Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

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

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Preámbulo

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

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

2. Módulo Juego

2.1. Interfaz

se explica con: JUEGO

géneros: `juego`

usa: VARIANTE, COLA, LETRA, OCURRENCIA, ¿LISTA?

operaciones:

NUEVOJUEGO(**in** k : `nat`, **in** v : `variante`, **in** r : `cola(letra)`) $\rightarrow$ res : `juego`
Pre $\equiv \{tamaño(r) \geq tamañoTablero(v) * tamañoTablero(v) + k * \#fichas(v) \wedge k > 0\}$
Post $\equiv \{res =_{\text{obs}} nuevoJuego(k, v, r)\}$
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: Dada una cantidad de jugadores, una variante de juego y un repositorio de fichas, se inicia un nuevo juego con el tablero vacío.
Aliasing: v : `variante` tiene referencia **no** modificable.
r : `cola(letra)` tiene referencia modificable.

JUGADAVÁLIDA?(**in** j : `juego`, **in** o : `occurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} jugadaVálida?(j, o)\}$
Complejidad: $O(L^2_{\texttt{máx}})$
Descripción: Determina si una jugada es válida.
Aliasing: Se pasa o : `ocurrencia` como referencia **no** modificable.

UBICAR(**in/out** j : `juego`, **in** o : `occurrencia`)
Pre $\equiv \{jugadaVálida(j, o) \wedge j =_{\text{obs}} J_0\}$
Post $\equiv \{j =_{\text{obs}} ubicar(J_0, o)\}$
Complejidad: $O(m)$, donde m es el número de fichas que se ubican.
Descripción: Ubica un conjunto de fichas en el tablero.
Aliasing: Se pasa o : `ocurrencia` como referencia **no** modificable.

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

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

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

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

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

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

TIEMPOFICHA(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$enTablero?(tablero(J), i, j) \wedge_{\text{L}} hayLetra?(tablero(J), i, j)$}
Post $\equiv$ {res es igual a la cantidad de generadores "ubicar" de j 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 $\equiv$ {$res =_{\text{obs}} \#(x, fichas(j, i))$}
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

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

donde

$historialSinVacías$: `lista(tupla(ocurrencia,nat))` $\longrightarrow$ `lista(tupla(ocurrencia,nat))`

$historialSinVacías(hcv, hsv) \equiv$

if $vacía?(hcv)$
then hsv
else
if $vacío?(\pi_1(prim(hcv)))$
then $historialSinVacías(fin(hcv), hsv)$
else $historialSinVacías(fin(hcv), prim(hcv) \bullet hsv)$
fi

fi

$historiales$: `juego_estr` $\longrightarrow$ `multiconj(ocurrencia)`

$historiales(e') \equiv historialesHastaTiempo(e'.jugadores, 0, e.tiempo)$

$historialesHastaTiempo$: `ad(jugador)` $\times$ `nat` $\times$ `nat`
$\longrightarrow$ `multiconj(ocurrencia)`

$historialesHastaTiempo(js, k, t) \equiv$

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

$historialHastaTiempo$: `ad(jugador)` $\times$ `nat` $\times$ `nat`
$\longrightarrow$ `multiconj(ocurrencia)`

$historialHastaTiempo(js, k, t) \equiv historialHastaTiempo'(js[k].historial, t)$

