) (existen n1,n2 :nat)

((vect[n1] && vect[n2]) YLuego

(pi1(t1) == pi1(t2) and pi2(t1) != p1(t2)) OR (pi1(t1) != pi1(t2) and pi2(t1) == p1(t2))

iordenarVectorDeFichas(in\out vect : vector(tupla(nat,nat,letra)) { O(n^2) bubble -> O(Lm^2)

```

    swapped <- true
    i <- longitud(vect)
    while(swapped){
        swapped <- false
        j <- 0
        while( j < i ) {
            mayorFila <- ( vect[j].fila > vect[j+1].fila )
            mayorCol <- ( vect[j].columna > vect[j+1].columna )

```

```

        if( mayorFila OR mayorCol){
            temp <- vect[j]
            vect[j] <- vect[j+1]
            vect[j+1] <- temp
            swapped <- true
        }
        j <- j+1
    }
    i <- i - 1
}
}

```

si forma una palabra contigua en el tablero la devuelve, sino devuelve un vector **VACIO**

```

iPrincipalHorizontal(in tablero : tab , in fichas : vector(fichas) ) -> principal : lista(letra){
    principal <- lista(letra).vacía
    principal <- AgregarAtras(principal, fichas[0].letra)

    fila0 <- fichas[0].fila
    columna0 <- fichas[0].columna

    izquierda <- columna0 - 1
    derecha <- columna0 + 1

    i <- 1
    esContigua <- true
    termino <- false

    while( 0 <= izquierda && hayLetra?(tablero, fila0, izquierda) ){
        letrozquierda <- letra(tablero[fila0][izquierda])
        AgregarAdelante(principal, letrozquierda)
        izquierda --
    }

    while( derecha < tamaño(tablero) && esContigua && ¬ termino) {
        if ( hayLetra(tablero, fila0, derecha) ){

            letraDerecha <- tablero[fila0][derecha].letra
            AgregarAtras(principal, letraDerecha)

        } elif ( i == longitud(fichas) ) { // terminó de formar la
            termino <- true
        } elif ( fichas[i].columna == derecha) { // válido porque guarda previa -> i <
            longitud(fichas)
            letraNueva<- fichas[i].letra
            AgregarAtras(principal, letraNueva)
            i ++
        }
    }
}

```



```

    } else {
        // hay un agujero, no forma nada
        contiguo!
        principal <- list(letra) vacio
        esContigua <- false
    }
    derecha + +
}
return principal
}

```

si forma una palabra contigua en el tablero la devuelve, sino devuelve un vector **VACIO**

```

iPrincipalVertical(in tablero : tab , in fichas : vector(fichas) ) -> principal : lista(letra){
    principal <- lista(letra).Vacía
    principal <- AgregarAtras(principal, fichas[0].letra)

    fila0 <- fichas[0].fila
    columna0 <- fichas[0].columna

    arriba <- fila0 - 1
    abajo <- fila0 + 1
    i <- 1

    esContigua <- true
    termino <- false

    while( 0 <= arriba && hayLetra?(tablero, arriba, columna0) ){
        letraArriba <- letra(tablero[arriba][columna0])
        AgregarAdelante(principal, letraArriba)
        arriba --
    }

    while( abajo < tamaño(tablero) && esContigua && ¬ termino) {

        if ( hayLetra(tablero, abajo, columna0) ){

            letraAbajo <- tablero[abajo][columna0].letra
            AgregarAtras(principal, letraAbajo)

        } elif ( i == longitud(fichas) ) {
            // terminó de formarla
            termino <- true

        } elif ( fichas[i].fila == abajo) { // válido porque guarda previa -> i <
longitud(fichas)
            letraNueva <- fichas[i].letra
            AgregarAtras(principal, letraNueva)
            i + +
        }
    }
}

```

```

        } else {
            // hay un agujero, no forma nada
            contiguo!
            principal <- list(letra) vacio
            esContigua <- false
        }
        abajo + +
    }
    return principal
}

```

Post si forma una palabra contigua en el tablero la devuelve, sino devuelve un vector **VACIO**
iFormarPalabraPrincipal(in tablero : tab, in o : ocurrencia, in sentido : bool) -> principal :
lista(letra) {

```

    fichas <- Vector(Tupla(Nat,Nat,letra) ocurrenciaAVector(o)
    ordenarVectorDeFichas(fichas)
    principal <- lista(letra).vacio

    if(sentido){
        principal <- PrincipalHorizontal(tablero, fichas)
    }else{
        principal <- PrincipalVertical(tablero, fichas)
    }

    return principal
}

```

PalabrasJugadas(in juego : juego, in o : ocurrencia) -> res : cola(lista(letra))

Pre{ jugadaValida(juego, o) &&L ubicadas0 = palabrasUbicadas(ocurrenciasDePalabras(
ponerLetras(tablero(juego), o)), o) }

Post{
 Cardinal(ubicadas0) = Longitud(res) \wedge L
 (\forall o : ocurrencia)(o \in ubicadas0 \Rightarrow L
 (\exists palabra : lista(letra))(está?(palabra,res) \wedge L Cardinal(o) = Longitud(palabra) \wedge L
 (\forall f : ficha)(f \in o \Rightarrow L (\exists let : letra)(esta?(letra, palabra) \Rightarrow L pi3(o) = palabra[i]))))

 (\forall palabra : lista(letra))(está?(palabra,res) \Rightarrow L
 (\exists o : ocurrencia)(o \in ubicadas0 \wedge L cardinal(o) = Longitud(palabra) \wedge L
 (\forall letr : letra)(esta?(letr,palabra) \Rightarrow L (\exists f : ficha)(f \in o \Rightarrow L f.letr = letr)))
)
}

Complejidad : $O(Lm^2)$

Aliasing : todos los datos de entrada se pasan por referencia

algoritmo

```
iPalabrasJugadas(in juego : juego, in o : ocurrencia) -> res : cola(lista(letra)){
    tablero <- tablero(juego)
    esHorizontal, esVertical <- HorizontalOVertical(o)
    bool sentido <- true
    if(esVertical){
        sentido <- false
    }

    res <- PalabrasTransversales(tablero, o, sentido)
    palabraPrincipal <- formarPalabraPrincipal( tablero, o, sentido)
    Encolar(res, palabraPrincipal)

    if( Cardinal(o) == 1 ){
        palabraHorizontal <- formarPalabraPrincipal( tablero, o, sentido)
        if( Longitud(palabraPrincipal) == 1 && Longitud(palabraHorizontal) == 1){
            Desencolar(res)
        }
    }

    return res
}
```