

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	3
2. Módulo Juego	3
2.1. Interfaz	3
2.2. Implementación	5
2.2.1. Representación	5
2.2.2. Invariante de Representación	6
2.2.3. Función de Abstracción	8
2.2.4. Algoritmos	9
2.2.5. Algoritmos Auxiliares	13
3. Módulo Servidor	15
3.1. Interfaz	15
3.2. Implementación	16
3.2.1. Representación	16
3.2.2. Invariante de Representación	16
3.2.3. Función de Abstracción	16
3.2.4. Algoritmos	16
4. Módulos auxiliares	19
4.1. Módulo Letra	19
4.2. Módulo Variante	19
4.2.1. Interfaz	19
4.2.2. Implementación	20
4.3. Módulo Ocurrencia	21
4.3.1. Algoritmos auxiliares	22
4.4. Módulo Notificación	22
4.5. Módulo Trie de Palabras	22
4.5.1. Interfaz	22
4.5.2. Implementación	23

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.
- $L_{\text{máx}}$ — 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: VARIANTE, COLA, LETRA, OCURRENCIA, ¿LISTA?

operaciones:

NUEVOJUEGO(in k : nat, in v : variante, in r : cola(letra)) $\rightarrow res$: juego

Pre $\equiv \{ \text{tamaño}(r) \geq \text{tamañoTablero}(v) * \text{tamañoTablero}(v) + k * \#fichas(v) \wedge k > 0 \}$

Post $\equiv \{ res =_{\text{obs}} \text{nuevoJuego}(k, v, r) \}$

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: v : variante tiene referencia **no** modificable.

r : cola(letra) tiene referencia modificable.

JUGADAVÁLIDA?(in j : juego, in o : ocurrencia) $\rightarrow res$: bool

Pre $\equiv \{ \text{true} \}$

Post $\equiv \{ res =_{\text{obs}} \text{jugadaVálida?}(j, o) \}$

Complejidad: $O(L_{\text{máx}}^2)$

Descripción: Determina si una jugada es válida.

Aliasing: Se pasa o : ocurrencia como referencia **no** modificable.

UBICAR(in/out j : juego, in o : ocurrencia)

Pre $\equiv \{ \text{jugadaVálida}(j, o) \wedge j =_{\text{obs}} J_0 \}$

Post $\equiv \{ j =_{\text{obs}} \text{ubicar}(J_0, o) \}$

Complejidad: $O(m)$, donde m es el número de fichas que se ubican.

Descripción: Ubica un conjunto de fichas en el tablero.

Aliasing: Se pasa o : ocurrencia como referencia **no** modificable.

VARIANTE(in j : juego) $\rightarrow res$: variante

Pre $\equiv \{ \text{true} \}$

Post $\equiv \{ res =_{\text{obs}} \text{variante}(j) \}$

Complejidad: $O(1)$

Descripción: Obtiene información sobre la variante del juego.

Aliasing: Se devuelve res : variante como referencia **no** modificable.

TURNO(in j : juego) $\rightarrow res$: nat

Pre $\equiv \{ \text{true} \}$

Post $\equiv \{ res =_{\text{obs}} \text{turno}(j) \}$

Complejidad: $O(1)$

Descripción: Obtiene al jugador del turno actual.

TIEMPO(**in** j : juego) $\rightarrow res$: nat

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res \text{ es igual a la cantidad de generadores "ubicar" de } j\}$

Complejidad: $O(1)$

Descripción: Obtiene la cantidad de jugadas totales que se hicieron desde que empezó el juego.

PUNTAJE(**in** j : juego, **in** i : nat) $\rightarrow res$: nat

Pre $\equiv \{i < \#jugadores(j)\}$

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

Complejidad: $O(1 + m \cdot L_{\text{máx}})$, 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.

ENTABLERO?(**in** J : juego, **in** i : nat, **in** j : nat) $\rightarrow res$: bool

Pre $\equiv \{\text{true}\}$

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

Complejidad: $O(1)$

Descripción: Determina si una coordenada (i, j) está en el rango del tablero.

HAYFICHA?(**in** J : juego, **in** i : nat, **in** j : nat) $\rightarrow res$: bool

Pre $\equiv \{\text{enTablero?}(\text{tablero}(J), i, j)\}$

Post $\equiv \{res =_{\text{obs}} \text{hayLetra?}(\text{tablero}(J), i, j)\}$

Complejidad: $O(1)$

Descripción: Determina si una celda del tablero dada una coordenada (i, j) está ocupada por una letra.

FICHA(**in** J : juego, **in** i : nat, **in** j : nat) $\rightarrow res$: letra

Pre $\equiv \{\text{enTablero?}(\text{tablero}(J), i, j) \wedge_L \text{hayLetra?}(\text{tablero}(J), i, j)\}$

Post $\equiv \{res =_{\text{obs}} \text{letra}(\text{tablero}(J), i, j)\}$

Complejidad: $O(1)$

Descripción: Obtiene el contenido de una celda del tablero dada una coordenada (i, j) .

TIEMPOFICHA(**in** J : juego, **in** i : nat, **in** j : nat) $\rightarrow res$: nat

Pre $\equiv \{\text{enTablero?}(\text{tablero}(J), i, j) \wedge_L \text{hayLetra?}(\text{tablero}(J), i, j)\}$

Post $\equiv \{res \text{ es igual a la cantidad de generadores "ubicar" de } j \text{ desde que empezó el juego hasta que hubo un "ubicar" con una ocurrencia que contenía esa coordenada.}\}$

Complejidad: $O(1)$

Descripción: Obtiene el momento en que una ficha del tablero fue puesta dada una coordenada (i, j) .

#LETRATIENEJUGADOR(**in** j : juego, **in** x : letra, **in** i : nat) $\rightarrow res$: nat

Pre $\equiv \{i < \#jugadores(j)\}$

Post $\equiv \{res =_{\text{obs}} \#(x, \text{fichas}(j, i))\}$

Complejidad: $O(1)$

Descripción: Dada una cierta letra x del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra.

