Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

JuegoDePalabras

Grupo: PepitoCodea

Integrante	LU	Correo electrónico
Ponce, Ezequiel	730/21	ezequielponcePe11@gmail.com
Rodriguez Sanudo, Ignacio	956/21	rodriguezsanudoignacio@ gmail.com
Horn, Manuel	321/21	manuhorn1910@gmail.com
Antonio, Cristian Ignacio	619/14	cristianantonio57@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Módulos

1.1. Módulo Juego

Interfaz

se explica con: JUEGO

géneros: `juego`.

Operaciones básicas

NUMDEJUGADORES(**in** j : `juego`) $\rightarrow$ res : `Nat`
Pre $\equiv$ {true}
Post $\equiv$ {$\widehat{res} =_{\text{obs}}$ NumDeJugadores($\widehat{j}$)}
Complejidad: $\Theta(1)$
Descripción: Devuelve la cantidad de jugadores.
Aliasing: es retornado por referencia constante

NUEVOJUEGO(**in** k: `nat`, **in** v: `variante`, **in** rep: `cola(letra)`) $\rightarrow$ res : `juego`
Pre $\equiv$ {$tamano(\widehat{rep}) \geq tamanoTablero(\widehat{v}) * tamanoTablero(\widehat{v})k * numDeFichas(\widehat{v}) \wedge \widehat{k} > 0$}
Post $\equiv$ {$\widehat{res} =_{\text{obs}} nuevoJuego(\widehat{k}, \widehat{v}, \widehat{rep})$}
Complejidad: $\Theta(\text{N}^2 + \text{k*F} + \Sigma * k)$
Descripción: Devuelve una nueva instancia del juego donde N^2 es la cantidad de casilleros en el tablero, k la cantidad de jugadores y Σ la cantidad de letras en el alfabeto

UBICAR(**in/out** j : `juego`, **in** o: `ocurrencia`)
Pre $\equiv$ {$j =_{\text{obs}} j_0 \wedge jugadaValida?(\widehat{j}, \widehat{o})$}
Post $\equiv$ {fichasUbicadas($\widehat{j}, \widehat{o}$)}
Complejidad:$\Theta(m)$
Descripción: m es la cantidad de fichas siendo ubicadas, ubicar ubica las fichas

JUGADAVALIDA?(**in** j : `juego`, **in** o: `ocurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv$ {$True$}
Post $\equiv$ {$\widehat{res} =_{\text{obs}} jugadaValida?(\widehat{j}, \widehat{o})$}
Complejidad:$\Theta(\text{L}^2_{max})$
Descripción: Chequea si una jugada es valida

VARIANTE(**in** j : `juego`) $\rightarrow$ res : `variante`
Pre $\equiv$ {$True$}
Post $\equiv$ {$\widehat{res} =_{\text{obs}} variante(\widehat{j})$}
Complejidad:$\Theta(1)$
Descripción: retorna la variante
Aliasing: es devuelto como referencia inmodificable

OBTENERTURNO(**in** j : `juego`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$True$}
Post $\equiv$ {$\widehat{res} =_{\text{obs}} turno(\widehat{j})$}
Complejidad:$\Theta(1)$
Descripción: retorna el turno
Aliasing: lo hace como referencia inmodificable

OBTENERPUNTAJE(**in/out** j : `juego`,**in** i : `jugador`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$i < k$}
Post $\equiv$ {$\widehat{res} =_{\text{obs}} puntaje(j, i)$}
Complejidad:$\Theta(1 + m * L_{max})$
Descripción: retorna el puntaje de la jugada. m es la cantidad de fichas ubicadas
Aliasing: el puntaje se pasa por copia

FICHAENPOSICION(**in** t : `tablero`,**in** i : `nat`,**in** j : `nat`) $\rightarrow$ res : `ficha`
Pre $\equiv \{i < tamanoTablero(v) \land j < tamanoTablero(v) \land hayLetra?(i,j)\}$
Post $\equiv \{\widehat{res} =_{\text{obs}} letra(t,i,j)\}$
Complejidad:$\Theta(1)$
Descripción: retorna la letra que se ubica en el tablero en la posicion pasada por parametro

HAYFICHA(**in** t : `tablero`,**in** i : `nat`,**in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{i < tamanoTablero(v) \land j < tamanoTablero(v)\}$
Post $\equiv \{\widehat{res} =_{\text{obs}} \pi_2(tablero[i][j])\}$
Complejidad:$\Theta(1)$
Descripción: Devuelve true si una ficha fue puesta en esa posición, de lo contrario false

CUANTOTIENEDEESTAFICHA(**in** x : `Letra`,**in** j : `juego`,**in** i : `jugador`) $\rightarrow$ res : `nat`
Pre $\equiv \{True\}$
Post $\equiv \{\widehat{res} =_{\text{obs}} \#(x, fichas(j,i))\}$
Complejidad:$\Theta(1)$
Descripción: retorna la cantidad de una ficha pasada como parametro tiene el jugador
Aliasing: se pasa por copia inmodificable

1.2. Representación de Juego

Representación

Representación del juego

Un juego es una tupla donde se guarda:
el repositorio
el tablero
la variante
el numero de jugadores
el turno

