1. Juego()

Interfaz

parámetros formales
géneros juego
se explica con: JUEGO, VARIANTE, TABLERO, MULTICONJUNTO

Operaciones básicas de Juego

INICIAR(**in** k : `nat`, **in** v : `variante`, **in** r : `cola(letra)`) $\rightarrow$ res : `juego`
Pre $\equiv$ {k >0 $\wedge$ tamañoTablero(r) $\leq$ tamañoTablero(v)) $*$ tamañoTablero(v) + $k * fichas(v)$}
Post$\equiv$ {$res = crearJuego(k, v, r)$}
Complejidad: O($N^2 + |\sum|K + FK$)
Descripción: Crea un nuevo juego con k jugadores, de variante v y la bolsa de fichas r.
Aliasing: K se pasa por copia. La variante se pasa por referencia no mutable y el repositorio r se pasa por referencia modificable.

JUGADAVÁLIDA(**in** j : `juego`,**in** o : `ocurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res = jugadaVálida?(j,o)}
Complejidad: O(L_{max}^2)
Descripción: Determinar si una jugada es válida.
Aliasing: El juego se pasa por referencia no modificable. La ocurrencia se pasa por copia.

UBICAR(**in/out** j : `juego`, **in** o : `ocurrencia`)
Pre $\equiv$ {$j = j_0 \wedge$ jugadaVálida(j,o)}
Post $\equiv$ {j = ubicar(j_0, o)}
Complejidad: O(m)
Descripción: Ubicar un conjunto de fichas en el juego.
Aliasing: El juego se pasa por referencia modificable. La ocurrencia se pasa por copia.

VARIANTE(**in** j : `Juego`) $\rightarrow$ res : `variante`
Pre $\equiv$ {true}
Post $\equiv$ {res = variante(j)}
Complejidad: O(1)
Descripción: Obtener información sobre la variante del juego
Aliasing: El juego se pasa por referencia no modificable.

TURNO(**in** j : `juego`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ {res = turno(j)}
Complejidad: O(1)
Descripción: Obtener número de jugador a quien le toca jugar actualmente
Aliasing: El juego se pasa por referencia no modificable.

PUNTAJE(**in** j : `juego`, **in** cid : `Nat`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {cid <#jugadores(j)}
Post $\equiv$ {puntaje(j,cid)}
Complejidad: O($1 + m \cdot L_{max}$)
Descripción: Obtener el puntaje de un jugador.
Aliasing: El juego se pasa por referencia modificable. Cid se pasa por copia.

CONTENIDOTABLEROEN(**in** j : `juego`,**in** i : `Nat`,**in** j : `Nat`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {enTablero?(t,i,j)}
Post $\equiv$ {contenidoDelTablero(t,i,j)}
Complejidad: Obtener el contenido del tablero en una coordenada (i, j).
Descripción: El juego se pasa por referencia no modificable. i, j se pasan por copia.

CANTLETRA(**in** j : `Juego`, **in** cid : `Nat`, **in** j : `Nat`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {cid <#jugadores(j)}
Post $\equiv$ {res = #(x,fichas(j,cid)) (donde # es la cantidad de apariciones del elemento x en el multiconjunto.)}
Complejidad: O(1)
Descripción: Dada una letra x del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra.
Aliasing: El juego se pasa por referencia no modificable. Cid se pasa por copia.

FICHASJUGADOR(**in** j : `Juego`, **in** cid : `Nat`,) $\rightarrow$ res : `array[nat]`
Pre $\equiv$ {cid <#jugadores(j)}
Post $\equiv$ {res = fichas(j,cid) }
Complejidad: O(1)
Descripción: Devuelve las letras de un jugador
Aliasing: El juego se pasa por referencia no modificable. Cid se pasa por copia.

Representación

Juego se representa con **estr** donde

estr es tupla(tablero: array(array(tuplas(ocupada: bool, letra: letra, turno: nat))), repositorio: cola(fichas), turno: nat, variante: variante, fichasJugadores: array(array(nat), puntaje: array(nat), puntajePorSumar(array(cola(tupla(i: nat, j: nat, letra: letra, turno: nat)))))

donde variante se representa como tupla(n: nat, cantFichas: nat, puntajeFichas: array(nat), palabras: conj(palabra))

donde letra se representa como nat y palabra como array(letra)

Invariante de representación

- El tamaño del tablero tiene que ser el mismo dado por la variante.
- Todas las letras del tablero tienen que estar en el alfabeto de la variante.
- El turno del tablero tiene que coincidir con el del juego.
- Todas las letras del repositorio tienen que pertenecer al alfabeto de la variante.
- Para todas las fichas en el tablero, el turno en el que fueron ubicadas es menor igual al turno actual.
- La cantidad de fichas de la variante tiene que coincidir con la cantidad de fichas por jugador en fichasJugadores. Además la longitud de cada elemento de fichasJugadores es igual al tamaño del alfabeto de la variante.
- La palabra más larga tiene que ser menor o igual al tamaño del tablero.