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

if $vacía?(ls)$

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

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

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

fi

fi

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

if $vacía?(os)$

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

fi

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

if $vacía?(os)$

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

fi

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

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

donde

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

if $tiempo = 0?$

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

fi

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

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

if $vacío?(os)$

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

fi

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

if $vacía?(o)$

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

fi

2.2.3. Función de Abstracción

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

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

2.2.4. Algoritmos

```
iNuevoJuego(in k : nat, in v : variante, in r : cola(letra)) ⟶ juego_estr
1:  res.variante ← v
2:  res.repositorio ← r
3:  res.tiempo ← 0
4:  filas ← CrearArreglo(n)
5:  for columnas ∈ filas do                                   ▷ O(N * (N + N)) = O(N^2)
6:      columnas ← CrearArreglo(n)
7:      for celda ∈ columnas do
8:          celda ← NULL
9:  res.jugadores ← CrearArreglo(k)
10: for jugador ∈ res.jugadores do                            ▷ O(K)
11:     jugador.puntaje ← 0
12:     jugador.historial ← Vacía()
13:     jugador.historialSinVacias ← Vacía()
14:     jugador.jugadasSinCalcularPuntaje ← 0
15:     // Por cada jugador le damos su primer mazo de fichas del repositorio
16:     jugador.cantFichasPorLetra ← CrearArreglo(dom())
17:     for cant ∈ jugador.cantFichasPorLetra do              ▷ O(|Σ|)
18:         cant ← 0
19:     for 1 ... fichasPorJugador(v) do                      ▷ O(F)
20:         ficha ← Desapilar(res.repositorio)
21:         jugador.cantFichasPorLetra[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) $\longrightarrow$ bool

```
1: if Cardinal(o) > longPalabraMásLarga(e.variante) then          ▷ Con este if evitamos acotar por m
2:     return false
3: for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                          ▷ O(L_máx)
4:     ficha ← Siguiente(oIt)
5:     if ¬enTablero?(j, π1(ficha), π2(ficha)) ∨_L hayLetra?(e, π1(ficha), π2(ficha)) then
6:         return false
7: if haySuperpuestas?(o) ∨ ¬(esHorizontal?(o) ∨ esVertical?(o)) then                ▷ O(L²_máx)
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 ← Siguiente(CrearIt(o))
14:     rango ← 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 ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                     ▷ O(L_máx)
17:         ficha ← Siguiente(oIt)
18:         if ¬(π1(rango) ≤ π2(ficha) ≤ π2(rango)) then
19:             sacarLetras(e, o)
20:             return false
21:     if ¬formaPalabraLegítima?(e, rango, true, π1(cualquierFicha)) then       ▷ O(L_máx)
22:         sacarLetras(e, o)
23:         return false
24:     // Vemos las palabras que se forman en las columnas
25:     for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                     ▷ O(L_máx)
26:         ficha ← Siguiente(CrearIt(o))
27:         rango ← rangoDePalabraVertical(e, ficha)
28:         // Si se forman nuevas palabras en las columnas ver que sean legítimas
29:         if π1(rango) ≠ π2(rango) ∧_L ¬formaPalabraLegítima?(e, rango, false, π2(ficha)) then   ▷ O(L_máx)
30:             sacarLetras(e, o)
31:             return false
32: else
33:     // Es el mismo código del branch true pero ahora la ocurencia es vertical
34:     cualquierFicha ← Siguiente(CrearIt(o))
35:     rango ← rangoDePalabraVertical(e, cualquierFicha)
36:     for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                     ▷ O(L_máx)
37:         ficha ← Siguiente(oIt)
38:         if ¬(π1(rango) ≤ π1(ficha) ≤ π2(rango)) then
39:             sacarLetras(e, o)
40:             return false
41:     if ¬formaPalabraLegítima?(e, rango, false, π2(cualquierFicha)) then      ▷ O(L_máx)
42:         sacarLetras(e, o)
43:         return false
44:     // Vemos las palabras que se forman en las filas
45:     for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                     ▷ O(L_máx)
46:         ficha ← Siguiente(CrearIt(o))
47:         rango ← rangoDePalabraHorizontal(e, ficha)
48:         // Si se forman nuevas palabras en las filas ver que sean legítimas
49:         if π1(rango) ≠ π2(rango) ∧_L ¬formaPalabraLegítima?(e, rango, true, π1(ficha)) then   ▷ O(L_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 ← j.jugadores[turno(j)]
2:  for oIt ← CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) do                ▷ O(m)
3:      ficha ← Siguiente(oIt)
4:      e.tablero[π1(ficha)][π2(ficha)] ← &⟨π3(ficha), e.tiempo⟩
5:      jugador.cantFichasPorLetra[ord(π3(ficha))] − −
6:      jugador.cantFichasPorLetra[ord(Desapilar(j.repositorio))] + +       ▷ O(1)
7:  if Cardinal(o) ≠ 0 then
8:      AgregarAtras(jugador.historialSinVacía, ⟨o, j.tiempo⟩)
9:      jugador.jugadasSinCalcularPuntaje + +
10: AgregarAtras(jugador.historial, ⟨o, j.tiempo⟩)
11: j.tiempo + +
```

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

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

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

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

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