`juego` **se representa con** `estr`

donde `estr` es `tupla`(*Repositorio*: `Cola(Letra)`, *Tablero*: `Array(Array(Tupla<char,Turno,bool>))`, *Variante*: `Variante`, *numDeJugadores*: `nat`, *Jugadores*: `Array(tupla(Mano:Array(nat),Puntaje:nat, FichasQuePusoDesde:Ocurrencia))`, *Turno*: `nat`)

donde `Variante` es `tupla`(*palabras*: $\text{Conj}_{Digital}(palabra : array(letra))$, ***CantDeFichas*** : `Nat`, ***Abecedario*** : `Array(valor)`, ***tamañoTablero*** : `nat`, l_{max} : `nat`)

Ocurrencia es $\text{Conjunto}_{lineal}(tupla <$ *filaJugada* : `nat`, *columnaJugada* : `nat`, *letra* : `char` $>)$

Valor es Nat

Rep : estr $\longrightarrow$ bool

Rep(j) $\equiv$ true $\iff$ **tamaño(j.repositorio) > 0** $\wedge$
j.numDeJugadores = longitud(Jugadores) $\wedge$
Toda letra i en el repositorio se corresponde inyectivamente a una posicion j del abecedario tal que abecedario[j] = puntaje(i)$\wedge$
Para cada jugador para cada indice i en el vector mano su valor se corresponde con la cantidad de fichas que tiene de esa letra es de decir ord^{-1}(i)$\wedge$
La sumatoria de los valores en el vector mano es igual a K (la cantidad de fichas que tiene en cada jugador) $\wedge$
N = j.variante.tamañoTablero $\wedge$
El tablero es una matriz cuadrada de NxN $\wedge$
La sumatoria de los valores en el vector mano es igual a K (la cantidad de fichas que tiene en cada jugadpr) $\wedge$
(para todo i,j en el rango tablero)(π_3(j.Tablero[i][j]) = True $\Rightarrow$ ((π_2(j.Tablero[i][j]) < j.Turno) $\wedge$**(π_1 existe en el abecedario)))** $\wedge$
Las fichas que están en FichasQuePusoDesde (si las hay) tienen que ser válidas, estar en el tablero, y tienen que corresponder al turno de un jugador en particular $\wedge$**Todas las fichas en el tablero forman palabras válidas** $\wedge$
Todas las letras en la ocurrencia pertenecen al abecedario

1.3. Abstraccion del Juego

Abs : estr j $\longrightarrow$ juego {Rep(j)}

Abs(j) $\equiv$ $j : juego | j.variente =_{\text{obs}} variante(j) \wedge$
$j.numDeJugadores =_{\text{obs}} \#\text{jugadores} \wedge$
$j.repositorio =_{\text{obs}} repositorio(j) \wedge$
$long(j.Tablero) =_{\text{obs}} tamanio(Tablero(j)) \wedge$

$(\forall i : nat)(0 \leq i < long(j.Jugadores) \Rightarrow_{\text{L}}$
$(\forall j : nat)(0 \leq j < long(\pi_1(j.Jugadores[i]) \Rightarrow_{\text{L}}$
$(\pi_1(j.Jugadores[i])[j] = \#(ord^{-1}(j), \text{fichas[j,i]})) \wedge$**(puntaje(j,id) es el puntaje que está guardado en la estructura mas el puntaje de las palabras formadas por las fichas que el jugador tiene en la tupla FichasQuePusoDesde)**

1.4. Algoritmos de Juego

Algoritmos

---

iCuantoTieneDeEstaFicha(**in** x : `Letra`,**in** j : `juego`,**in** i : `jugador`) $\rightarrow$ res : Nat

1: $res \leftarrow \pi_1(j.Jugadores[i])[ord(x)]$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

---

---

iNumDeJugadores(**in** j : `juego`) $\rightarrow$ res : Nat

1: $res \leftarrow j.NumDeJugadores$

Complejidad: $\Theta(1)$
Justificacion: Es obtener un valor de una tupla

---

iVariante(**in** j : `juego`) $\rightarrow$ res : Nat

1: $res \leftarrow j.Variante$

Complejidad: $\Theta(1)$
Justificacion: Es obtener un valor de una tupla
Aliassing: devuelve el valor por referencia inmutable

iNuevoJuego(**in** k : `Nat` **in** v : `Variante` **in** rep : `Cola(Letra)`) $\rightarrow$ res : juego

1: $tablero \leftarrow CrearTablero(v.TamanoTablero)$
2: $jugadores \leftarrow RepartirFichas(v, k, rep)$
3: $res \leftarrow < rep, tablero, v, k, jugadores, 0 >$

Complejidad: $\Theta(N^2 + k*F + \Sigma * k)$
Justificacion: Crea un tablero de N^2 con los valores correspondientes y las estructuras necesarias para el juego.

iUbicar(**in/out** j : `Juego` **in** o : `Ocurrencia`) $\rightarrow$ res : juego

1: $itConj \leftarrow CrearIt(o)$
2: **mientras** haySiguiente(o) **hacer**
3: $\quad \pi_1(j.tablero[\pi_1(Siguiente(o))][\pi_2(Siguiente(o))]) \leftarrow \pi_3(Siguiente(o))$
4: $\quad j.Jugadores.Mano[ord(\pi_3(Siguiente(o)))] - -$
5: $\quad \pi_2(j.tablero[\pi_1(Siguiente(o))][\pi_2(Siguiente(o))]) \leftarrow j.Turno$
6: $\quad \pi_3(j.tablero[\pi_1(Siguiente(o))][\pi_2(Siguiente(o))]) \leftarrow True$
7: $\quad AgregarRapido(\pi_3(j.Jugadores[iObtenerTurno(j)]), Siguiente(o))$
8: $\quad Avanzar(itConj)$
9: $j.turno++$