Funcion de abstraccion:

$\forall e : estr$

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

Abs(e) $\equiv$ j: juego | tamaño(tablero(j)) = |estr.tablero| = e.variante.n $\wedge$
($\forall n, m : nat$)($0 \leq n, m \leq$ |$e.tablero$| $\Rightarrow_{\text{L}}$ ($hayLetra?(tablero(j), n, m) = e.tablero[n][m].ocupada \wedge$
letra(tablero(j, n, m)) = e.tablero[n][m].letra $\wedge$
turno(j) = e.turno mod #jugadores(j) $\wedge$
repositorio(j)= e.repositorio $\wedge$
|e.fichasJugadores| = |e.puntaje| = |e.puntajePorSumar| = #jugadores(j) $\wedge$
($\forall i : nat$)(0$\leq$ i <|puntaje| $\Rightarrow_{\text{L}}$e.puntaje[i] + e.puntajePorSumar[i] = puntaje(j,i)) $\wedge$
#fichas(variante(j)) = e.variante.cantFichas
($\forall i : nat$)(0$\leq$ i <|e.variante.puntajeFichas| $\Rightarrow_{\text{L}}$e.variante.puntajeFichas[i] = puntajeLetra(variante(j),i)) $\wedge$
($\forall p : palabra$)($p\in$ e.variante.palabras $\iff$ palabraLegítima?(variante(j), p)) $\wedge$
($\forall i : nat$)(0$\leq$ i <|fichasJugadores| $\Rightarrow_{\text{L}}$(0 $\leq$ k <|fichasJugadores[i]| $\Rightarrow_{\text{L}}$fichasJugadores[i][k] = #(k,fichas(j,i))))

sumarPuntajePendiente(**in** *pendiente*: `cola(tupla(i: nat, j: nat, letra: letra, turno: nat)`, **in** *j*: `juego`) → *res* : *nat*

todasFichas = vacio(multiconj)

$i \leftarrow 0$

mientras i <long(fichas) **hacer**

para j=1 hasta j=fichas[i] **hacer**

agregar(todasFichas,i)

devolver todasFichas

Algoritmos

IINICIAR(**in** k : `Nat`, **in** v : `variante`, r : `cola(letras)`) ⟶ `juego`

1: *estr.tablero* ← var tablero: arreglo estatico[v.n] de arreglo estatico[v.n] de (false,a,0) ▷ $\mathcal{O}(n^2)$
2: *estr.turno* ← 0
3: *estr.fichasJugadores* ← arreglo estatico [k] de arreglo estatico[long(estr.variante.puntajeFichas)] de nat ▷ $\mathcal{O}(k \cdot |\sum|)$
4: *estr.puntaje* ← arreglo estatico [k] de nat
5: *estr.puntajePorSumar* ← arreglo estatico [k] de cola
6: *estr.variante* ← v
7: $i \leftarrow 0$
8: **mientras** $i < k$ **hacer** ▷ $\mathcal{O}(f \cdot k)$
9: estr.fichasJugadores[i] = crearArray[long(estr.variante.puntajeFichas] de nat
10: i++
11: $i \leftarrow 0$
12: **mientras** $i < k$ **hacer** ▷ $\mathcal{O}(f \cdot k)$
13: $j \leftarrow 0$
14: **mientras** $j < estr.v.f$ **hacer**
15: estr.fichasJugadores[i][proximo(r)] = estr.fichasJugadores[i][proximo(r)] + 1
16: desencolar(r)
17: j++
18: i++
estr.repositorio ← r
19: **devolver** estr
=0

Complejidad: $\mathcal{O}(N^2 + |\sum| K + FK)$ (las lineas que no aclaran nada tienen $\mathcal{O}(1)$).

Justificación: Se crea un array de n × n, luego se recorre un diccionario de longitud $|\sum|$ y por último se asignan f fichas a cada uno de los k jugadores. Las fichas se pasan por referencia

IJUGADAVALIDA(**in** j : `Juego`, **in** o: `ocurrencia` $\longrightarrow$ `bool`

```
1:  formaVertical bool
2:  si long(o) > lMax(palabras) entonces
3:      devolver false
4:  tuplaFunAux ← (cumpleOcurrencia(j, o))                                  ▷ O(L²_max)
5:  ocurrenciaOrdenada ← tuplaFunAux.1
6:  si tuplaFunAux.0 entonces
7:      devolver false
8:  esVertical ← tuplaFunAux.2
9:
10: i ← 0
11: ocurrenciaOrdenada ← lista vacia de tupla(posicion, posicion, letra)
12: ocurrenciaOrdenada ← usar algoritmo de ordenamiento para ordenar listaOcurrencia, si esVertical
13: entonces se ordena por el primer elemento de la secuencia, sino por el segundo   ▷ O(L²_max)
14: tuplaFunAux ← (completarEspaciosOcurrencia(ocurrenciaOrdenada, j))      ▷ O(L_max)
15: ocurrenciaOrdenada ← tuplaFunAux.1
16: si tuplaFunAux.0 entonces
17:     devolver false
18: tuplaFunAux ← completarEspaciosAtrasAdelante(ocurrenciaOrdenada, j, esVertical)
19: ocurrenciaOrdenada ← tuplaFunAux.1
20: si tuplaFunAux.0 entonces
21:     devolver false
22: potencialPalabra ← array[Lmax]de str
23: i ← 0
24: mientras haySiguiente(it) hacer                                         ▷ O(L_max)
25:     potencialPalabra[i] ← siguiente(it)₂
26:     i++
27: si Pertenece?(potencialPalabra, estr.variable.palabras) entonces
28:     devolver true
29: else
30:     devolver false
```