2.2. Implementación

2.2.1. Representación

juego se representa con `juego_estr`

```
donde juego_estr es tupla(  
    tablero: array_dimensionable(array_dimensionable(puntero(tupla(letra,nat))))  
    , jugadores: array_dimensionable(jugador)  
    , tiempo: nat  
    , repositorio: cola(letra)  
    , variante: variante  
)  
  
y jugador es tupla(  
    puntaje: nat  
    , historial: lista(tupla(ocurrencia: ocurrencia, tiempo: nat))  
    , historialSinVacias: lista(tupla(ocurrencia: ocurrencia, tiempo: nat))  
    , jugadasSinCalcularPuntaje: nat  
    , cantFichasPorLetra: array_dimensionable(nat)  
)
```

2.2.2. Invariante de Representación

Rep : juego_estr \longrightarrow bool

Rep(e) \equiv $tam(e.tablero) = tamañoTablero(e.variante) \wedge$
 $(\forall i : \mathbf{nat})(i < tam(e.tablero) \Rightarrow_L tam(e.tablero[i]) = tam(e.tablero)) \wedge_L$
 $(\forall i, j : \mathbf{nat})((i, j < tam(e.tablero) \wedge_L e.tablero[i][j] \neq \mathbf{NULL}) \Rightarrow_L e.tablero[i][j].tiempo < e.tiempo) \wedge_L$
 $(\forall i : \mathbf{nat})(i < tam(e.jugadores) \Rightarrow_L ($
 $tam(e.jugadores[i].cantFichasPorLetra) = \mathbf{DOM}() \wedge_L$
 $\sum_{f < \mathbf{DOM}()} e.jugadores[i].cantFichasPorLetra[f] = \#fichas(e.variante) \wedge$
 $tam(e.jugadores[i].historial) = \lceil e.tiempo / tam(e.jugadores) \rceil \wedge$
 $(\forall h : \mathbf{nat})(h < long(e.jugadores[i].historial) \Rightarrow_L$
 $e.jugadores[i].historial[h].tiempo = h * tam(e.jugadores) + i \wedge$
 $(\forall p, q : \mathbf{nat})(\forall l, l' : \mathbf{letra})($
 $\{ \langle p, q, l \rangle, \langle p, q, l' \rangle \} \subseteq e.jugadores[i].historial[h].ocurrencia \Rightarrow l = l'$
 $) \wedge_L$
 $e.jugadores[i].historialSinVacías =_{\text{obs}} \text{historialSinVacías}(e.jugadores[i].historial, <>) \wedge$
 $e.jugadores[i].jugadasSinCalcularPuntaje \leq tam(e.jugadores[i].historialSinVacías)$
 $)$
 $) \wedge_L$
 $ocurrenciasVálidas?(nuevoTablero(tamaño(e.tablero)), historiales(e.jugadores, 0)) \wedge_L$
 $e.tablero =_{\text{obs}} \text{ponerOcurrencias}(nuevoTablero(tamaño(e.tablero)), historiales(e)) \wedge_L$
 $(\forall i : \mathbf{nat})(i < tam(e.jugadores) \Rightarrow_L$
 $e.jugadores[i].puntaje = \sum_{k < tam(e.jugadores[i].historial) - e.jugadores[i].jugadasSinCalcularPuntaje} \text{puntajeDeOcurrenciaEnTiempo}(e, i, k)$
 $)$

donde

$historialSinVacías : \text{lista}(\text{tupla}(\text{ocurrencia}, \mathbf{nat})) \longrightarrow \text{lista}(\text{tupla}(\text{ocurrencia}, \mathbf{nat}))$
 $historialSinVacías(hcv, hsv) \equiv$
if vacía?(hcv)
 then hsv
 else
 if vacío?($\pi_1(\text{prim}(hcv))$)
 then $historialSinVacías(\text{fin}(hcv), hsv)$
 else $historialSinVacías(\text{fin}(hcv), \text{prim}(hcv) \bullet hsv)$
 fi
 fi

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

$historialesHastaTiempo : \text{ad}(\text{jugador}) \times \mathbf{nat} \times \mathbf{nat}$
 $\longrightarrow \text{multiconj}(\text{ocurrencia})$
 $historialesHastaTiempo(js, k, t) \equiv$
 if $k \geq tam(js)$
 then \emptyset
 else $historialHastaTiempo(js, k, t)$
 $\cup \text{historialesHastaTiempo}(js, k + 1, t)$
 fi

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

```

historialHastaTiempo' : lista(tupla(ocurrencia,nat)) × nat
→ multiconj(ocurrencia)
historialHastaTiempo'(ls, t) ≡
  if vacía?(ls)
  then ∅
  else historialHastaTiempo'(fin(ls), t) ∪
    if  $\pi_2(\text{prim}(\text{ls})) < t$ 
    then prim(ls)
    else ∅
  fi

fi

ocurrenciasVálidas? : tab × multiconj(ocurrencia) → bool
ocurrenciasVálidas?(t, os) ≡
  if vacía?(os)
  then true
  else celdasLibres?(t, dameUno(os)) ∧L
    ocurrenciasVálidas?(ponerLetras(t, dameUno(os)), sinUno(os))
  fi

ponerOcurrencias : tab × multiconj(ocurrencia) → tab
ponerOcurrencias(t, os) ≡
  if vacía?(os)
  then t
  else ponerOcurrencias(ponerLetras(t, dameUno(os)), sinUno(os))
  fi

puntajeDeOcurrenciaEnTiempo : estr_juego × nat × nat → nat
puntajeDeOcurrenciaEnTiempo(e, i, k) ≡
  puntajePalabrasEstr(e.variante, t',
    palabrasUbicadas(ocurrenciasDePalabras(t'), e.jugadores[i].historial[k].ocurrencia))

  donde
    tiempo ≡ e.jugadores[i].historial[k].tiempo
    t' ≡ ponerOcurrencias(nuevoTablero(tamaño(e.tablero)),
      if tiempo = 0?
      then ∅
      else historialesHastaTiempo(e.jugadores, 0, tiempo − 1)
    fi
    ∪ historialHastaTiempo(e.jugadores, i, tiempo)
    ∪ {e.jugadores[i].historial[k].ocurrencia}

puntajePalabrasEstr : variante × tab × conj(ocurrencia) → nat
puntajePalabrasEstr(v, t, os) ≡
  if vacío?(os)
  then 0
  else puntajePalabraEstr(v, t, dameUno(os))
    + puntajePalabras(v, t, sinUno(os))

```

fi

$puntajePalabraEstr : \text{variante} \times \text{tab} \times \text{ocurrencia} \longrightarrow \text{nat}$

$puntajePalabraEstr(v, t, o) \equiv$

if $vacía?(o)$

then 0

else $puntajeLetra(v, \pi_3(dameUno(o)))$

+ $puntajePalabra(v, t, sinUno(o))$

fi

2.2.3. Función de Abstracción

$Abs : \text{juego_estr } e \longrightarrow \text{juego}$

$\{\text{Rep}(e)\}$

$Abs(e) =_{\text{obs}} J : \text{juego} \mid e.\text{variante} =_{\text{obs}} \text{variante}(J) \wedge$
 $e.\text{repositorio} =_{\text{obs}} \text{repositorio}(J) \wedge$
 $e.\text{tiempo} \equiv \text{turno}(J) \text{ (mod } \#jugadores(J)) \wedge$
 $tam(e.\text{tablero}) =_{\text{obs}} \text{tamaño}(T) \wedge_L$
 $(\forall i, j : \text{nat})((enTablero?(T, i, j) \wedge_L hayLetra?(T, i, j)) \Rightarrow_L$
 $(e.\text{tablero}[i][j] \neq NULL \wedge_L letra(T, i, j) =_{\text{obs}} \pi_1(*e.\text{tablero}[i][j]))) \wedge$
 $(tam(e.jugadores) =_{\text{obs}} \#jugadores(J) \wedge_L$
 $(\forall i : \text{nat})(i < tam(e.jugadores) \Rightarrow_L ($
 $e.jugadores[i].puntaje =_{\text{obs}} puntaje(J, i) \wedge$
 $(\forall l : \text{letra})(e.cantFichasPorLetra[ORD(l)] = \#(l, fichas(J, i)))$
 $))$