```
1:  return j.tiempo
```

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

```
1:  k ← &e.jugadores[i].jugadasSinCalcularPuntaje
2:  jugador ← e.jugadores[i]
3:  // Usamos el historial sin ocurrencias vacías para evitar acotar por su cantidad con vacías
4:  histIt ← CrearItUlt(jugador.historialSinVacías)
5:  while *k > 0 do   ▷ Todo el ciclo está acotado por O(m · L_máx) porque usamos ni mas ni menos que las m fichas
6:      jugada ← Anterior(histIt)
7:      ocIt ← CrearIt(jugada.ocurrencia)
8:      esHorizontal ← true
9:      ficha ← Siguiente(ocIt)
10:     if HaySiguiente?(Avanzar(ocIt)) then
11:         esHorizontal ← π1(ficha) = π1(Siguiente(ocIt))
12:     if esHorizontal then
13:         // Sumamos las fichas de la palabra alineada horizontalmente en O(L_máx)
14:         fila ← π1(ficha)
15:         i ← π2(ficha)
16:         j ← π2(ficha) + 1
17:         while enTablero?(e, fila, i) ∧L hayLetra?(e, fila, i) ∧L fichaTiempo(e, fila, i) ≤ jugada.tiempo do
18:             jugador.puntaje += puntajeLetra(e.variante, ficha(e, fila, i))
19:             i − −
20:         while enTablero?(e, fila, j) ∧L hayLetra?(e, fila, j) ∧L fichaTiempo(e, fila, j) ≤ jugada.tiempo do
21:             jugador.puntaje += puntajeLetra(e.variante, ficha(e, fila, j))
22:             j + +
23:         // Sumamos el resto de las fichas de las palabras cruzadas verticalmente en O(#jugada.ocurrencia · L_máx)
24:         for ocIt ← CrearIt(jugada.ocurrencia); HaySiguiente(ocIt); Avanzar(ocIt) do
25:             col ← π2(ficha)
26:             i ← π1(ficha) − 1                                   ▷ Así evitamos sumar de vuelta la ficha
27:             j ← π1(ficha) + 1
28:             while enTablero?(e, i, col) ∧L hayLetra?(e, i, col) ∧L fichaTiempo(e, i, col) ≤ jugada.tiempo do
29:                 jugador.puntaje += puntajeLetra(e.variante, ficha(e, i, col))
30:                 i − −
31:             while enTablero?(e, j, col) ∧L hayLetra?(e, j, col) ∧L fichaTiempo(e, j, col) ≤ jugada.tiempo do
32:                 jugador.puntaje += puntajeLetra(e.variante, ficha(e, j, col))
33:                 j + +
34:     else
35:         // Lo mismo que el branch true pero verticalmente
36:         col ← π2(ficha)
37:         i ← π1(ficha)
38:         j ← π1(ficha) + 1
39:         while enTablero?(e, i, col) ∧L hayLetra?(e, i, col) ∧L fichaTiempo(e, i, col) ≤ jugada.tiempo do
40:             jugador.puntaje += puntajeLetra(e.variante, ficha(e, i, col))
41:             i − −
42:         while enTablero?(e, j, col) ∧L hayLetra?(e, j, col) ∧L fichaTiempo(e, j, col) ≤ jugada.tiempo do
43:             jugador.puntaje += puntajeLetra(e.variante, ficha(e, j, col))
44:             j + +
45:         for ocIt ← CrearIt(jugada.ocurrencia); HaySiguiente(ocIt); Avanzar(ocIt) do
46:             fil ← π1(ficha)
47:             i ← π2(ficha) − 1                                   ▷ Así evitamos sumar de vuelta la ficha
48:             j ← π2(ficha) + 1
49:             while enTablero?(e, fil, i) ∧L hayLetra?(e, fil, i) ∧L fichaTiempo(e, fil, i) ≤ jugada.tiempo do
50:                 jugador.puntaje += puntajeLetra(e.variante, ficha(e, fil, i))
51:                 i − −
52:             while enTablero?(e, fil, j) ∧L hayLetra?(e, fil, j) ∧L fichaTiempo(e, fil, j) ≤ jugada.tiempo do
53:                 jugador.puntaje += puntajeLetra(e.variante, ficha(e, fil, j))
54:                 j + +
55:     Retroceder(histIt)
56:     *k − −                                                      ▷ 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) $\longrightarrow$ bool