Complejidad: $\mathcal{O}(L^2_{max})$

Justificación: La long max de la ocurrencia nunca supera lMax, ordeno la ocurrencia con un algoritmo n^2 entonces no supera $lMax^2$. Cuando ya tengo la potencial palabra, chequeo si la palabra pertenece al conjDigital, como en un trie acotado por lMax. La complejidad de la pertenencia es lMax

CUMPLEOCURRENCIA(**in** j : `juego`,**in** o : `ocurrencia` $\longrightarrow$ `tupla(esValido,ocurrenciaOrdenada, formaVertical)`

crearIt(o) it
crear array estatico de longitud lMax(palabras) de listaOcurrencia de tupla(posicion,posicion, letra)
$h \leftarrow 0$
$v \leftarrow 0$
mientras HaySiguiente(it) **hacer** ▷ $\mathcal{O}(L_{max}^2)$
  $potencialPalabraLetraUbicada \leftarrow lista[Lmax]destr$
  si $tablero[siguiente(it2)0, siguiente(it2)1].ocupada == true$ **entonces**
    devolver (false,o,false)
  $actual \leftarrow siguiente(it)$
  $it \leftarrow avanzar(it)$
  si $tablero[actual_0, actual_1]_1 == tablero[siguiente(it2)_0, siguiente(it2)_1]_1$ **entonces**
    $formaVertical \leftarrow true$
    v++
  else
    h++
    $formaVertical \leftarrow false$
  $it \leftarrow Retroceder(it)$
  $i \leftarrow 0$
  $j \leftarrow 0$
  agregarAdelate(potencialPalabraLetraUbicada,siguiente(it)$_2$)
  si formaVertical == true **entonces**
    listaOcurrencia[v] = siguiente(it)
    mientras $tablero[siguiente(it)_0 - i, siguiente(it)_1].ocupada == true$ **hacer** ▷ $\mathcal{O}(L_{max})$
      agregarAdelante(potencialPalabraLetraUbicada, $tablero[siguiente(it)_0 - i, siguiente(it)_1].letra$)
      i++
    mientras $tablero[siguiente(it) + j, siguiente(it)].ocupada == true$ **hacer** ▷ $\mathcal{O}(L_{max})$
      AgregarAtras(potencialPalabraLetraUbicada, $tablero[siguiente(it)_0 + j, siguiente(it)_1].letra$)
      j++
  else
    listaOcurrencia[h] = siguiente(it)
    mientras $tablero[siguiente(it), siguiente(it)] - i].ocupada == true$ **hacer** ▷ $\mathcal{O}(L_{max})$
      agregarAdelante(potencialPalabraLetraUbicada, $tablero[siguiente(it)_0, siguiente(it)_1 - i].letra$)
      i++
    mientras $tablero[siguiente(it), siguiente(it) + j_{.ocupada==true}$ **hacer** ▷ $\mathcal{O}(L_{max})$
      AgregarAtras(potencialPalabraLetraUbicada, $tablero[siguiente(it)_0, siguiente(it)_1 + j].letra$)
      j++
  pasarListaAArray(potencialPalabraLetraUbicada)
  si not(potencialPalabraLetraUbicada in estr.variable.palabras or long(potencialPalabraLetraUbicada) == 1) **entonces**
    devolver (false,o,false)
si h != long(o) or v != long(o) **entonces**
  devolver (false,o,false)
devolver (false,listaOcurrencia,formaVertical)
Complejidad: $\mathcal{O}(lMax^2)$
Justificación: Recorro toda la ocurrencia, long max de la ocurrencia es lMax y la agrego en un arreglo estatico, mientras recorre la ocurrencia se chequea que por cada letra ubicada, esta forma una palabra de la forma opuesta a la que se esta ubicando la palabra. Recorrer la ocurrencia tiene long(o) y chequear si forma palabra es lMax