2.2.4. Algoritmos

```

INUEVOJUEGO(in  $k$ : nat, in  $v$ : variante, in  $r$ : cola(letra))  $\rightarrow$  juego_estr
1:  $res.variante \leftarrow v$ 
2:  $res.repositorio \leftarrow r$ 
3:  $res.tiempo \leftarrow 0$ 
4:  $filas \leftarrow \text{CREARARREGLO}(n)$ 
5: for  $columnas \in filas$  do  $\triangleright O(N * (N + N)) = O(N^2)$ 
6:    $columnas \leftarrow \text{CREARARREGLO}(n)$ 
7:   for  $celda \in columnas$  do
8:      $celda \leftarrow \text{NULL}$ 
9:  $res.jugadores \leftarrow \text{CREARARREGLO}(k)$ 
10: for  $jugador \in res.jugadores$  do  $\triangleright O(K)$ 
11:    $jugador.puntaje \leftarrow 0$ 
12:    $jugador.historial \leftarrow \text{VACÍA}()$ 
13:    $jugador.historialSinVacias \leftarrow \text{VACÍA}()$ 
14:    $jugador.jugadasSinCalcularPuntaje \leftarrow 0$ 
15:   // Por cada jugador le damos su primer mazo de fichas del repositorio
16:    $jugador.cantFichasPorLetra \leftarrow \text{CREARARREGLO}(\text{DOM}())$ 
17:   for  $cant \in jugador.cantFichasPorLetra$  do  $\triangleright O(|\Sigma|)$ 
18:      $cant \leftarrow 0$ 
19:   for  $1 \dots \text{FICHASPORJUGADOR}(v)$  do  $\triangleright O(F)$ 
20:      $ficha \leftarrow \text{DESAPILAR}(res.repositorio)$ 
21:      $jugador.cantFichasPorLetra[\text{ORD}(ficha)]++$ 
22: 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  $e$ : estr_juego, in  $o$ : ocurrencia)  $\rightarrow$  bool
1: if CARDINAL( $o$ ) > LONGPALABRAMÁSLARGA( $e.variante$ ) then ▷ Con este if evitamos acotar por  $m$ 
2:   return false
3: for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(L_{\text{máx}})$ 
4:    $ficha \leftarrow$  SIGUIENTE( $oIt$ )
5:   if  $\neg$ ENTABLERO?( $j, \pi_1(ficha), \pi_2(ficha)$ )  $\vee_L$  HAYLETRA?( $e, \pi_1(ficha), \pi_2(ficha)$ ) then
6:     return false
7: if HAYSUPERPUESTAS?( $o$ )  $\vee$   $\neg$ (ESHORIZONTAL?( $o$ )  $\vee$  ESVERTICAL?( $o$ )) then ▷  $O(L_{\text{máx}}^2)$ 
8:   return false
9: // Ponemos las fichas de la ocurrencia para validar
10: PONERLETRAS( $e, o$ )
11: if ESHORIZONTAL?( $o$ ) then
12:   // Elegimos cualquier ficha y expandimos para atrás con i y para adelante con j para obtener toda la palabra horizontal
13:    $cualquierFicha \leftarrow$  SIGUIENTE(CREAMIT( $o$ ))
14:    $rango \leftarrow$  rangoDePalabraHorizontal( $e, cualquierFicha$ )
15:   // Ver que todas las fichas de la ocurrencia estén incluidas en el rango, sino sacamos las letras del tablero y devolvemos false
16:   for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(L_{\text{máx}})$ 
17:      $ficha \leftarrow$  SIGUIENTE( $oIt$ )
18:     if  $\neg(\pi_1(rango) \leq \pi_2(ficha) \leq \pi_2(rango))$  then
19:       SACARLETRAS( $e, o$ )
20:       return false
21:   if  $\neg$ FORMAPALABRALEGÍTIMA?( $e, rango, true, \pi_1(cualquierFicha)$ ) then ▷  $O(L_{\text{máx}})$ 
22:     SACARLETRAS( $e, o$ )
23:     return false
24:   // Vemos las palabras que se forman en las columnas
25:   for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(L_{\text{máx}})$ 
26:      $ficha \leftarrow$  SIGUIENTE(CREAMIT( $o$ ))
27:      $rango \leftarrow$  rangoDePalabraVertical( $e, ficha$ )
28:     // Si se forman nuevas palabras en las columnas ver que sean legítimas
29:     if  $\pi_1(rango) \neq \pi_2(rango) \wedge_L \neg$ FORMAPALABRALEGÍTIMA?( $e, rango, false, \pi_2(ficha)$ ) then ▷  $O(L_{\text{máx}})$ 
30:       SACARLETRAS( $e, o$ )
31:       return false
32: else
33:   // Es el mismo código del branch true pero ahora la ocurrencia es vertical
34:    $cualquierFicha \leftarrow$  SIGUIENTE(CREAMIT( $o$ ))
35:    $rango \leftarrow$  rangoDePalabraVertical( $e, cualquierFicha$ )
36:   for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(L_{\text{máx}})$ 
37:      $ficha \leftarrow$  SIGUIENTE( $oIt$ )
38:     if  $\neg(\pi_1(rango) \leq \pi_1(ficha) \leq \pi_2(rango))$  then
39:       SACARLETRAS( $e, o$ )
40:       return false
41:   if  $\neg$ FORMAPALABRALEGÍTIMA?( $e, rango, false, \pi_2(cualquierFicha)$ ) then ▷  $O(L_{\text{máx}})$ 
42:     SACARLETRAS( $e, o$ )
43:     return false
44:   // Vemos las palabras que se forman en las filas
45:   for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(L_{\text{máx}})$ 
46:      $ficha \leftarrow$  SIGUIENTE(CREAMIT( $o$ ))
47:      $rango \leftarrow$  rangoDePalabraHorizontal( $e, ficha$ )
48:     // Si se forman nuevas palabras en las filas ver que sean legítimas
49:     if  $\pi_1(rango) \neq \pi_2(rango) \wedge_L \neg$ FORMAPALABRALEGÍTIMA?( $e, rango, true, \pi_1(ficha)$ ) then ▷  $O(L_{\text{máx}})$ 
50:       SACARLETRAS( $e, o$ )
51:       return false
52:   // Sacamos las fichas de la ocurrencia para no modificar el tablero
53:   SACARLETRAS( $j, o$ )
54:   return true