Complejidad: $\Theta(m)$
Justificacion: Para cada elemento de la ocurrencia ubica la ficha y asigna valores a la tupla del tablero.

iObtenerTurno(**in** j : `juego`) $\rightarrow$ res : Nat

1: $res \leftarrow j.Turno \mod j.NumDeJugadores$

Complejidad: $\Theta(1)$
Justificacion: Obtener valores previamente guardados

iObtenerPuntaje(**in/out** j : `juego` **in** i : `jugador`) $\rightarrow$ res : Nat

1: $conjPalabras \leftarrow ConjDigitalVacio()$
2: $itConj \leftarrow crearIt(j.Jugadores[i].FichasQueJugoDesde)$
3: **mientras** ($haySiguiente(itConj)$) **hacer**
4: $prim \leftarrow < \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj)) >$
5: $ult \leftarrow < \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj)) >$
6: $arr \leftarrow < \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj)) >$
7: $abj \leftarrow < \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj)) >$
8: $turno \leftarrow j.tablero[\pi_1(prim)][\pi_2(prim)].Turno$
9: **mientras** $(('enRango' \wedge \pi_3(j.tablero[\pi_1(arr) - 1][\pi_2(arr)]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(arr) - 1][\pi_2(arr)]) \vee ('enRango' \wedge \pi_3(j.tablero[\pi_1(abj)+1][\pi_2(abj)]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(abj)+1][\pi_2(abj)]))$ **hacer**
10: **si** $\pi_3(j.tablero[\pi_1(arr) - 1][\pi_2(arr)] == true \wedge turno \geq \pi_2(j.tablero[\pi_1(arr) - 1][\pi_2(arr)])$ **entonces**
11: $\pi_1(arr) \leftarrow \pi_1(arr) - 1$
12: **si** $\pi_3(j.tablero[\pi_1(abj) + 1][\pi_2(abj)]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(abj) + 1][\pi_2(abj)])$ **entonces**
13: $\pi_1(abj) \leftarrow \pi_1(abj) + 1$
14: **mientras** $(('enRango' \wedge \pi_3(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]) \vee ('enRango' \wedge \pi_3(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]))$ **hacer**
15: **si** $\pi_3(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]$ **entonces**
16: $\pi_1(prim) \leftarrow \pi_1(prim) - 1$
17: **si** $\pi_3(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]) == true \wedge turno \geq \pi_2(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]$ **entonces**
18: $\pi_1(ult) \leftarrow \pi_1(ult) + 1$
19: $AgregarPalabraHorizontal(conjPalabras, prim, ult, vacio, j.tablero)$
20: $AgregarPalabraVertical(conjPalabras, arr, abj, vacio, j.tablero)$
21: $j.Jugadores[i].FichasQueJugoDesde \leftarrow Vacio()$
22: $res \leftarrow PuntajeTotal(conjPalabras) + j.Jugadores[i].Puntaje$
23: $j.Jugadores[i].Puntaje \leftarrow res$

Complejidad: Θ(1+m*Lmax)
Justificacion: m = FichasQueJugoDesde. Para cada ficha hace barridos horizontal y vertical para buscar las palabras que forman y las guarda en un conjunto. Después para cada palabra de este conjunto calcula su puntaje y lo suma al puntaje previamente almacenado. Luego actualiza la información de el jugador.

---

iFichaEnPosicion(**in** t : `tablero`,**in** i : `nat`,**in** j : `nat`) $\rightarrow$ res : Letra

1: $res \leftarrow \pi_1(t[i][j])$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

---

iCuantoTieneDeEstaFicha(**in** x : `Letra`,**in** j : `juego`,**in** i : `jugador`) $\rightarrow$ res : Nat

1: $res \leftarrow \pi_1(j.Jugadores[i])[ord(x)]$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

iJugadaValida?(**in** j : `juego`, **in** o : `ocurrencia`) $\rightarrow$ res : bool