COMPLETARESPACIOSOCURRENCIA(**in** *ocurrenciaOrdenada*: `lista`,**in** *j*: `juego` $\longrightarrow$ `tupla(noCumple: bool,ocurrenciaOrdenada: lista)`

crearIt(ocurrenciaOrdenada) it
mientras HaySiguiente(it) **hacer** ▷ $\mathcal{O}(L_{max})$
  $actual \leftarrow siguiente(it)$
  avanzar(it)
  si formaVertical **entonces**
    si $actual_1 + 1! = siguiente(it2)_1$ **entonces**
      si $tablero[actual_0, actual_1 + 1].ocupada == false$ **entonces**
        devolver (true,ocurrenciaOrdenada)
      else
        AgregarComoSiguiente(it2, $(actual_0, actual_1 + 1, tablero[actual_0, actuall_1 + 1]$.letra)
  else
    si $actual_0 + 1! = siguiente(it2)_0$ **entonces**
      si $tablero[actual_0 + 1, actual_1].ocupada == false$ **entonces**
        devolver (true,ocurrenciaOrdenada)
      else
        AgregarComoSiguiente(it, $(actual_0 + 1, actual_1$,tablero$[actual_0 + 1, actuall_1]$.letra)
devolver (false,ocurrenciaOrdenada)
Complejidad: $\mathcal{O}(lMax)$
Justificación: Recorre la ocurrencia ordenada, agregando los espacios que hay entre las fichas que ubico el jugador, si la palabra que ubica esta partida y si no hay en el tablero para completar, la jugada es invalida

COMPLETARESPACIOSATRASADELANTE(**in** *ocurrenciaOrdenada*: `lista`,**in** *j*: `juego`,**in** *formaVertical*: `bool` $\longrightarrow$ `tupla(noCumple: bool,ocurrenciaOrdenada: lista)`

si formaVertical **entonces**
  mientras $ocurrenciaOrdenada[0]_1 - 1 \geq 0$ and
    tablero $[ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0]_1 - 1].ocupada$ **hacer** ▷ $\mathcal{O}(L_{max})$
    si $long(ocurrenciaOrdenada > Lmax$ **entonces**
      devolver (true,ocurrenciaOrdenada)
    else
      AgregarAdelante(ocurrenciaOrdenada, $(ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0] - 1,$ $tablero(ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0]_1 - 1)))$
  mientras $ocurrenciaOrdenada[0]_0 + 1 < n$ and
    tablero$[ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0] + 1].ocupada$ **hacer** ▷ $\mathcal{O}(L_{max})$
    si $long(ocurrenciaOrdenada > Lmax$ **entonces**
      devolver (true,ocurrenciaOrdenada)
    else
      AgregarAdelante(ocurrenciaOrdenada, $(ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0] + 1,$ $tablero(ocurrenciaOrdenada[0]_0, ocurrenciaOrdenada[0] + 1)))$
else
  mientras $ocurrenciaOrdenada[0]_0 - 1 \geq 0$ and
    tablero $[ocurrenciaOrdenada[0]_0 - 1, ocurrenciaOrdenada[0]_1].ocupada$ **hacer** ▷ $\mathcal{O}(L_{max})$
    si $long(ocurrenciaOrdenada > Lmax$ **entonces**
      devolver (true,ocurrenciaOrdenada)
    else
      AgregarAdelante(ocurrenciaOrdenada, $(ocurrenciaOrdenada[0]_0 - 1, ocurrenciaOrdenada[0]_1,$ $tablero(ocurrenciaOrdenada[0]_0 - 1, ocurrenciaOrdenada[0]_1)))$
  mientras $ocurrenciaOrdenada[0]_0 + 1 < n$ and
    tablero$[ocurrenciaOrdenada[0]_0 + 1, ocurrenciaOrdenada[0]_1].ocupada$ **hacer** ▷ $\mathcal{O}(L_{max})$
    si $long(ocurrenciaOrdenada > Lmax$ **entonces**
      devolver (true,ocurrenciaOrdenada)
    else
      AgregarAdelante(ocurrenciaOrdenada, $(ocurrenciaOrdenada[0]_1 + 1, ocurrenciaOrdenada[0]_1,$ $tablero(ocurrenciaOrdenada[0]_1, ocurrenciaOrdenada[0]_1)))$
devolver (false,ocurrenciaOrdenada)
Complejidad: $\mathcal{O}(L_{max})$
Justificación: Se agrega los elementos que estaban los tableros previos a la ocurrencia y los que estan luego de que terminan, la longitud nunca supera lMax

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

1: $it \leftarrow crearIt(o)$
2: **mientras** haySiguiente(o) **hacer** ▷ $\mathcal{O}(m)$
3: tablero$[siguiente(o)_0, siguiente(o)_1]$ = (true , $siguiente(it)_2$, estr.turno)
4: $significado(fichasJugadores[turnoDe], siguiente(it_2)) = significado(fichasJugadores[turnoDe], siguiente(it_2)) - 1$
5: $estr.turno = estr.turno + 1$
6: $i \leftarrow 0$ `turnoDe = estr.turno mod long(estr.puntajes)`
7: **mientras** $i < long(o)$ **hacer** ▷ $\mathcal{O}(m)$
8: fichasJugadores[turnoDe][Proximo(estr.repositorio)] = fichasJugadores[turnoDe][Proximo(estr.repositorio)] + 1
9: Desencolar(estr.repositorio)
10: Encolar(puntajePorSumar[turnoDe], (o,estr.turno)

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

Justificación: Se crea una ocurrencia de m elementos, luego se los agrega en un tablero con $\mathcal{O}(1)$ y se modifica la cantidad de fichas que tiene el jugador. Después se agregan m elementos a puntajePendiente del jugador y se le reponen m fichas.