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

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

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

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

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

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

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

2.2.5. Algoritmos Auxiliares

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

1: **for** $oIt \leftarrow$ CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) **do** ▷ $O(\#o)$
2: $ficha \leftarrow$ 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** $oIt \leftarrow$ CrearIt(o); HaySiguiente(oIt); Avanzar(oIt) **do** ▷ $O(\#o)$
2: $ficha \leftarrow$ Siguiente(oIt)
3: $e.tablero[\pi_1(ficha)][\pi_2(ficha)] \leftarrow$ NULL

rangoDePalabraHorizontal(**in** e : estr_juego, **in** *ficha* : tupla(nat,nat,letra)) $\longrightarrow$ tupla(nat,nat)

1: $fila \leftarrow \pi_1(ficha)$
2: $i \leftarrow \pi_2(ficha)$
3: $j \leftarrow \pi_2(ficha)$
4: **while** enTablero?$(e, fila, i) \wedge_{\text{L}}$ hayFicha?$(e, fila, i)$ **do**
5: $i--$
6: **while** enTablero?$(e, fila, j) \wedge_{\text{L}}$ hayFicha?$(e, fila, j)$ **do**
7: $j++$
8: **return** $\langle i, j \rangle$

rangoDePalabraVertical(**in** e : `estr_juego`, **in** *ficha* : `tupla(nat,nat,letra)`) $\longrightarrow$ `tupla(nat,nat)`

```
1: columna ← π2(ficha)
2: i ← π1(ficha)
3: j ← π1(ficha)
4: while enTablero?(e, i, columna) ∧L hayFicha?(e, i, columna) do
5:     i − −
6: while enTablero?(e, j, columna) ∧L hayFicha?(e, j, columna) do
7:     j + +
8: return ⟨i, j⟩
```

formaPalabraLegítima?(**in** e : `estr_juego`, **in** r : `tupla(nat,nat)`, **in** *horizontal* : `bool`, **in** *padding* : `nat`) $\longrightarrow$ `bool`

```
1: // Hacemos un pseudo counting sort para tener la palabra en O(#o)
2: ⟨i, j⟩ ← r
3: palabra ← CrearArreglo(j − i)                    ▷ Es arreglo_dimensionable(letra)
4: if horizontal then
5:     for i ≤ k ≤ j do
6:         palabra[k − i] ← ficha(e, padding, k)
7: else
8:     for i ≤ k ≤ j do
9:         palabra[k − i] ← ficha(e, k, padding)
10: palabra′ ← Vacía()                               ▷ Lista Enlazada
11: for 0 ≤ k < tam(palabra) do
12:     AgregarAtras(palabra′, palabra[k])
13: return palabraLegítima?(e.variante, palabra′)    ▷ O(L_máx)
```

3. Módulo Servidor

3.1. Interfaz

se explica con: SERVIDOR

géneros: `servidor`

usa: ??

operaciones:

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

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

CONSULTAR(**in/out** s : `servidor`, **in** cid : `nat`) $\rightarrow$ res : `cola(notif)`
Pre $\equiv \{cid \leq \#conectados(s) \wedge s =_{\text{obs}} S_0\}$
Post $\equiv \{s =_{\text{obs}} consultar(S_0, cid) \wedge res =_{\text{obs}} notificaciones(S_0, cid)\}$
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: ??