```

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

```

1:  $jugador \leftarrow j.jugadores[\text{TURNO}(j)]$ 
2: for  $oIt \leftarrow \text{CREARIT}(o)$ ; HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do  $\triangleright O(m)$ 
3:    $ficha \leftarrow \text{SIGUIENTE}(oIt)$ 
4:    $e.tablero[\pi_1(ficha)][\pi_2(ficha)] \leftarrow \&\langle \pi_3(ficha), e.tiempo \rangle$ 
5:    $jugador.cantFichasPorLetra[\text{ORD}(\pi_3(ficha))]$  – –
6:    $jugador.cantFichasPorLetra[\text{ORD}(\text{DESAPILAR}(j.repositorio))]$  + +  $\triangleright O(1)$ 
7: if  $\text{CARDINAL}(o) \neq 0$  then
8:    $\text{AGREGARATRAS}(jugador.historialSinVacía, \langle o, j.tiempo \rangle)$ 
9:    $jugador.jugadasSinCalcularPuntaje$  + +
10:  $\text{AGREGARATRAS}(jugador.historial, \langle o, j.tiempo \rangle)$ 
11:  $j.tiempo$  + +

```

IVARIANTE(**in** j : **estr_juego**) \rightarrow **variante**

```

1: return  $j.variante$ 

```

ITURNO(**in** j : **estr_juego**) \rightarrow **nat**

```

1: return  $j.tiempo \% \text{tam}(j.jugadores)$ 

```

ITIEMPO(**in** j : **estr_juego**) \rightarrow **nat**

```

1: return  $j.tiempo$ 

```

IPUNTAJE(**in** $e : \text{estr_juego}$, **in** $i : \text{nat}$) $\rightarrow \text{nat}$

```

1:  $k \leftarrow \&e.jugadores[i].jugadasSinCalcularPuntaje$ 
2:  $jugador \leftarrow e.jugadores[i]$ 
3: // Usamos el historial sin ocurrencias vacías para evitar acotar por su cantidad con vacías
4:  $histIt \leftarrow \text{CREARITULT}(jugador.historialSinVacías)$ 
5: while  $*k > 0$  do  $\triangleright$  Todo el ciclo está acotado por  $O(m \cdot L_{\text{máx}})$  porque usamos ni mas ni menos que las  $m$  fichas
6:    $jugada \leftarrow \text{ANTERIOR}(histIt)$ 
7:    $ocIt \leftarrow \text{CREARIT}(jugada.ocurrencia)$ 
8:    $esHorizontal \leftarrow \text{true}$ 
9:    $ficha \leftarrow \text{SIGUIENTE}(ocIt)$ 
10:  if HAYSIGUIENTE?(AVANZAR(ocIt)) then
11:     $esHorizontal \leftarrow \pi_1(ficha) = \pi_1(\text{SIGUIENTE}(ocIt))$ 
12:  if  $esHorizontal$  then
13:    // Sumamos las fichas de la palabra alineada horizontalmente en  $O(L_{\text{máx}})$ 
14:     $fila \leftarrow \pi_1(ficha)$ 
15:     $i \leftarrow \pi_2(ficha)$ 
16:     $j \leftarrow \pi_2(ficha) + 1$ 
17:    while ENTABLERO?( $e, fila, i$ )  $\wedge_L$  HAYLETRA?( $e, fila, i$ )  $\wedge_L$  FICHATIEMPO( $e, fila, i$ )  $\leq jugada.tiempo$  do
18:       $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, fila, i))$ 
19:       $i --$ 
20:    while ENTABLERO?( $e, fila, j$ )  $\wedge_L$  HAYLETRA?( $e, fila, j$ )  $\wedge_L$  FICHATIEMPO( $e, fila, j$ )  $\leq jugada.tiempo$  do
21:       $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, fila, j))$ 
22:       $j ++$ 
23:    // Sumamos el resto de las fichas de las palabras cruzadas verticalmente en  $O(\#jugada.ocurrencia \cdot L_{\text{máx}})$ 
24:    for  $ocIt \leftarrow \text{CREARIT}(jugada.ocurrencia)$ ; HAYSIGUIENTE(ocIt); AVANZAR(ocIt) do
25:       $col \leftarrow \pi_2(ficha)$ 
26:       $i \leftarrow \pi_1(ficha) - 1$   $\triangleright$  Así evitamos sumar de vuelta la ficha
27:       $j \leftarrow \pi_1(ficha) + 1$ 
28:      while ENTABLERO?( $e, i, col$ )  $\wedge_L$  HAYLETRA?( $e, i, col$ )  $\wedge_L$  FICHATIEMPO( $e, i, col$ )  $\leq jugada.tiempo$  do
29:         $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, i, col))$ 
30:         $i --$ 
31:      while ENTABLERO?( $e, j, col$ )  $\wedge_L$  HAYLETRA?( $e, j, col$ )  $\wedge_L$  FICHATIEMPO( $e, j, col$ )  $\leq jugada.tiempo$  do
32:         $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, j, col))$ 
33:         $j ++$ 
34:  else
35:    // Lo mismo que el branch true pero verticalmente
36:     $col \leftarrow \pi_2(ficha)$ 
37:     $i \leftarrow \pi_1(ficha)$ 
38:     $j \leftarrow \pi_1(ficha) + 1$ 
39:    while ENTABLERO?( $e, i, col$ )  $\wedge_L$  HAYLETRA?( $e, i, col$ )  $\wedge_L$  FICHATIEMPO( $e, i, col$ )  $\leq jugada.tiempo$  do
40:       $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, i, col))$ 
41:       $i --$ 
42:    while ENTABLERO?( $e, j, col$ )  $\wedge_L$  HAYLETRA?( $e, j, col$ )  $\wedge_L$  FICHATIEMPO( $e, j, col$ )  $\leq jugada.tiempo$  do
43:       $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, j, col))$ 
44:       $j ++$ 
45:    for  $ocIt \leftarrow \text{CREARIT}(jugada.ocurrencia)$ ; HAYSIGUIENTE(ocIt); AVANZAR(ocIt) do
46:       $fil \leftarrow \pi_1(ficha)$ 
47:       $i \leftarrow \pi_2(ficha) - 1$   $\triangleright$  Así evitamos sumar de vuelta la ficha
48:       $j \leftarrow \pi_2(ficha) + 1$ 
49:      while ENTABLERO?( $e, fil, i$ )  $\wedge_L$  HAYLETRA?( $e, fil, i$ )  $\wedge_L$  FICHATIEMPO( $e, fil, i$ )  $\leq jugada.tiempo$  do
50:         $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, fil, i))$ 
51:         $i --$ 
52:      while ENTABLERO?( $e, fil, j$ )  $\wedge_L$  HAYLETRA?( $e, fil, j$ )  $\wedge_L$  FICHATIEMPO( $e, fil, j$ )  $\leq jugada.tiempo$  do
53:         $jugador.puntaje += \text{PUNTAJELETRA}(e.variante, \text{FICHA}(e, fil, j))$ 
54:         $j ++$ 
55:     $\text{RETROCEDER}(histIt)$ 
56:     $*k --$   $\triangleright jugadasSinCalcularPuntaje$  se va a cero, como se espera
57: return  $e.jugadores[i].puntaje$ 