1: **si** $cardinal(o) > Lmax$ **entonces**
2: $res \leftarrow false$
3: **return** res
4: $itConj \leftarrow crearIt(o)$
5: $conjPosiblesPalabras \leftarrow Vacio()$
6: $horizontal \leftarrow ihorizontal(o)$
7: $vertical \leftarrow ivertical(o)$
8: $n \leftarrow j.variante.TamanoTablero$
9: $prim \leftarrow < 0, n-1 >$
10: $ult \leftarrow < 0, 0 >$
11: $arr \leftarrow < n-1, 0 >$
12: $abj \leftarrow < 0, 0 >$
13: **si** $\neg$(horizontal $\vee vertical$) **entonces**
14: $res \leftarrow false$
15: **return** res
16: **mientras** $(haySiguiente(itConj) \wedge (\pi_2(ult) - \pi_2(prim) < Lmax) \wedge (\pi_1(abj) - \pi_1(arr) < Lmax)$ **hacer**
17: **si** $(\pi_3(tablero[\pi_1(siguiente(itConj))][\pi_2(siguiente(itConj))] == true)$ **entonces**
18: $res \leftarrow false$
19: **return** res
20: **si** horizontal **entonces**
21: $prim \leftarrow < \pi_1(siguiente(itConj), min(\pi_2(prim), \pi_2(siguiente(itConj)) >$
22: $ult \leftarrow < \pi_1(siguiente(itConj), max(\pi_2(ult), \pi_2(siguiente(itConj)) >$
23: $arr \leftarrow < \pi_1(siguiente(itConj), \pi_2(siguiente(itConj) >$
24: $abj \leftarrow < \pi_1(siguiente(itConj), \pi_2(siguiente(itConj) >$
25: **else**
26: $prim \leftarrow < \pi_1(siguiente(itConj), \pi_2(siguiente(itConj) >$
27: $ult \leftarrow < \pi_1(siguiente(itConj), \pi_2(siguiente(itConj) >$
28: $arr \leftarrow < min(\pi_1(arr), \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj) >$
29: $abj \leftarrow < max(\pi_1(abj), \pi_1(siguiente(itConj)), \pi_2(siguiente(itConj) >$
30: **mientras** $((\pi_3(j.tablero[\pi_1(arr) - 1][\pi_2(arr)]) == true \vee \pi_3(j.tablero[\pi_1(abj) + 1][\pi_2(abj)]) == true) \wedge (\pi_1(abj) - \pi_1(arr) < Lmax)$ **hacer**
31: **si** $\pi_3(j.tablero[\pi_1(arr) - 1][\pi_2(arr)]) == true$ **entonces**
32: $\pi_1(arr) \leftarrow \pi_1(arr) - 1$
33: **si** $\pi_3(j.tablero[\pi_1(abj) + 1][\pi_2(abj)]) == true$ **entonces**
34: $\pi_1(abj) \leftarrow \pi_1(abj) + 1$
35: **si** $horizontal$ **entonces**
36: $AgregarPalabraVertical(conjPosiblesPalabras, arr, abj, conj(siguiente(itConj)), j.tablero)$
37: **mientras** $((\pi_3(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]) == true \vee \pi_3(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]) == true) \wedge (\pi_1(ult) - \pi_1(prim) < Lmax)$ **hacer**
38: **si** $\pi_3(j.tablero[\pi_1(prim)][\pi_2(prim) - 1]) == true$ **entonces**
39: $\pi_1(arr) \leftarrow \pi_1(arr) - 1$
40: **si** $\pi_3(j.tablero[\pi_1(ult)][\pi_2(ult) + 1]) == true$ **entonces**
41: $\pi_1(abj) \leftarrow \pi_1(abj) + 1$
42: **si** $vertical$ **entonces**
43: $AgregarPalabraHorizontal(conjPosiblesPalabras, prim, ult, conj(siguiente(itConj)), j.tablero)$
44: **si** $horizontal$ **entonces**
45: **si** $SonContiguas(prim, ult, horizontal, o)$ **entonces**
46: $AgregarPalabraHorizontal(conjPosiblesPalabras, prim, ult, o, j.tablero)$
47: **else**
48: $res \leftarrow false$
49: **return** res
50: **else**
51: **si** $SonContiguas(arr, abj, horizontal, o)$ **entonces**
52: $AgregarPalabraVertical(conjPosiblesPalabras, arr, abj, o, j.tablero)$
53: **else**
54: $res \leftarrow false$
55: **return** res
56: **si** $(\pi_2(ult) - \pi_2(prim) == Lmax) \vee (\pi_1(abj) - \pi_1(arr) == Lmax)$ **entonces**
57: $res \leftarrow false$

1.5. Funciones Auxiliares de Juego

iCrearTablero(**in** n : `Nat`) $\rightarrow$ res : tablero

1: $tablero \leftarrow crearArreglo(crearArreglo(tupla(\text{""}, 0, False), n), n)$

Complejidad: $\Theta(N^2)$
$Justificacion$: Es crear una matriz de NXN

iRepartirFichas(**in** v : `variante`,**in** k : `Nat`,**in** rep : `cola(letra)`) $\rightarrow$ res : Jugadores

1: $mano \leftarrow crearArreglo(0, long(v.abecedario))$
2: $puntaje \leftarrow 0$
3: $FichasQuePusoDesde : ConjLineal \leftarrow \emptyset$
4: $Jugadores \leftarrow crearArreglo(tupla(mano : mano, puntaje : 0, FichasQuePusoDesde : FichasQuePusoDesde), k)$
5: $jugRepartidos \leftarrow 0$
6: **mientras** $jugRepartidos < k$ **hacer**
7: $FichasRepartidas \leftarrow 0$
8: **mientras** $FichasRepartidas < v.CantDeFichas$ **hacer**
9: ficha = proximo(rep)
10: desencolar(rep)
11: Jugadores[jugRepartidos].mano[ord(ficha)]++
12: FichasRepartidas ++
13: jugRepartidos ++
14: $res \leftarrow Jugadores$

Complejidad: $\Theta($ k*F + $\Sigma * k)$

Justificacion: El ciclo while se recorre k veces y para cada iteracion se recorre F veces, para cada jugador se crea un arreglo con longitud del abecedario.