IVARIANTE(**in** j : `Juego`) $\longrightarrow$ `variante`

1: **devolver** estr.variante
=0

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

ITURNO(**in** j : `Juego` $\longrightarrow$) `Nat`

1: **devolver** estr.turno mod long(puntaje)

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

IPUNTAJE(**in** j : `Juego`, **in** cid: `Nat` $\longrightarrow$ `Nat`

1: $puntaje \leftarrow estr.puntaje$
2: $it \leftarrow crearIt(puntajePorSumar[cid])$
3: **mientras** haySiguiente(it) **hacer** ▷ $\mathcal{O}(m \cdot L_{max})$
4: $it2 \leftarrow crearIt(siguiente(it)_0)$
5: avanzar(it2)
6: **si** $anterior(it2)_0 = (siguiente(it2)_0)$ **entonces**
7: $avanzaVertical \leftarrow true$
8: $puntaje = puntaje + puntajePorPalabraFormadaVertical(it2_0, it2_1, it2_2)$ ▷ $\mathcal{O}(lMax)$
9: **else**
10: $puntaje = puntaje + puntajePorPalabraFormadaHorizontal(it2_0, it2_1, it2_2)$ ▷ $\mathcal{O}(lMax)$
11: retroceder(it2)
12: **mientras** haySiguiente(it2) **hacer** ▷ $\mathcal{O}(m)$
13: **si** avanzaVertical **entonces**
14: $puntaje = puntaje + puntajePorLetraVertical(it2_0, it2_1, it2_2)$ ▷ $\mathcal{O}(lMax)$
15: **else**
16: $puntaje = puntaje + puntajePorLetraHorizontal(it2_0, it2_1, it2_2)$ ▷ $\mathcal{O}(lMax)$
$puntajePorSumar[cid] \leftarrow vacio()$
17: **devolver** puntaje

Complejidad: $\mathcal{O}(1 + m * L_{max})$

Justificación: Recorrer el puntaje por sumar del jugador que tiene longitud m fichas, si es un nuevo turno se busca la palabra que se formo y sumando el puntaje de cada letra en particular, accediendo a variante.puntajeFichas O(1). También por cada letra se busca la palabra que esta forma y sumando su puntaje. Se consigue una complejidad de $(1 + m * L_{max})$ porque se accede al puntaje viejo del jugador, se recorren m fichas y por cada ficha se busca la palabra que esta formo, como mucho la longitud de esta en L_{max}

PUNTAJEPORPALABRAFORMADAVERTICAL(**in** i: `int`, **in** j: `int, in j : juego`,**in** $turno$: `nat`) $\longrightarrow$ `variante`

$k \leftarrow j$
mientras $tablero[i,j].turno <=$ turno and tablero$[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
$puntajeParcial+ = variante.puntajeFicha[tablero[i,j]]$
$i--$
mientras $tablero[k,j].turno <=$ turno and tablero$[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
$puntajeParcial+ = variante.puntajeFicha[tablero[i,j]]$
$k--$
devolver puntajeParcial
Complejidad: $\mathcal{O}(L_{max})$
Justificación: Mientras en el tablero haya fichas y estas hayan sido ubicadas previas al turno en el que el jugador ubico, entonces se suma el puntaje de la letra y se sigue recorriendo.
=0

puntajePorPalabraFormadaHorizontal(**in** i: `int`, **in** j: `int, in j : juego`,**in** *turno*: `nat`) $\longrightarrow$ `variante`

1: $k \leftarrow j$
2: **mientras** $tablero[i,j].turno <=$ turno and tablero$[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
3: $\quad puntajeParcial+ = variante.puntajeFicha[tablero[i,j]]$
4: $\quad j--$
5: **mientras** $tablero[i,k].turno <=$ turno and tablero$[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
6: $\quad puntajeParcial+ = variante.puntajeFicha[tablero[i,j]]$
7: $\quad k++$
8: **devolver** puntajeParcial

Complejidad: $\mathcal{O}(L_{max})$
Justificación: Mientras en el tablero haya fichas y estas hayan sido ubicadas previas al turno en el que el jugador ubico, entonces se suma el puntaje de la letra y se sigue recorriendo.

puntajePorLetraVertical(**in** i: `int`, **in** j: `int, in j : juego`,**in** *turno*: `nat`) $\longrightarrow$ `variante`

1: $k \leftarrow j - 1$
2: $j \leftarrow k$
3: **mientras** $tablero[i,j].turno \leq$ turno and $tablero[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
4: $\quad puntajeParcial+ = significado(variante.puntajeFicha, tablero[i,j])$
5: $\quad j--$
$j \leftarrow k + 2$
6: **mientras** $tablero[i,j].turno \leq turno$) **hacer** ▷ $\mathcal{O}(L_{max})$
7: $\quad puntajeParcial+ = significado(variante.puntajeFicha, tablero[i,j])$
8: $\quad j++$
9: **devolver** puntajeParcial