```

IENTABLERO?(**in** j : **estr_juego**, **in** i : **nat**, **in** j : **nat**) \rightarrow **bool**

 1: **return** $i < tam(t) \wedge j < tam(t)$

IHAYLETRA?(**in** j : **estr_juego**, **in** i : **nat**, **in** j : **nat**) \rightarrow **bool**

 1: **return** $t[i][j] \neq \text{NULL}$

IFICHA(**in** j : **estr_juego**, **in** i : **nat**, **in** j : **nat**) \rightarrow **letra**

 1: **return** $\pi_1(*t[i][j])$

IFICHATIEMPO(**in** j : **estr_juego**, **in** i : **nat**, **in** j : **nat**) \rightarrow **nat**

 1: **return** $\pi_2(*t[i][j])$

 2: **return** **LETRA**($j.tablero, i, j$)

I#LETRATIENEJUGADOR(**in** j : **estr_juego**, **in** l : **letra**, **in** i : **nat**) \rightarrow **nat**

 1: **return** $j.jugadores[i].cantFichasPorLetra[\text{ORD}(l)]$

2.2.5. Algoritmos Auxiliares

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

 1: **for** $oIt \leftarrow \text{CREARIT}(o)$; **HAYSIGUIENTE**(oIt); **AVANZAR**(oIt) **do** $\triangleright O(\#o)$

 2: $ficha \leftarrow \text{SIGUIENTE}(oIt)$

 3: $e.tablero[\pi_1(ficha)][\pi_2(ficha)] \leftarrow \&\langle \pi_3(ficha), e.tiempo \rangle$

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

 1: **for** $oIt \leftarrow \text{CREARIT}(o)$; **HAYSIGUIENTE**(oIt); **AVANZAR**(oIt) **do** $\triangleright O(\#o)$

 2: $ficha \leftarrow \text{SIGUIENTE}(oIt)$

 3: $e.tablero[\pi_1(ficha)][\pi_2(ficha)] \leftarrow \text{NULL}$

RANGODEPALABRAHORIZONTAL(**in** e : **estr_juego**, **in** $ficha$: **tupla**(**nat**, **nat**, **letra**) \rightarrow **tupla**(**nat**, **nat**)

 1: $fila \leftarrow \pi_1(ficha)$

 2: $i \leftarrow \pi_2(ficha)$

 3: $j \leftarrow \pi_2(ficha)$

 4: **while** **IENTABLERO?**($e, fila, i$) \wedge_L **HAYFICHA?**($e, fila, i$) **do**

 5: $i --$

 6: **while** **IENTABLERO?**($e, fila, j$) \wedge_L **HAYFICHA?**($e, fila, j$) **do**

 7: $j ++$

 8: **return** $\langle i, j \rangle$

RANGODEPALABRAVERTICAL(**in** e : **estr_juego**, **in** $ficha$: **tupla**(**nat**,**nat**,**letra**)) \rightarrow **tupla**(**nat**,**nat**)

```

1:  $columna \leftarrow \pi_2(ficha)$ 
2:  $i \leftarrow \pi_1(ficha)$ 
3:  $j \leftarrow \pi_1(ficha)$ 
4: while ENTABLERO?( $e, i, columna$ )  $\wedge_L$  HAYFICHA?( $e, i, columna$ ) do
5:    $i - -$ 
6: while ENTABLERO?( $e, j, columna$ )  $\wedge_L$  HAYFICHA?( $e, j, columna$ ) do
7:    $j + +$ 
8: return  $\langle i, j \rangle$ 

```

FORMAPALABRALEGÍTIMA?(**in** e : **estr_juego**, **in** r : **tupla**(**nat**,**nat**), **in** $horizontal$: **bool**, **in** $padding$: **nat**) \rightarrow **bool**

```

1: // Hacemos un pseudo counting sort para tener la palabra en  $O(\#o)$ 
2:  $\langle i, j \rangle \leftarrow r$ 
3:  $palabra \leftarrow \text{CREARARREGLO}(j - i)$   $\triangleright$  Es arreglo_dimensionable(letra)
4: if  $horizontal$  then
5:   for  $i \leq k \leq j$  do
6:      $palabra[k - i] \leftarrow \text{FICHA}(e, padding, k)$ 
7: else
8:   for  $i \leq k \leq j$  do
9:      $palabra[k - i] \leftarrow \text{FICHA}(e, k, padding)$ 
10:  $palabra' \leftarrow \text{VACÍA}()$   $\triangleright$  Lista Enlazada
11: for  $0 \leq k < \text{tam}(palabra)$  do
12:    $\text{AGREGARATRAS}(palabra', palabra[k])$ 
13: return PALABRALEGÍTIMA?( $e.variante, palabra'$ )  $\triangleright O(L_{\text{máx}})$ 

```

3. Módulo Servidor

3.1. Interfaz

se explica con: SERVIDOR

géneros: servidor

usa: VARIANTE, COLA, LETRA, OCURRENCIA, JUEGO, NOTIFICACIÓN

operaciones:

NUEVOSEVIDOR(in $k : \text{nat}$, in $v : \text{variante}$, in $r : \text{cola}(\text{letra})$) $\rightarrow res : \text{servidor}$

Pre $\equiv \{ \text{tamaño}(r) \geq \text{tamañoTablero}(v) * \text{tamañoTablero}(v) + k * \#fichas(v) \}$

Post $\equiv \{ res =_{\text{obs}} \text{nuevoServidor}(k, v, r) \}$

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: $v : \text{variante}$ tiene referencia **no** modificable.

$r : \text{cola}(\text{letra})$ tiene referencia modificable.

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

Pre $\equiv \{ \neg \text{empezó?}(s) \wedge s =_{\text{obs}} S_0 \}$

Post $\equiv \{ s =_{\text{obs}} \text{conectarCliente}(S_0) \}$

Complejidad: $O(1)$

Descripción: Conecta un cliente a un servidor.

CONSULTAR(in/out $s : \text{servidor}$, in $cid : \text{nat}$) $\rightarrow res : \text{cola}(\text{notif})$

Pre $\equiv \{ cid \leq \#conectados(s) \wedge s =_{\text{obs}} S_0 \}$

Post $\equiv \{ s =_{\text{obs}} \text{consultar}(S_0, cid) \wedge res =_{\text{obs}} \text{notificaciones}(S_0, cid) \}$

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: Se devuelve $res : \text{cola}(\text{notif})$ como referencia **no** modificable.

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

Pre $\equiv \{ cid \leq \#conectados(s) \wedge s =_{\text{obs}} S_0 \}$

Post $\equiv \{ s =_{\text{obs}} \text{recibirMensaje}(S_0, cid, o) \}$

Complejidad: $O(m \cdot L_{\text{máx}}^2 + |\Sigma|K + FK)$

Descripción: Recibe un mensaje de un cliente.

Aliasing: Se pasa $o : \text{ocurrencia}$ como referencia **no** modificable.