SonValidas(**in** j : `juego`, **in** ps : `conj(palabra)`) $\rightarrow$ res : bool

1: $itConj \leftarrow crearIt(ps)$
2: **mientras** $(haySiguiente(itConj))$ **hacer**
3: **si** $(\neg Pertenece(j.variante.palabras, siguiente(itConj)))$ **entonces**
4: $res \leftarrow false$
5: return
6: $res \leftarrow true$

Complejidad: Θ(long(ps)*Lmax)
Justificacion: Para cada palabra ver si la palabra pertenece al Conjunto de todas las palabras válidas

iHayFicha(**in** t : `tablero`,**in** i : `nat`,**in** j : `nat`) $\rightarrow$ res : bool

1: $res \leftarrow \pi_3(t[i][j])$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

AgregarPalabraHorizontal(**in/out** c: `conj(secu(letras))`, **in** $inicio$: `tupla<nat,nat>`, **in** fin: `tupla<nat,nat>`), **in** o: `ocurrencia`, **in** t: `tablero`)

```
1:  palabra ← Vacia()
2:  itConj ← CrearIt(MergeSortSegundaComponente(o))
3:  mientras (π2(inicio) <= π2(fin)) hacer
4:      si (π3(t[π1(inicio)][π2(inicio)])) == true entonces
5:          AgregarAtras(palabra, π1(t[π1(inicio)][π2(inicio)]))
6:      else
7:          AgregarAtras(palabra, π3(siguiente(itConj)))
8:          Avanzar(itConj)
9:      π2(inicio) ← π2(inicio) + 1
10: Agregar(palabra,c)
```

Complejidad: Θ(Lmax*Log(Lmax))
Justificacion: Luego del ordenamiento, se itera a lo sumo Lmax veces. Las operaciones con la palabra son O(1).

AgregarPalabraVertical(**in/out** c: `conj(secu(letras))`, **in** $inicio$: `tupla<nat,nat>`, **in** fin: `tupla<nat,nat>`), **in** o: `ocurrencia`, **in** t: `tablero`)

```
1:  palabra ← Vacia()
2:  itConj ← CrearIt(MergeSortPrimeraComponente(o))
3:  mientras (π1(inicio) <= π1(fin)) hacer
4:      si (π3(t[π1(inicio)][π2(inicio)])) == true entonces
5:          AgregarAtras(palabra, π1(t[π1(inicio)][π2(inicio)]))
6:      else
7:          AgregarAtras(palabra, π3(siguiente(itConj)))
8:          Avanzar(itConj)
9:      π1(inicio) ← π1(inicio) + 1
10: Agregar(palabra,c)
```

Complejidad: Θ(Lmax*Log(Lmax))
Justificacion: Luego del ordenamiento, se itera a lo sumo Lmax veces. Las operaciones con la palabra son O(1).

SonContiguas(**in** $inicio$: `tupla<nat,nat>`, **in** fin: `tupla<nat,nat>`, **in** $esHorizontal$: `bool`, **in** o: `ocurrencia`, **in** t: `tablero`) → res : bool

```
1:  i ← 0
2:  si esHorizontal entonces
3:      mientras (π2(inicio) <= π2(fin)) hacer
4:          si (π3(t[π1(inicio)][π2(inicio)])) == false entonces
5:              i ← i + 1
6:          π2(inicio) ← π2(inicio) + 1
7:  else
8:      mientras (π1(inicio) <= π1(fin)) hacer
9:          si (π3(t[π1(inicio)][π2(inicio)])) == false entonces
10:             i ← i + 1
11:         π1(inicio) ← π1(inicio) + 1
12: res ← i == tamano(o)
```

Complejidad: Θ(Lmax)
Justificacion: Verifica si la jugada no deja espacios blancos en el medio de la posible palabra.

ihorizontal(in o : ocurrencia) $\rightarrow$ res : bool

```
1:  res ← true
2:  itConj ← CrearIt(o)
3:  si Vacio?(o) entonces
4:      True
5:  else
6:      p ← π1[Siguiente(itConj)]
7:      mientras haySiguiente(itConj) hacer
8:          si (π1[siguiente(itConj)] ! = p) entonces
9:              res ← false
10:         Avanzar(itConj)
11: return res
```

Complejidad: $\Theta(|o|)$
Justificacion: Chequea si la palabra formada es horizontal

ivertical(in o : ocurrencia) $\rightarrow$ res : bool

```
1:  res ← true
2:  itConj ← CrearIt(o)
3:  si Vacio?(o) entonces
4:
5:      returnres
6:  else
7:      p ← π2[Siguiente(itConj)]
8:      mientras haySiguiente(itConj) hacer
9:          si (π2[siguiente(itConj)] ! = p) entonces
10:             res ← false
11:         Avanzar(itConj)
12: return res
```

Complejidad: $\Theta(|o|)$
Justificacion: Chequea si la palabra formada es vertical

iPuntajetotal(in J : juego, in C : Conjlineal(Palabra)) $\rightarrow$ res : nat

```
1:  res ← 0
2:  itConj ← CrearIt(C)
3:  mientras haySiguiente(itConj) hacer
4:      j ← 0
5:      mientras j < siguiente(itConj).longitud hacer
6:          res ← res + J.Variante.Abecedario[ord(siguiente(itConj)[j])]
7:          j + +
8:      Avanzar(itConj)
9:  return res
```

Complejidad: $\Theta(L^2_{max})$
10: Justificacion: Puntaje obtenido de sumar el puntaje de todas las letras de un conjunto de palabras

1.6. Módulo servidor

Interfaz

se explica con: SERVIDOR

géneros: servidor.