RECIBIR(**in/out** s : `servidor`, **in** cid : `nat`, **in** o : `ocurrencia`)
Pre $\equiv \{cid \leq \#conectados(s) \wedge s =_{\text{obs}} S_0\}$
Post $\equiv \{s =_{\text{obs}} recibirMensaje(S_0, cid, o)\}$
Complejidad: $O(m \cdot L^2_{\texttt{máx}} + |\Sigma|K + FK)$
Descripción: Recibe un mensaje de un cliente.
Aliasing: ??

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

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

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

EMPEZÓ?(**in** s : `servidor`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} empezó?(s)\}$
Complejidad: $O(1)$
Descripción: Determina si la partida empezó.
Aliasing: ??

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

Rep : servidor_estr $\longrightarrow$ bool

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

3.2.3. Función de Abstracción

Abs : servidor_estr e $\longrightarrow$ servidor $\{\text{Rep}(e)\}$

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

3.2.4. Algoritmos

iNuevoServidor(**in** k : `nat`, **in** v : `variante`, **in** r : `cola(letra)`) $\longrightarrow$ `servidor_estr`

```
1: res.juego ← nuevoJuego(k, v, r)                                   ▷ O(N^2 + |Σ|K + FK)
2: res.jugadoresConectados ← 0
3: res.jugadoresEsperados ← k
4: res.notificaciones ← CrearArreglo(k)
5: for notif ∈ res.notificaciones do
6:     notif ← Vacía()
7: return res
```

iConectar(**in/out** s : `servidor`)

```
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(tamañoTablero(variante(s.juego))))
6:         Encolar(notif, TurnoDe(0))
```

iConsultar(**in/out** s : `servidor`, **in** cid : `nat`) $\longrightarrow$ `cola(notif)`

```
1: res ← Copiar(s.notificaciones[cid])   ▷ Θ(Σ_{i=1}^{n} copy(c[i])) = Θ(Σ_{i=1}^{n} 1) = Θ(n) ∈ O(n), con c = s.notificaciones[cid]
2: s.notificaciones[cid] ← Vacía()                                    ▷ O(1)
3: return res
```

iRecibir(**in/out** s : `servidor`, **in** cid : `nat`, **in** o : `ocurrencia`)

```
1: if empezó?(s) ∧_L turno(s.juego) = cid ∧_L jugadaVálida?(s.juego, o) then
2:     #fichasRepuestas ← fichasPorJugador(s.configuración.variante) − #o
3:     repoViejo ← copy(s.configuración.repositorio)
4:     repuestos ← Vacío()                ▷ Asumimos que multiconj(letra) es un Diccionario Lineal
5:     for 1 ... #fichasRepuestas do
6:         ficha ← Proximo(repoViejo)
7:         if Definido?(repuestos, ficha) then
8:             Definir(repuestos, ficha, Significado(repuestos, ficha) + 1)
9:         else
10:            Definir(repuestos, ficha, 1)
11:        Desencolar(repoViejo)
12:    puntajeViejo ← puntaje(s.juego, cid)
13:    ubicar(s.juego, o)
14:    puntajeNuevo ← puntaje(s.juego, cid)
15:    for 0 ≤ i < tam(s.notificaciones) do
16:        notif ← s.notificaciones[i]
17:        Encolar(notif, Ubicar(cid, o))          ▷ Ubicar se refiere al item de notificación
18:        Encolar(notif, SumaPuntos(cid, puntajeNuevo − puntajeViejo))
19:        if i = cid then
20:            Encolar(notif, Reponer(repuestos))
21:        Encolar(notif, TurnoDe(turno(s.juego)))
22: else
23:    Encolar(s.notificaciones[cid], Mal)
```