CLIENTES ESPERADOS(in $s : \text{servidor}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{ \text{true} \}$

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

Complejidad: $O(1)$

Descripción: Obtiene el número de clientes esperados.

CLIENTES CONECTADOS(in $s : \text{servidor}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{ \text{true} \}$

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

Complejidad: $O(1)$

Descripción: Obtiene el número de clientes conectados.

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

Pre $\equiv \{ \text{empezó?}(s) \}$

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

Complejidad: $O(1)$

Descripción: Obtiene el juego que se está jugando en el servidor.

Aliasing: Se devuelve $res : \text{juego}$ como referencia **no** modificable.

$\text{EMPEZÓ?}(\text{in } s : \text{servidor}) \rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

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

Complejidad: $O(1)$

Descripción: Determina si la partida empezó.

3.2. Implementación

3.2.1. Representación

servidor se representa con servidor_estr

donde servidor_estr es tupla(
 juego: juego
 , jugadoresConectados: nat
 , jugadoresEsperados: nat
 , notificaciones: array_dimensionable(cola(notif))
)

3.2.2. Invariante de Representación

$\text{Rep} : \text{servidor_estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \# \text{jugadores}(e.\text{juego}) = e.\text{jugadoresEsperados} \wedge$
 $e.\text{jugadoresConectados} \leq e.\text{jugadoresEsperados} \wedge$
 $\text{tam}(e.\text{notificaciones}) = e.\text{jugadoresEsperados}$

3.2.3. Función de Abstracción

$\text{Abs} : \text{servidor_estr } e \rightarrow \text{servidor}$

$\{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} S : \text{servidor} \mid e.\text{jugadoresEsperados} =_{\text{obs}} \# \text{esperados}(S) \wedge$
 $e.\text{jugadoresConectados} =_{\text{obs}} \# \text{conectados}(S) \wedge$
 $e.\text{juego} =_{\text{obs}} \text{juego}(S) \wedge$
 $\langle \text{variante}(e.\text{juego}), \text{repositorio}(e.\text{juego}) \rangle =_{\text{obs}} \text{configuracion}(S) \wedge$
 $(\forall i : \text{nat})(i < \# \text{esperados}(S) \Rightarrow_{\text{L}} e.\text{notificaciones}[i] =_{\text{obs}} \text{notificaciones}(S, i))$

3.2.4. Algoritmos

$\text{INUEVOSERVIDOR}(\text{in } k : \text{nat}, \text{in } v : \text{variante}, \text{in } r : \text{cola(letra)}) \rightarrow \text{servidor_estr}$

```

1:  $res.\text{juego} \leftarrow \text{NUEVOJUEGO}(k, v, r)$   $\triangleright O(N^2 + |\Sigma|K + FK)$ 
2:  $res.\text{jugadoresConectados} \leftarrow 0$ 
3:  $res.\text{jugadoresEsperados} \leftarrow k$ 
4:  $res.\text{notificaciones} \leftarrow \text{CREARARREGLO}(k)$ 
5: for  $\text{notif} \in res.\text{notificaciones}$  do
6:    $\text{notif} \leftarrow \text{VACÍA}()$ 
7: return  $res$ 
```

ICONECTAR(in/out s : servidor_estr)

```

1: ENCOLAR(s.notificaciones[s.jugadoresConectados], IDCLIENTE(s.jugadoresConectados))
2: s.jugadoresConectados ++
3: if EMPEZÓ?(s) then
4:   for notif ∈ s.notificaciones do
5:     ENCOLAR(notif, EMPEZAR(TAMAÑO TABLERO(VARIANTE(s.juego)))
6:     ENCOLAR(notif, TURNODE(0))

```

ICONSULTAR(in/out s : servidor_estr, in cid : nat) → cola(notif)

```

1: res ← COPIAR(s.notificaciones[cid])                                ▷  $\Theta(\sum_{i=1}^n \text{copy}(c[i])) = \Theta(\sum_{i=1}^n 1) = \Theta(n) \in O(n)$ , con
   c = s.notificaciones[cid]
2: s.notificaciones[cid] ← VACÍA()                                ▷  $O(1)$ 
3: return res

```

IRECIBIR(in/out s : servidor_estr, in cid : nat, in o : ocurrencia)

```

1: if EMPEZÓ?(s) ∧L TURNO(s.juego) = cid ∧L JUGADA VÁLIDA?(s.juego, o) then
2:   #fichasRepuestas ← FICHASPORJUGADOR(s.configuración.variante) − #o
3:   repoViejo ← copy(s.configuración.repositorio)
4:   repuestos ← CREARARREGLO(DOM()) ▷ Asumimos que multiconj(letra) es un Arreglo Dimensionable del
   tamaño de dominio de Letra
5:   for cant ∈ repuestos do
6:     cant ← 0
7:   for 1 . . . #fichasRepuestas do
8:     ficha ← PROXIMO(repoViejo)
9:     repuestos[ORD(ficha)] ++
10:    DESENCOLAR(repoViejo)
11:   puntajeViejo ← PUNTAJE(s.juego, cid)
12:   UBICAR(s.juego, o)
13:   puntajeNuevo ← PUNTAJE(s.juego, cid)
14:   for 0 ≤ i < tam(s.notificaciones) do
15:     notif ← s.notificaciones[i]
16:     ENCOLAR(notif, UBICAR(cid, o))                                ▷ UBICAR se refiere al ítem de notificación
17:     ENCOLAR(notif, SUMAPUNTOS(cid, puntajeNuevo − puntajeViejo))
18:     if i = cid then
19:       ENCOLAR(notif, REPONER(repuestos))
20:       ENCOLAR(notif, TURNODE(TURNO(s.juego)))
21: else
22:   ENCOLAR(s.notificaciones[cid], MAL)

```

ICLIENTES ESPERADOS(in s : servidor_estr) → nat

```

1: return s.jugadoresEsperados

```

ICLIENTES CONECTADOS(in s : servidor_estr) → nat

```

1: return s.jugadoresConectados

```

IPARTIDA(in s : servidor_estr) → juego

```

1: return s.juego

```

¿EMPEZÓ?(**in** $s : \text{servidor_estr}$) \rightarrow bool

1: **return** $s.jugadoresEsperados = s.jugadoresConectados$

4. Módulos auxiliares

4.1. Módulo Letra

Se asume una implementación acorde¹ al módulo de género **letra** con las siguientes operaciones en la interfaz (todas con orden de complejidad $O(1)$):

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

4.2. Módulo Variante

4.2.1. Interfaz

se explica con: **VARIANTE**

géneros: variante

usa: DICCIONARIO LINEAL, CONJUNTO LINEAL, LISTA, LETRA, ¿TRIE DE PALABRAS?

operaciones:

```
NUEVAVARIANTE(
    in n : nat,
    in f : nat,
    in puntajes : dicc(letra, nat),
    in legítimas : conj(lista(letra))
) → res : variante
Pre ≡ { $n > 0 \wedge f > 0$ }
Post ≡ { $res =_{\text{obs}} \text{nuevaVariante}(n, f, \text{puntajes}, \text{legítimas})$ }
Complejidad:  $O(|\Sigma| + \#\text{legítimas} \cdot L_{\text{máx}})$ 
Descripción: Genera una variante de juego.
Aliasing: Se pasa puntajes : dicc(letra, nat) como referencia no modificable.
                Se pasa legítimas : conj(lista(letra)) como referencia no modificable.
```

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

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

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

```
PALABRALEGÍTIMA?(in v : variante, in p : lista(letra)) → res : bool
Pre ≡ {true}
```

¹Una buena opción es usar un **Enumerado**.