Operaciones básicas

INCIALIZARSERVIDOR(**in** k: `nat`, **in** v: `variante`, **in** rep: `cola(letra)`) $\rightarrow$ res : `servidor`
Pre $\equiv$ $\{tamano(\widehat{r}) \geq tamanoTablero(\widehat{v}) * tamanoTablero(\widehat{v})k * numDeFichas(\widehat{v}) \wedge \widehat{k} > 0\}$
Post $\equiv$ $\{\widehat{res} =_{\text{obs}} nuevoServidor(\widehat{k}, \widehat{v}, \widehat{rep})\}$
Complejidad: $\Theta(\text{N}^2 + \text{k*F} + \Sigma * k)$
Descripción: Devuelve una nueva instancia del servidor donde N^2 es la cantidad de casilleros en el tablero, k la cantidad de jugadores y Σ la cantidad de letras en el alfabeto

CONECTARCLIENTE(**in/out** s: `servidor`)
Pre $\equiv$ $\{\widehat{s} =_{\text{obs}} \widehat{s_0} \wedge \neg empezo?(\widehat{s})\}$
Post $\equiv$ $\{\widehat{s} = jugadorConectado(\widehat{s_0})\}$
Complejidad:$\Theta(1)$
Descripción: agrega un jugador al vector de jugadores y le otorga un id

CONSULTARNOTIFICACIONES(**in/out** s: `servidor`,**in** j: `jugador`)
Pre $\equiv$ $\{\widehat{s} =_{\text{obs}} \widehat{s_0}\}$
Post $\equiv$ $\{\widehat{s} =_{\text{obs}} notificacionesConsultadas(\widehat{s_0}, \widehat{j})\}$
Complejidad: $\Theta(n)$
Descripción: Chequea las notificaciones del jugador y vacia su cola de notificaciones n es la cantidad de notificaciones siendo consultadas
Aliasing:las notificaciones se devuelven por referencia

RECIBIRMENSAJE(**in/out** s: `servidor`,**in** j: `jugador` **in** o: `ocurrencia`)
Pre $\equiv$ $\{\widehat{s} =_{\text{obs}} \widehat{s_0} \wedge jugador \wedge empezo?(\widehat{s_0}) \wedge jugadaValida?(\widehat{j}, \widehat{o})\}$
Post $\equiv$ $\{\widehat{s} =_{\text{obs}} mensajeProcesado(\widehat{s_0}, \widehat{j})\}$
Complejidad: $\Theta(1+\text{m*L}_{max})$
Descripción: si la jugada es valida debe sumar los puntos de la jugada y notificar a todos el nuevo puntaje sino avisar al jugador pasado por parametro que fue invalida

NUMEROCLIENTESESPERADOS(**in** s: `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ $\{True\}$
Post $\equiv$ $\{\widehat{res} =_{\text{obs}} \#esperados(\widehat{s})\}$
Complejidad:$\Theta(1)$
Descripción: retorna el numero de clientes esperados
Aliasing: se devuelve por copia

NUMEROCLIENTESCONECTADOS(**in** s: `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ $\{True\}$
Post $\equiv$ $\{\widehat{res} =_{\text{obs}} \#conectados(\widehat{s})\}$
Complejidad:$\Theta(1)$
Descripción: retorna el numero de clientes conectados
Aliasing: se devuelve por copia

JUEGO(**in** s: `servidor`) $\rightarrow$ res : `juego`
Pre $\equiv$ $\{empezo?(\widehat{s})\}$
Post $\equiv$ $\{\widehat{res} =_{\text{obs}} (\widehat{s})\}$
Complejidad:$\Theta(1)$
Descripción: retorna el juego

1.7. Representación de Servidor

Representación

Representación del servidor

`Servidor` se representa con `estr`

donde `estr` es `tupla`(*notificaciones*: `tupla<notificacionesTodos,porJugador>`, *numNotif*: `nat`, *clientes*: `tupla<esperados : nat, conectados : nat>`, *Juego*: `Juego`)

notificacionesTodos se representa con
tupla(vector(tupla(notificacion, nat)),array(nat))
en el vector se guardan todas las notificaciones que hay en el servidor junto con su numero de notif unica mientras que en el array se guarda cual fue la ultima posicion a la accedio el jugador en el vector de notificaciones totales.

porJugador se representa con
array(array((tupla(notificacion, nat)))
en el mismo se guarda en la posicion i las notificaciones del jugador i junto con su numero de notif unica
notificacion se representa con
donde `estr` es `tupla`(*tipoNotif*: `string` , *idCliente*: `nat` , *n*: `nat` , *agarraLetras*: `multiConj(letra)` , *jugada*: `ocurrencia`)

Rep : estr $\longrightarrow$ bool

Rep(s) $\equiv$ true $\Longleftrightarrow$

π_2**(s.clientes)** $\leq \pi_1$**(s.clientes)** $\wedge$
long($\pi_2(\pi_1$**(s.notificaciones)))** $= \pi_1$**(s.clientes)** $\wedge$
$(\forall i : Nat)(0 \leq i < long(\pi_2(\pi_1$**(s.notificaciones)))** $\Rightarrow_{\text{L}}$**long(**$\pi_2(\pi_1$**(s.notificaciones))[i]** $\leq long(\pi_1(\pi_1$**(s.notificaciones)))** $\wedge$
$(\forall i : Nat)(0 \leq i < long(e.porJugador) \Rightarrow_{\text{L}}$
$(\forall n : notificacion)(esta?(n, porJugador[i]) \Rightarrow_{\text{L}} \not esta?(n, notificacionesTodos)$
long(porJugador) $= \pi_1$**(s.clientes)** $\wedge$ **en juego se guarda el juego que se esta corriendo en el servidor** $\wedge$ **numNotif representa un identificador único de notificación que será asignado**