Complejidad: $\mathcal{O}(L_{max})$
Justificación: Recorro de la forma opuesta que se ubicó la palabra para sumar el puntaje que hizo ubicar una letra, es decir ver que palabra forma esta. Como mucho forma una palabra de longitud lMax

puntajePorLetraHorizontal(**in** i: `int`, **in** j: `int, in j : juego`,**in** *turno*: `nat`) $\longrightarrow$ `variante`

1: **puntajePorLetraHorizontal(i,j,turno)**
2: $k \leftarrow i - 1$
3: $i \leftarrow k$
4: **mientras** $tablero[i,j].turno \leq$ turno and $tablero[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
5: $\quad puntajeParcial+ = significado(variante.puntajeFicha, tablero[i,j])$
6: $\quad i--$
$i \leftarrow k + 2$
7: **mientras** $tablero[i,j].turno \leq$ turno and $tablero[i,j]_2 == true$) **hacer** ▷ $\mathcal{O}(L_{max})$
8: $\quad puntajeParcial+ = significado(variante.puntajeFicha, tablero[i,j])$
9: $\quad i++$
10: **devolver** puntajeParcial

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

Justificación: Recorro de la forma opuesta que se ubicó la palabra para sumar el puntaje que hizo ubicar una letra, es decir ver que palabra forma esta. Como mucho forma una palabra de longitud lMax

icontenidoTableroEn(**in** t: `tablero`, **in** i: `Nat`, j:Nat) $\longrightarrow$ `Str`

1: **devolver** estr.tablero[i][j]

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

icantLetra(**in** j: `juego`, **in** *cid*: `Nat`, **in** x: `Str` $\longrightarrow$ `Nat`

1: **devolver** estr.fichasJugadores[cid][x]

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

IFICHASJUGADOR(**in** j : `Juego` **in** cid : `Nat` $\longrightarrow$) `array(nat)`

1: **devolver** estr.variante.fichasJugador[cid]

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

2. Servidor

Interfaz

se explica con: SERVIDOR **géneros**: SERVIDOR operaciones basicas

INICIARSERVIDOR(**in** k : `nat`, **in** v : `variante`, **in** r : `cola(letra)`) $\rightarrow$ res : `juego`
Pre $\equiv$ $\{long(r) \leq tamanoTablero(v) * tamaoTablero(v) + k * fichas(v)\}$
Post$\equiv$ $\{res =_{\text{obs}} nuevoServidor(k, v, r)\}$
Complejidad: O($N^2 + |\sum |K + FK$)
Descripción: Inicializar un servidor
Aliasing: La variante se pasa por referencia no modificable. El repositorio r se pasa por referencia modificable.

CONECTARCLIENTE(**in** s : `servidor`) $\rightarrow$ res : `servidor`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} conectarCliente(s)\}$
Complejidad: O(1)
Descripción: Conectar cliente
Aliasing: El servidor se pasa por referencia modificable.

CONSULARCOLANOTIFICACIONES(**in** s : `servidor`, **in** cid : `nat`) $\rightarrow$ res : `servidor`
Pre $\equiv$ $\{cid < conectados(s)\}$
Post $\equiv$ $\{res =_{\text{obs}} consultar(s, cid)\}$
Complejidad: O(n)
Descripción: Consulta la cola de notificaciones y la vacia
Aliasing: El servidor se pasa por referencia modificable.

RECIBIRMENSAJE(**in** s : `servidor`, **in** cid : `Nat`, **in** o : `ocurrencia`) $\rightarrow$ res : `servidor`
Pre $\equiv$ $\{cid < conectados(s)\}$
Post $\equiv$ $\{res =_{\text{obs}} recibirMensaje(s, cid, o)\}$
Complejidad: No tiene una cola explicita
Descripción: Recibir un mensaje de un jugador
Aliasing: El servidor se pasa por referencia modificable. La ocurrencia se pasa por copia.

CANTESPERADOS(**in** s : `servidor`) $\rightarrow$ res : `Nat`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} esperados(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Obtener el numero de clientes esperados
Aliasing: El servidor se pasa por referencia no modificable.

CANTCONECTADOS(**in** s : `servidor`) $\rightarrow$ res : `Nat`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} conectados(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Obtener el numero de clientes conectados
Aliasing: El servidor se pasa por referencia no modificable.

JUEGOACTUAL(**in** s : `servidor`) $\rightarrow$ res : `juego`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} juego(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Obtener el juego que se está jugando en el servidor
Aliasing: El servidor se pasa por referencia no modificable.