Post $\equiv \{res =_{\text{obs}} \text{palabraLegítima}(v, p)\}$

Complejidad: $O(L_{\text{máx}})$

Descripción: Determina si una palabra es legítima dentro de la variante de juego.

Aliasing: Se pasa $p : \text{lista}(\text{letra})$ como referencia **no** modificable.

LONGPALABRAMÁSLARGA(in $v : \text{variante}$) $\rightarrow res : \text{nat}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{$

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

$\}$

Complejidad: $O(1)$

Descripción: Obtiene la longitud de la palabra legítima más larga de la variante.

4.2.2. Implementación

Representación

variante se representa con variante_estr

donde variante_estr es tupla(

$\text{tablero} : \text{nat}$

, $\text{fichas} : \text{nat}$

, $\text{puntaje} : \text{array_dimensionable}(\text{nat})$

, $\text{palabras} : \text{triePalabra}$

, $\#\text{palabraMásLarga} : \text{triePalabra}$

)

Invariante de Representación

$\text{Rep} : \text{variante_estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv e.\text{tablero} > 0 \wedge e.\text{fichas} > 0 \wedge$

$\text{tam}(e.\text{puntaje}) = \text{DOM}() \wedge_L (\forall l : \text{letra}) (e.\text{puntaje}[\text{ORD}(l)] > 0) \wedge$

$\neg(\exists p : \text{lista}(\text{letra})) (\text{DEFINIDA?}(e.\text{palabras}, p) \Rightarrow \text{LONGITUD}(p) > e.\#\text{palabraMásLarga})$

Función de Abstracción

$\text{Abs} : \text{variante_estr } e \rightarrow \text{variante}$

$\{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} V : \text{variante} \mid e.\text{tablero} =_{\text{obs}} \text{tamañoTablero}(V) \wedge$

$e.\text{fichas} =_{\text{obs}} \#\text{fichas}(V) \wedge$

$(\forall l : \text{letra}) (e.\text{puntaje}[\text{ORD}(l)] =_{\text{obs}} \text{puntajeLetra}(V, l)) \wedge$

$(\forall p : \text{secu}(\text{letra})) (p \in e.\text{palabras} =_{\text{obs}} \text{palabraLegítima?}(V, l))$

Algoritmos

```

INUEVAVARIANTE(in  $n : \text{nat}$ , in  $f : \text{nat}$ , in  $\text{puntajes} : \text{dicc}(\text{letra}, \text{nat})$ , in  $\text{legítimas} : \text{conj}(\text{secu}(\text{letra}))$ )  $\rightarrow$ 
variante_estr
1:  $\text{res.tablero} \leftarrow n$ 
2:  $\text{res.fichas} \leftarrow f$ 
3:  $\text{res.puntaje} \leftarrow \text{CREARARREGLO}(\text{DOM}())$ 
4: for  $0 \leq i < \text{tam}(\text{res.puntaje})$  do  $\triangleright O(|\Sigma|)$ 
5:   if  $\text{DEFINIDO?}(\text{puntajes}, \text{ORD}^{-1}(i))$  then
6:      $\text{res.puntaje}[i] \leftarrow \text{SIGNIFICADO}(\text{puntajes}, \text{ORD}^{-1}(i))$ 
7:   else
8:      $\text{res.puntaje}[i] \leftarrow 1$ 
9:  $\text{res.palabras} \leftarrow \text{VACÍA}()$ 
10:  $\text{res.\#palabraMásLarga} \leftarrow 0$ 
11: for  $\text{lgIt} \leftarrow \text{CREARIT}(\text{lg}); \text{HAYSIGUIENTE}(\text{lgIt}); \text{AVANZAR}(\text{lgIt})$  do  $\triangleright O(\#\text{legítimas})$ 
12:    $\text{palabra} \leftarrow \text{SIGUIENTE}(\text{lgIt})$ 
13:    $\text{DEFINIR}(\text{res.palabras}, \text{palabra})$   $\triangleright O(L_{\text{máx}})$ 
14:   if  $\text{LONGITUD}(\text{palabra}) > \text{res.\#palabraMásLarga}$  then  $\triangleright O(1)$ 
15:      $\text{res.\#palabraMásLarga} \leftarrow \text{LONGITUD}(\text{palabra})$ 
16: return  $\text{res}$ 

```

```

ITAMAÑOTABLERO(in  $v : \text{variante\_estr}$ )  $\rightarrow \text{nat}$ 

```

```

1: return  $v.\text{tablero}$ 

```

```

IFICHASPORJUGADOR(in  $v : \text{variante\_estr}$ )  $\rightarrow \text{nat}$ 

```

```

1: return  $v.\text{fichas}$ 

```

```

IPUNTAJELETRA(in  $v : \text{variante\_estr}$ , in  $l : \text{letra}$ )  $\rightarrow \text{nat}$ 

```

```

1: return  $v.\text{puntaje}[\text{ORD}(l)]$ 

```

```

IPALABRALEGÍTIMA?(in  $v : \text{variante\_estr}$ , in  $p : \text{secu}(\text{letra})$ )  $\rightarrow \text{bool}$ 

```

```

1: return  $\text{DEFINIDA?}(v.\text{palabras}, p)$   $\triangleright O(L_{\text{máx}})$ 

```

```

ILONGPALABRAMÁS LARGA(in  $v : \text{variante\_estr}$ )  $\rightarrow \text{nat}$ 

```

```

1: return  $v.\#palabraMásLarga$ 

```

4.3. Módulo Ocurrencia

Es renombre de $\text{conj}(\text{tupla}(\text{nat}, \text{nat}, \text{letra}))$ con las siguientes operaciones auxiliares.

operaciones:

$\text{ESHORIZONTAL?}(\text{in } o : \text{ocurrencia}) \rightarrow \text{res} : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{\text{res} =_{\text{obs}} \text{true} \iff (\forall f, f' : \text{tupla}(\text{nat}, \text{nat}, \text{letra}))(f, f' \in o \wedge f \neq f' \Rightarrow \pi_2(f) = \pi_2(f'))\}$

Complejidad: $O(\#o^2)$

Descripción: Determina si una ocurrencia está alineada horizontalmente.