1.8. Abstraccion del Servidor

Abs : estr s $\longrightarrow$ **servidor** **{Rep(**s**)}**

Abs(s**)** $\equiv$ $s : sim | esperados(\widehat{s}) = s.clientes.esperados \wedge$
$conectados(\widehat{s}) = s.clientes.conectados \wedge$
$configuracion(\widehat{s}) = s.Juego \wedge$
$(\forall i : Nat)(0 \leq i < long(s.notificaciones.todos.nots) \Rightarrow_{\text{L}}$
$(\forall id : Nat)(0 \leq id < conectados(\widehat{s}) \Rightarrow_{\text{L}} pertenece(s.notificaciones.todos.nots[i], notificaciones(s, id)) \wedge$
$(\forall j : Nat)(0 \leq j < long(s.notificaciones.porJugador[id]) \Rightarrow_{\text{L}}$
$pertenece(s.notificaciones.porJugador[j], notificaciones(s, id))))) \wedge$
$juego(\widehat{s}) =_{\text{obs}} s.juego$

1.9. Algoritmos de Servidor

Algoritmos

---

iInicializarServidor(**in** k : `nat`, **in** v : `variante`, **in** rep : `cola(letra)`) $\rightarrow$ res : Servidor

1: $res \leftarrow (<<<vacio(), crearArreglo(k, 0)>, crearArreglo(k, vacio())>, 0, <k, 0>, NuevoJuego(k, v, rep)>$

Complejidad: $\Theta(N^2 + k*F + \Sigma * k)$
Justificacion: Crea un nuevo juego y estructuras

---

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

1: $s.clientes.conectados \leftarrow s.clientes.conectados + 1$
2: $agregarAtras((\pi_2(s.notificaciones))[s.clientes.conectados], <$ <IdCliente,s.clientes.conectados,0, $\emptyset, \emptyset >$ $s.numNotif >)$
3: $s.numNotif++$
4: **si** s.clientes.conectados = s.clientes.esperados **entonces**
5: $agregarAtras(\pi_1(s.notificaciones.notificacionesTodos), <<Empezar, 0, s.Juego.variante.tamanoTablero, \emptyset, \emptyset>, s.numNotif>)$
6: $agregarAtras(\pi_1(s.notificaciones.notificacionesTodos), <<TurnoDe, 0, 0, \emptyset, \emptyset>, s.numNotif + 1)$
7: $s.numNotif += 2$

Complejidad: $\Theta(1)$
Justificacion: Conecta un jugador al servidor

---

iConsultarNotificaciones(**in/out** s : `servidor`, **in** j : `jugador`) $\rightarrow$ res : lista

1: $res \leftarrow vacia()$
2: $desde \leftarrow \pi_2(s.notificaciones.notificacionesTodos)[j]$
3: $porJugador \leftarrow (s.notificaciones.porJugador)[j]$
4: $i \leftarrow 0$
5: **mientras** $i < longitud(porJugador) \wedge desde < longitud(\pi_1(s.notificaciones.notificacionesTodos))$ **hacer**
6: **si** $(\pi_2(porJugador[i]) < \pi_2(\pi_1(s.notificaciones.notificacionesTodos)[desde]))$ **entonces**
7: $AgregarAtras(res, \pi_1(porJugador[i]))$
8: $i++$
9: **else**
10:
11: $AgregarAtras(res, \pi_1(\pi_1(s.notificaciones.notificacionesTodos)[desde]))$
12: $desde++$
13: **mientras** $i < longitud(porJugador)$ **hacer**
14: $AgregarAtras(res, \pi_1(porJugador[i]))$
15: $i++$
16: **mientras** $desde < longitud(\pi_1(s.notificaciones.notificacionesTodos))$ **hacer**
17:
18: $AgregarAtras(res, \pi_1(\pi_1(s.notificaciones.notificacionesTodos)[desde]))$
19: $desde++$
20: $\pi_2(s.notificaciones.notificacionesTodos)[j] \leftarrow longitud(\pi_1(s.notificaciones.notificacionesTodos))$
21: $(porJugador)[j] \leftarrow vacia()$

Complejidad: $\Theta(n)$

Justificacion: Consulta las notificaciones de un jugador y desencola n notifiaciones

---

iRecibirMensaje(**in/out** s : `servidor`, **in** j : `jugador`, **in** o : `ocurrencia`)

1: **si entonces** $JugadaValida?(s.juego, o) \wedge TieneFichas(s.juego, j, o) \wedge comenzo?(s) \wedge ObtenerTurno(s.juego) == j$
2: $puntajeAnterior \leftarrow s.juego.jugadores[j].puntaje$
3: $ubicar(s.juego, o)$
4: $agregarAtras(\pi_1$(s.notificaciones.notificacionesTodos),< <sumaPuntos,j,ObtenerPuntaje(s.juego,j) - puntajeAnterior, $\emptyset, \emptyset >, s.numNotif >)$
5: $agregarAtras(\pi_1$(s.notificaciones.notificacionesTodos),< <Ubicar,j,0, $\emptyset, \emptyset >, s.numNotif >, s.numNotif + 1)$
6: $agregarAtras(((s.notificaciones.porJugador)[j], << Reponer, 0, 0, sacarDelRepositorio(s.juego, o.longitud), \emptyset >$ $, s.numNotif + 2 >)$
7: $agregarAtras(\pi_1$(s.notificaciones.notificacionesTodos),< <TurnoDe,j.turno + 1 mod(j.numDeJugadores),0, $\emptyset, \emptyset > s.numNotif + 3 >)$
8: $s.numNotif+ = 4$
9: **else**
10: $agregarAtras(((s.notificaciones.porJugador)[j], << Mal, 0, 0, \emptyset, \emptyset >, s.numNotif >)$
11: $s.numNotif + +$