iClientesEsperados(**in** s : `servidor_estr`) $\longrightarrow$ `nat`

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

iClientesConectados(**in** s : `servidor_estr`) $\longrightarrow$ `nat`

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

iPartida(**in** s : `servidor_estr`) $\longrightarrow$ `juego`

1: **return** $s.juego$

iEmpezó?(**in** s : `servidor_estr`) $\longrightarrow$ `bool`

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

4. Módulos auxiliares

4.1. Módulo Letra

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

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

4.2. Módulo Variante

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))`

) $\rightarrow$ *res* : `variante`
Pre $\equiv \{n > 0 \land f > 0\}$
Post $\equiv \{res =_{\text{obs}} nuevaVariante(n, f, puntajes, legítimas)\}$
Complejidad: $O(|\Sigma| + \#legítimas \cdot L_{\texttt{máx}})$
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`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} tamañoTablero(v)\}$
Complejidad: $O(1)$
Descripción: Devuelve el tamaño del tablero.

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

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

PALABRALEGÍTIMA?(**in** v : `variante`, **in** p : `lista(letra)`) $\rightarrow$ *res* : `bool`
Pre $\equiv \{\text{true}\}$

[1] Una buena opción es usar un Enumerado.

Post $\equiv \{res =_{\text{obs}} palabraLegítima(v, p)\}$
Complejidad: $O(L_{\texttt{máx}})$
Descripción: Determina si una palabra es legítima dentro de la variante de juego.
Aliasing: Se pasa p : `lista(letra)` como referencia **no** modificable.

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

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

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

4.2.2. Implementación

Representación

`variante` **se representa con** `variante_estr`

donde `variante_estr` es `tupla`(

tablero: `nat`
, *fichas*: `nat`
, *puntaje*: `array_dimensionable(nat)`
, *palabras*: `triePalabra`
, *#palabraMásLarga*: `triePalabra`

)

Invariante de Representación

Rep : variante_estr $\longrightarrow$ bool

Rep(e) $\equiv$ $e.tablero > 0 \land e.fichas > 0 \land$
$tam(e.puntaje) = \text{DOM}() \land_{\text{L}} (\forall l : \texttt{letra})(e.puntaje[\text{ORD}(l)] > 0) \land$
$\neg(\exists p : \texttt{lista(letra)})(\text{DEFINIDA?}(e.palabras, p) \Rightarrow \text{LONGITUD}(p) > e.\#palabraMásLarga)$

Función de Abstracción

Abs : variante_estr e $\longrightarrow$ variante $\{\text{Rep}(e)\}$

Abs(e) $=_{\text{obs}}$ V: variante | $e.tablero =_{\text{obs}} tamañoTablero(V) \land$
$e.fichas =_{\text{obs}} \#fichas(V) \land$
$(\forall l : \texttt{letra})(e.puntaje[\text{ORD}(l)] =_{\text{obs}} puntajeLetra(V, l)) \land$
$(\forall p : \texttt{secu(letra)})(p \in e.palabras =_{\text{obs}} palabraLegítima?(V, l))$

Algoritmos

```
iNuevaVariante(in n: nat, in f: nat, in puntajes: dicc(letra,nat), in legítimas: conj(secu(letra))) ⟶ variante_estr
1: res.tablero ← n
2: res.fichas ← f
3: res.puntaje ← CrearArreglo(dom())
4: for 0 ≤ i < tam(res.puntaje) do                                   ▷ O(|Σ|)
5:     if Definido?(puntajes, ord^-1(i)) then
6:         res.puntaje[i] ← Significado(puntajes, ord^-1(i))
7:     else
8:         res.puntaje[i] ← 1
9: res.palabras ← Vacía()
10: res.#palabraMásLarga ← 0
11: for lgIt ← CrearIt(lg); HaySiguiente(lgIt); Avanzar(lgIt) do      ▷ O(#legítimas)
12:     palabra ← Siguiente(lgIt)
13:     Definir(res.palabras, palabra)                               ▷ O(L_máx)
14:     if Longitud(palabra) > res.#palabraMásLarga then             ▷ O(1)
15:         res.#palabraMásLarga ← Longitud(palabra)
16: return res
```

```
iTamañoTablero(in v: variante_estr) ⟶ nat
1: return v.tablero
```

```
iFichasPorJugador(in v: variante_estr) ⟶ nat
1: return v.fichas
```

```
iPuntajeLetra(in v: variante_estr, in l: letra) ⟶ nat
1: return v.puntaje[ord(l)]
```

```
iPalabraLegítima?(in v: variante_estr, in p: secu(letra)) ⟶ bool
1: return Definida?(v.palabras, p)                                   ▷ O(L_máx)
```

```
iLongPalabraMásLarga(in v: variante_estr) ⟶ nat
1: return v.#palabraMásLarga
```