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

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

4.3.1. Algoritmos auxiliares

IESHORIZONTAL?(**in** o : ocurrencia) \rightarrow bool

```

1: colns  $\leftarrow$  VACÍO() ▷ Conjunto Lineal
2: for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(\#o)$ 
3:    $ficha \leftarrow$  SIGUIENTE( $oIt$ )
4:   AGREGAR( $colns, \pi_2(ficha)$ ) ▷  $O(\#o)$ 
5: return CARDINAL( $colns$ )=1

```

IESVERTICAL?(**in** o : ocurrencia) \rightarrow bool

```

1:  $filas \leftarrow$  VACÍO() ▷ Conjunto Lineal
2: for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(\#o)$ 
3:    $ficha \leftarrow$  SIGUIENTE( $oIt$ )
4:   AGREGAR( $filas, \pi_1(ficha)$ ) ▷  $O(\#o)$ 
5: return CARDINAL( $filas$ )=1

```

IHAYSUPERPUESTAS?(**in** o : ocurrencia) \rightarrow bool

```

1: for  $oIt \leftarrow$  CREAMIT( $o$ ); HAYSIGUIENTE( $oIt$ ); AVANZAR( $oIt$ ) do ▷  $O(\#o)$ 
2:    $ficha \leftarrow$  SIGUIENTE( $oIt$ )
3:   for  $oItSig \leftarrow$  AVANZAR( $copy(oIt)$ ); HAYSIGUIENTE( $oItSig$ ); AVANZAR( $oItSig$ ) do ▷  $O(\#o)$ 
4:      $ficha' \leftarrow$  SIGUIENTE( $oItSig$ )
5:     if  $\pi_1(ficha) = \pi_1(fichaSig) \wedge \pi_2(ficha) = \pi_2(fichaSig)$  then
6:       return true
7: return false

```

4.4. Módulo Notificación

Asumimos que existe el tipo notif.

4.5. Módulo Trie de Palabras

4.5.1. Interfaz

se explica con: CONJUNTO(SECUENCIA(LETRA))

géneros: triePalabra

operaciones:

VACÍO() $\rightarrow res : \text{triePalabra}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} \emptyset\}$

Complejidad: $O(1)$

Descripción: Genera un trie de palabras.

DEFINIR(in/out $t : \text{triePalabra}$, in $p : \text{lista(letra)}$)

Pre $\equiv \{p \notin t\}$

Post $\equiv \{p \in t\}$

Complejidad: $O(L_{\text{máx}})$

Descripción: Define una palabra en el trie.

Aliasing: Se pasa $p : \text{lista(letra)}$ como referencia **no** modificable.

DEFINIDA?(in $t : \text{triePalabra}$, in $p : \text{lista(letra)}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} p \in t\}$

Complejidad: $O(L_{\text{máx}})$

Descripción: Determina si una palabra está definida en el trie.

Aliasing: Se pasa $p : \text{lista(letra)}$ como referencia **no** modificable.

4.5.2. Implementación

Representación

`triePalabra` se representa con `trie_estr`

donde `trie_estr` es tupla(

hijos: `array_dimensionable(puntero(trie_estr))`
fin?: `bool`

)

Invariante de Representación

$\text{Rep} : \text{trie_estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{tam}(e.\text{hijos}) = \text{DOM}() \wedge_L$
 $(\forall h : \text{nat})(h < \text{tam}(e.\text{hijos}) \Rightarrow_L \text{definido?}(e.\text{hijos}[h]) \wedge$
 $(\forall h : \text{nat})(h < \text{tam}(e.\text{hijos}) \Rightarrow_L e.\text{hijos}[h] = \text{NULL}) \Rightarrow e.\text{fin?} = \text{true}$

Función de Abstracción

$\text{Abs} : \text{trie_estr } t \rightarrow \text{conj}(\text{secu}(\text{letra}))$

$\{\text{Rep}(t)\}$

$\text{Abs}(t) =_{\text{obs}} C : \text{conj}(\text{secu}(\text{letra})) \mid \langle \rangle \in C \Rightarrow C =_{\text{obs}} \text{palabrasDeTrie}(t) \vee_L$
 $C =_{\text{obs}} \text{palabrasDeTrie}(t) - \{\langle \rangle\}$

donde

$\text{palabrasDeTrie} : \text{trie_estr} \rightarrow \text{conj}(\text{secu}(\text{letra}))$

$\text{palabrasDeTrie}(t) \equiv \text{formarPalabrasDesde}(t, 0, \langle \rangle)$

$\text{formarPalabrasDesde} : \text{trie_estr} \times \text{nat} \times \text{secu}(\text{letra}) \rightarrow \text{conj}(\text{secu}(\text{letra}))$

$\text{formarPalabrasDesde}(t, k, \text{pre}) \equiv$

if $k = \text{DOM}()$

then if $t.\text{fin?}$ **then** $\text{Ag}(\text{pre}, \emptyset)$ **else** \emptyset **fi**

else $\text{formarPalabras}(t.\text{hijos}[k], \text{pre} \circ \text{ORD}^{-1}(k))$

$\cup \text{formarPalabrasDesde}(t, k + 1, \text{pre})$

```

    fi

    formarPalabras : puntero(trie_estr) × secu(letra) → conj(secu(letra))
    formarPalabras(p, pre) ≡
        if p = NULL
        then ∅
        else formarPalabrasDesde(*p, 0, pre)
    fi

```

Algoritmos

IVACÍO() → **trie_estr**

```

1: res.fin? ← false
2: res.hijos ← CREAMARREGLO(DOM())
3: for 0 ≤ i < tam(res.hijos) do
4:     res.hijos[i] ← NULL
5: return res

```

▷ $O(|\Sigma|)$

IDEFINIR(in/out t : trie_estr, in p : lista(letra))

```

1: pIt ← CREAMIT(p)
2: if ¬HAYSIGUIENTE(pIt) then
3:     t.fin? ← true
4: else
5:     nodo ← t.hijos[ORD(SIGUIENTE(pIt))]
6:     while nodo ≠ NULL do
7:         letra ← SIGUIENTE(pIt)
8:         nodo ← (*nodo).hijos[ORD(letra)]
9:     while HAYSIGUIENTE(pIt) do
10:        letra ← SIGUIENTE(pIt)
11:        nodo ← &VACÍO()
12:        nodo ← (*nodo).hijos[ORD(letra)]
13:        AVANZAR(pIt)
14:    nodo ← &VACÍO()
15:    (*nodo).fin? ← true

```

IDEFINIDA?(in t : trie_estr, in p : lista(letra)) → **bool**

```

1: pIt ← CREAMIT(p)
2: if ¬HAYSIGUIENTE(pIt) then
3:     return t.fin?
4: else
5:     nodo ← t.hijos[ORD(SIGUIENTE(pIt))]
6:     while HAYSIGUIENTE(pIt) ∧ nodo ≠ NULL do
7:         letra ← SIGUIENTE(pIt)
8:         nodo ← (*nodo).hijos[ORD(letra)]
9:         AVANZAR(pIt)
10:    return ¬HAYSIGUIENTE(pIt) ∧ (nodo ≠ NULL ∧L (*nodo).fin?)

```