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

Justificacion: el peor caso para esta funcion ocurre cuando la ocurrencia tiene longitud L_{max} en tal caso seria ${L_{max}}^2$

iNumeroClientesEsperados(**in** s : `Servidor`) $\rightarrow$ res : Nat

1: $res \leftarrow \pi_1(s.clientes)$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

iNumeroClientesConectados(**in** s : `Servidor`) $\rightarrow$ res : Nat

1: $res \leftarrow \pi_2(s.clientes)$

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

1.10. Funciones auxiliares de Servidor

comenzo?(**in** s : `servidor`) $\rightarrow$ res : bool

1: $res \leftarrow s.clientes.esperados == s.clientes.conectados$

Complejidad: $\Theta(1)$
Justificacion: Verifica si la cantidad de jugadores conectados es la misma que la de jugadores esperados

isacarDelRepositorio(**in/out** J : `juego`, **in** n : `nat`) $\rightarrow$ res : multiConj

1: $res \leftarrow ProximosN(J.repositorio, n)$
2: $DesencolarN(J.repositorio, n)$
3: $return\ res$

Complejidad: Θ(n)
Justificacion: Saca n elementos del respositorio y devuelve el multiconj que los contiene

iTieneFichas(**in** J : `Juego`,**in** j : `jugador`, **in** o : `Ocurrencia`) → res : bool

```
1:  it ← crearIt(o)
2:  fichaPertenece ← true
3:  mientras haySiguiente(o) hacer
4:      fichaPertenece &= (J.jugadores[j].mano[ord(siguiente(it).letra)]≥cantidadDeLetraEnOcurrencia(siguiente(it).letra,
    o))
6:      J.jugadores[j].Mano[ord(π3(Siguiente(o)))] − −
7:      π2(j.tablero[π1(Siguiente(o))][π2(Siguiente(o))]) ← j.Turno
8:      π3(j.tablero[π1(Siguiente(o))][π2(Siguiente(o))]) ← True
9:      Avanzar(it)
10:   return fichaPertenece
```

Complejidad: $\Theta(m^2)$
Justificacion: Para cada elemento de la ocurrencia ubica la ficha y asigna valores a la tupla del tablero.

icantidadDeLetraEnOcurrencia(**in** *letter* : `letra`, **in** o : `Ocurrencia`) → res : Nat

```
1:  it ← crearIt(o)
2:  res ← 0
3:  mientras haySiguiente(it) hacer
4:      si siguiente(it).letra = letter entonces
5:          res + +
```

Complejidad: Θ(m)
Justificacion: Es contar apariciones de una letra en ocurrencia

iObtenerTurno(**in** j : `juego`) → res : Nat

```
1:  res ← j.Turno  mód j.NumDeJugadores
```

Complejidad: $\Theta(1)$
Justificacion: Es obtener un indice de un array

iFichaEnPosicion(**in** t : `tablero`,**in** i : `nat`,**in** j : `nat`) → res : Letra

```
1:  res ← π1(t[i][j])
```

Complejidad: $\Theta(1)$
Justificacion: Es obtener el valor de una tupla después de indexar un array

1.11. Módulo ConjDigital(*String*)

Representación del ConjDigital

Un conjunto implementado sobre trie tendra la misma interfaz que un conjunto normal, donde los valores siempre seran un string donde nos interesa la pertenencia (o no) de la palabra. Todas las posts y pre condiciones fueron sacadas del apunte de módulos básicos.

Interfaz

se explica con: Conjunto (α))

géneros: .

Operaciones básicas

Vacio() → res : `conj(`α`)`

Pre≡ {true}
Post ≡ {$\widehat{res}$ =obs ∅}
Complejidad:Θ(1)
Descripción: se genera el conjunto vacio

AGREGAR(in/out C : conj(α), in$a$: α)→ res : itConj(α)
Pre≡ {$\widehat{C}$ =obs $\widehat{C_0}$}
Post≡ {c =obs Ag(a,C$_0$) ∧ $HaySiguiente(res)$ ∧$_L$ $Siguiente(res)$ = a ∧ $alias(esPermutacion?(SecuSuby(res), C)$}
Complejidad:Θ(|a|)
Descripción: agrega el elemento a al conjunto. Para poder acceder en O(1) al elemento a se devuelve un iterador a la posicion de a dentro de C
Aliasing: el elemento a se agrega por copia. El iterador se invalida sii se elimina el elemento siguiente del iterador sin utilizar la funcion ELIMINARSIGUIENTE(definida en el apunte de modulos basicos en la seccion de iteradores). Ademas, anteriores(res) y siguiente(res) podrian cambiar completamente ante cualquier operacion que modifique C sin utilizar las funciones del iterador

PERTENCE?(in C : conj(α), in$a$: α)→ res : bool
Pre ≡ {true}
Post ≡ {res =obs a ∈ C}
Complejidad:Θ(|a|)
Descripción: Devuelve true sii a pertenece al conjunto

CARDINAL(in C : conj(α))→ res : nat
Pre ≡ {true}
Post ≡ {$\widehat{res}$ =obs #C}
Complejidad: Θ(1)
Descripción: Devuelve la cantidad de elementos en el conjunto

TAD COLA(α)

Extendemos el TAD Cola para poder incluir las siguientes operaciones:

otras operaciones

próximosN : cola(α)q × Natn ⟶ Multiconj(α)	{n ≤ $tamano(q)$}
desencolarN : cola(α)q × Natn ⟶ cola(α)	{n ≤ $tamano(q)$}

Fin TAD