---

iIniciarServidor(**in** k : `Nat`, **in** v : `variante`, r : `cola(letras)` $\longrightarrow$ `servidor`

1: $estr.juego \leftarrow crearJuego(k, v, r)$ ▷ $\mathcal{O}(N^2 + |\sum|K + FK)$
2: $estr.notificaciones \leftarrow array[k](cola(notificacion))$ ▷ $\mathcal{O}(K)$
3: $estr.esperados \leftarrow k$
4: $estr.conectados \leftarrow 0$
5: $estr.configuracion \leftarrow (v, r)$
6: $estr.consultoEmpezar \leftarrow array[k]deFalse$ ▷ $\mathcal{O}(k)$
7: **devolver** ESTR

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

Justificación: Se crea un juego, se crea la cola de notificaciones de los clientes (array de long k) y se asignan los valores a la estructura.

---

iconectarCliente(**inout** s : `servidor`) $\longrightarrow$ `servidor`

1: $estr.conectados \leftarrow estr.conectados + 1$
2: $estr.notificaciones[estr.conectados - 1] \leftarrow conectados$

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

---

iconsultarColaNotificaciones(**in** s : `servidor`, **in** cid : `nat`) $\longrightarrow$ `lista`

$notDevueltas \leftarrow lista$

si $(esVacia?(s.notificaciones[cid]) == false)$ **entonces**

$agregarAtras(notDevueltas, proximo(s.notificaciones[cid]))$

$desencolar(s.notificaciones[cid])$

si estr.conectados == estr.esperados **entonces**

si estr.consultoEmpezar[cid] == False **entonces**

agregarAtras(notDevueltas,Empezar[estr.variante.tamaño])

$estr.consultoEmpezar[cid] \leftarrow True$

mientras $(esVacia?(s.notificaciones[cid]) == false)$ **hacer** ▷ $\mathcal{O}(n)$

$agregarAtras(notDevueltas, proximo(s.notificaciones[cid]))$

$desencolar(s.notificaciones[cid])$

devolver notDevueltas

Complejidad: $\mathcal{O}(n)$, donde es |s.notificaciones[cid]|

Justificación: Se recorre la cola de notificaciones del cliente O(n), mientras se hace esto se agrega la notificacion a una lista y se la borra de la cola.

---

icantEsperados(**in** s : `servidor`) $\longrightarrow$ `nat`

devolver estr.esperados

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

---

icantConectados(**in** s : `servidor`) $\longrightarrow$ `nat`

devolver estr.conectados

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

IJUEGOACTUAL(**in** s : `servidor`) $\longrightarrow$ `juego`

devolver estr.juego

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

IRECIBIRMENSAJE(**in** s : `servidor`, **in** cid : `nat`,**in** o : `ocurrencia`) $\longrightarrow$ `servidor`

```
1:  para int i = 0; i <esperados hacer
2:      si conectados = esperados entonces
3:          encolar(estr.notificacion[i], Empezar)
4:          si jugadaValida?(cid,o) entonces                                  ▷ O(L²_max)
5:              encolar(estr.notificaciones[cid],ubicar(cid,o)
6:              puntajeViejo ← juego.puntaje(id)
7:              puntajeViejo = puntaje(estr.juego,cid)
8:              fichasViejas = (fichasJugadores(estr.juego, [cid]))
9:              ubicar(estr.juego,o)                                          ▷ O(long(o))
10:             encolar(estr.notificacion[i], notif(SumaPuntos(cid,puntaje(estr.juego,cid)- puntajeViejo))
11:             encolar(estr.notificacion[i],notif(Reponer(fichasRepuestas(s,long(o),cid,fichasJugadores(estr.juego,
    [cid])), fichasViejas))                                                   ▷ O(F)
12:             encolar(estr.notificacion[i],Ubicar(cid,o))
13:             encolar(TurnoDe(turno(estr.juego)))
14:         else
15:             encolar(estr.notificacion[i],Mal)
16:     else
17:         encolar(estr.notificacion[i],Turno(0))
```

Complejidad: $\mathcal{O}(L^2_{max}+)$

Justificación: Chequear si el juego empezo, y si la jugada que quiere hacer el jugador es valida. Si lo es se agregan las notificaciones correspondientes y se modifica el juego ubicando la palabra.

FICHASREPUESTAS(**in** s : `servidor`,**in** m : `nat`,**in** cid : `nat`, **in** $fichasViejas$: `array[nat]` $\longrightarrow$ `array`

```
1: crear array estatico fichas de nat de long m
2: para int i = 0; i < long(estr.juego.fichasJugador[0]); i++ hacer
3:     para int j = 0; j <(estr.juego.fichasJugador[cid][i] -fichasViejas[i]); i++ hacer
4:         fichas[i+j] = i
   =0
```

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

Justificación: Recorre un array de longitud F donde f es la cantidad de letras del alfabeto Compara las fichas nuevas que tiene el jugador con las que tenía antes de jugar y agrega aquellas que no tenía a una lista de longitud m donde m es la cantidad de letras que jugó

Representación

Servidor se representa con **estr** donde

estr es tupla(esperados: nat, conectados: nat, configuracion : tupla(v: variante, r: cola(letra)), notificaciones: array(cola(notificación)),juego: juego, consultoEmpezar: array(bool))

donde variante se representa como tupla(tamaño: Nat, cantFichas: Nat, puntajeFichas: array(nat), palabras: conj(palabra))

donde letra se representa como nat y palabra como array(letra)

donde notificacion es enum(IdCliente, Empezar, TurnoDe, Ubicar, Reponer, SumaPuntos)

Invariante de representación

- La cantidad de conectados es menor igual a la cantidad de esperados

- La longitud de notificaciones coincide con la cantidad de esperados.
- Todas las letras del repositorio pertenecen al alfabeto de la variante.
- La cantidad de esperados tiene que coincidir con la cantidad de jugadores del juego dado.
- La longitud de consultoEmpezar es igual a la cantidad de esperados.