4.3. Módulo Ocurrencia

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

operaciones:

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

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

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

4.3.1. Algoritmos auxiliares

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

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

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

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

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

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

4.4. 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() → *res* : `triePalabra`
Pre ≡ {true}
Post ≡ {*res* =obs ∅}
Complejidad: $O(1)$
Descripción: Genera un trie de palabras.

DEFINIR(**in/out** *t* : `triePalabra`, **in** *p* : `lista(letra)`)
Pre ≡ {$p \notin t$}
Post ≡ {$p \in t$}
Complejidad: $O(L_{\texttt{máx}})$
Descripción: Define una palabra en el trie.
Aliasing: Se pasa *p* : `lista(letra)` como referencia **no** modificable.

DEFINIDA?(**in** *t* : `triePalabra`, **in** *p* : `lista(letra)`) → *res* : `bool`
Pre ≡ {true}
Post ≡ {*res* =obs $p \in t$}
Complejidad: $O(L_{\texttt{máx}})$
Descripción: Determina si una palabra está definida en el trie.
Aliasing: Se pasa *p* : `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

Rep : trie_estr ⟶ bool

$$\begin{aligned}\text{Rep}(e) \equiv\ & tam(e.hijos) = \text{DOM}() \wedge_{\text{L}} \\ & (\forall h : \texttt{nat})(h < tam(e.hijos) \Rightarrow_{\text{L}} definido?(e.hijos[h]) \wedge \\ & (\forall h : \texttt{nat})(h < tam(e.hijos) \Rightarrow_{\text{L}} e.hijos[h] = \text{NULL}) \ \Rightarrow\ e.fin? = true\end{aligned}$$

Función de Abstracción

Abs : trie_estr t ⟶ conj(secu(letra)) {Rep(t)}

$$\begin{aligned}\text{Abs}(t) =_{\text{obs}} \text{C: conj(secu(letra))} \mid <> \in C \ \Rightarrow\ & C =_{\text{obs}} palabrasDeTrie(t) \vee_{\text{L}} \\ & C =_{\text{obs}} palabrasDeTrie(t) - \{<>\}\end{aligned}$$

donde
$palabrasDeTrie$: `trie_estr` ⟶ `conj(secu(letra))`
$palabrasDeTrie(t) \equiv formarPalabrasDesde(t, 0, <>)$

$formarPalabrasDesde$: `trie_estr` × `nat` × `secu(letra)` ⟶ `conj(secu(letra))`
$formarPalabrasDesde(t, k, pre) \equiv$

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

then if $t.fin?$ **then** $Ag(pre, \emptyset)$ **else** $\emptyset$ **fi**
else $formarPalabras(t.hijos[k], pre \circ \text{ORD}^{-1}(k))$
$\cup\ formarPalabrasDesde(t, k + 1, pre)$

fi

$formarPalabras$: `puntero(trie_estr)` $\times$ `secu(letra)` $\longrightarrow$ `conj(secu(letra))`
$formarPalabras(p, pre) \equiv$
if $p = NULL$
then $\emptyset$
else $formarPalabrasDesde(*p, 0, pre)$
fi

Algoritmos

iVacío() $\longrightarrow$ `trie_estr`

```
1: res.fin? ← false
2: res.hijos ← CrearArreglo(dom())
3: for 0 ≤ i < tam(res.hijos) do                                   ▷ O(|Σ|)
4:     res.hijos[i] ← NULL
5: return res
```

iDefinir(**in/out** t : `trie_estr`, **in** p : `lista(letra)`)

```
1: pIt ← CrearIt(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)`) $\longrightarrow$ `bool`

```
1: pIt ← CrearIt(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?)
```