$\forall e : estr$

Funcion de abstracción

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

Abs(e) $\equiv$ s: servidor | #esperados(s) = e.esperados $\wedge$
#conectados(s) = e.conectados $\wedge$
configuración(s) = e.configuración $\wedge$
juego(s) = e.juego $\wedge$
$(\forall i : nat)(0 \leq$ i $<$#conectados $\longrightarrow$ notificaciones(s,i) = e.notificaciones[i])
(tomamos notificaciones como cola(notif) en lugar de secu(noti) como dice en la especificacion)
e.conectados $<$e.esperados $\longrightarrow$ $(\forall i : nat)(0 \leq$ i $<\#esperados \longrightarrow$ consultoEmpezar[i] == False)

3. Conjunto Digital

Interfaz

parámetros formales
géneros σ
se explica con: CONJUNTO
géneros: `conjDigital`.

Operaciones básicas de Conjunto

VACIO() $\rightarrow$ *res* : `conj(`α`)`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}}\}$
Complejidad: O(1)
Descripción: Crea un conjunto vacio

AGREGAR(**in/out** c: `conj(`α`)`, **in** $a : \alpha$)
Pre $\equiv \{c = c_0\}$
Post $\equiv \{c =_{\text{obs}} \text{Ag}(a, c_0)\}$
Complejidad: $|k|$ donde k es la longitud de la clave
Descripción: el elemento α se agrega por copia

PERTENCE(**in/out** c: `conj(`α`)`, **in** $a : \alpha$) $\rightarrow$ *res* : `bool`
Pre $\equiv \{long(a) > 0\}$
Post $\equiv \{\text{res} =_{\text{obs}} \text{a} \in \text{c}\}$
Complejidad: $|k|$ donde k es la longitud de la clave
Descripción: devuelve true si y solo si a pertenece al conjunto

LMAX(**in** c: `conj(`α`)`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{\text{res} \geq 0\}$
Complejidad: 1
Descripción: longitud de la palabra más larga en el conjunto

Representación

Representación del conjDigital

El conjDigital se representa un arreglo de longitud cantidad de letras en el alfabeto + 1 y su contenido es una tupla de si es hoja el elemento actual y una direccion de memoria a un arreglo de la misma longitud, también tiene una constante que representa la longitud maxima de todas las palabras en el conjunto

`conjDigital`(κ, σ) **se representa con** `conj`

donde `conj` es `tupla`(*trie*: `arreglo(tupla(esPalabra: bool, siguiente: puntero(nodo))`, *LMax*: `Nat`)

donde nodo es arreglo(tupla(esPalabra:bool, siguiente:puntero(nodo)))

Invariante de representación

- La longitud de la rama mas larga tiene que ser igual a LMáx
- No hay nodo inutil, es decir recorriendo para abajo, no existe una palabra
- La longitud de los array de todos los nodos son iguales

Abs : conj c $\longrightarrow$ conj(σ) {Rep(c)}

Abs(c) $\equiv$ Si esPalabra es true, entonces subir por el trie para ver que palabra forma y esta debe pertenecer a conj
Toda la palabra que pertenece en conj entonces pertenece al conjDigital

iVacío() $\rightarrow$ *res* : conj(α)

res $\leftarrow$ $\langle$ secu::Vacia(), 0$\rangle$

Complejidad: $\Theta(1)$

---

iAgregar(**in/out** *c*: `conj(`α`)`, **in** *key*: `str`)

```
Trie* actual = estr.trie;
i ← 0
mientras i< key hacer                                          ▷ O(|K|)
    si actual.siguiente[key[i]] ==nullptr entonces
        actual.siguiente[key[i]] =new Trie()
    actual = actual.siguiente[key[i]];
    i++;
actual.esPalabra = true;
si long(a) > estr.LMax entonces
    estr.LMax = long(a)
```

Complejidad: $\Theta(|k|)$

---

iPertenece(**in/out** *c*: `conj(`α`)`, **in** *key*: α) *to res*: bool

```
Trie* actual = estr.trie
para i = 0, i < long(key) hacer                                ▷ O(|k|)
    actual = actual.siguiente[key[i]]
    if (actual == null)
        return false
res ← actual.esPalabra
```

Complejidad: $\Theta(|k|)$

---

iLMax(**in** *c*: `conj(`α`)`) *to res*: Nat

```
1: res ← estr.lMax =0
```