Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Juego de palabras

La cosa se pone compleja

Grupo d-mifflin

Integrante	LU	Correo electrónico
Guglielmino, Adrian Roberto	39/18	adrianguglielmino@gmail.com
Hajek, David	52/32	hajekuba96@gmail.com
Lopez, Fernando Mariano	81/20	fmlopez@dc.uba.ar
Lopez Bianco, Macarena	268/18	maca_lopezbianco@hotmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

JUGADA VÁLIDA

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

```
ijugadaValida (in juego : jueg, in o : ocurrencia) -> res : bool {

        if( cardinal(o) == 0){
                return true
        }

        res <- true
        palabrasFormadas <- cola(lista(letra)) vacía

        varJ <- VarianteDelJuego(jueg)
        largoMax <- longitudMax(palabras_puntos(varJ))        O(1)     <-(fn's de trie y var a
        implementar)

        if( res && cantFichas(varJ) < cardinal(o) | | largoMax < cardinal(o) ){
        O(1)
                res <- false
        }

        fichasJugador <- arreglo(nat) Copia( Fichas(juego.jugadores[turno_de(juego)]) )
O(|alfabeto|)

        if( res && ¬ tieneLasFichas(fichasJugador(juego.turno_de, o) ){
O(cardinal(o))
                res <- false
        }
                                                                (checks si las posiciones son
válidas -> libre?)
        if( res && ¬ celdasLibres?( Tablero(juego), o) ){
O(cardinal(o))
                res <- false
        }

        esHorizontal <- bool false
        esVertical <- bool false
        if(res){
                esHorizontal, esVertical <- HorizontalOVertical(o)                              O(
        cardinal(o))
                if (¬ ( esHorizontal OR esVertical) {
        O(1)
                        res <- false
                }
        }
```

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

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

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

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

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

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

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

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

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


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

                palabraPrincipal <- formarPalabraPrincipal( tablero, o, sentido)

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

                if ( EsVacia?(palabraPrincipal) OR ¬palabraLegitima?(varJ, palabraPrincipal)
        ){
                        res <- false
                }else{
                        palabras <- PalabrasTransversales(tablero(juego), o, sentido)

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

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

            return res
}

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


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


iHorizontalOVertical( in o : ocurrencia) -> horizontal : bool, vertical : bool {        Complejidad
O(o.cardinal)
            horizontal <- true                                          O(1)
            vertical <- true                                            O(1)

            if( Cardinal(o) != 0){                                      O(1)

                it <- crearIT(o)                                        O(1)
                fila0 <- Siguiente(o).fila                              O(1)
                col0 <- Siguiente(o).columna                            O(1)

                it <- Avanzar(it)                                       O(1)

                while(HaySiguiente(it) && (horizontal || vertical ) ){          O( o.cardinal -1)
                    fila1 <- Siguiente(o).fila                          O(1)
                    col1 <- Siguiente(o).columna                        O(1)

                    if(horizontal){                                     O(1)
                        horizontal <- fila1 == fila0                    O(1)
                    }
```

```
                        if(vertical){                                   O(1)
                                vertical <- col0 != col1                O(1)
                        }

                        fila0 <- fila1                                  O(1)
                        col0 <- col1                                    O(1)
                        it <- avanzar(it)                               O(1)
                }
        }

        return horizontal, vertical
}

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


comentario // sentido true -> horizontal otherwise es vertica

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

```
}

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

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


// comentario sentido = true -> horizontal otherwise vertical

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

        if( largoMax < DistanciaEntreFichas(min, max , sentido) ){
                res <- false
        }
        return res
}
```

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

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

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

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

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

        return res
}

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

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

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

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

```
        }
        return res
}

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

iocurrenciaAvector(in o : ocurrencia) -> res : vector(ficha){ // O(cardinal(ocurrencia)) =
O(Lmax)
        res <- vector(ficha) vacío
        it <- crearIT(o)
        while(HaySiguiente(it)){
                ficha <- Siguiente(it)
                AgregarAtras(res, ficha)
                it <- Avanzar(it)
        }
        return res
}

// Pre (ForAll t1,t2 : tupla(nat,nat,letra))  (existen n1,n2 :nat)
        ( ( vect[n1] && vect[n2])  YLuego
        (pi1(t1) == pi1(t2) and pi2(t1) != p1(t2)  ) OR (pi1(t1) != pi1(t2) and pi2(t1) == p1(t2) )

iordenarVectorDeFichas(in\out vect : vector(tupla(nat,nat,letra)) {   O(n^2) bubble ->
O(Lm^2)
        swapped <- true
        i <- longitud(vect)
        while(swapped){
                swapped <- false
                j <- 0
                while( j < i) {
                        mayorFila <- ( vect[j].fila > vect[j+1].fila )
                        mayorCol <- ( vect[j].columna > vect[j+1].columna )
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        esContigua <- true
        termino <- false

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

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

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

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

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

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

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

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

```
iFormarPalabraPrincipal(in tablero : tab, in o : ocurrencia, in sentido : bool) -> principal  :
lista(letra) {

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

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

        return principal
}
```

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

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

Post{
        Cardinal(ubicadas0) = Longitud(res) ∧L
        (∀ o : ocurrencia)( o ∈ ubicadas0  ⟹L
        (∃ palabra : lista(letra))( está?(palabra,res) ∧L Cardinal(o) = Longitud(palabra) ∧L
          (∀ f : ficha)(f ∈ o ⟹L (∃ let : letra )( esta?(letra, palabra) ⟹L pi3(o) = palabra[i] ) )))

          (∀ palabra : lista(letra))(está?(palabra,res) ⟹L
        (∃ o : ocurrencia)(o ∈ ubicadas0 ∧L cardinal(o) = Longitud(palabra) ∧L
         (∀ letr : letra)(esta?(letr,palabra) ⟹L (∃ f : ficha)( f ∈ o ⟹L f.letr = letr)))
        )
}
Complejidad : O(Lm^2)
Aliasing : todos los datos de entrada se pasan por referencia
```

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

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

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

	return res
}
```

1. Desarrollo

1.1. Escribir pseudocódigo

También presentaremos el algoritmo del éxito[1]:

```
hacerGuia(in A : guia, parámetroInútil : Nat) ⟶ bool
1:  i ← 0                                                          ▷ esto es Θ(1)
2:  n ← guia.cantEjercicios()                                      ▷ O(1)
3:  consultas ← diccVacio
4:  prepararMate()                                                 ▷ Ω(n^n)
5:  mientras i < n hacer
6:      pensarEjercicio(I)
7:      si tengoConsultas(i) entonces
8:          escribirConsultasEjercicio(i,consultas)
9:      else
10:         comerBizochito()
11:     end si
12:     comerBizochito()
13: end mientras
14: para miVariable hacer
15:     hacer algo
16: end para
17: devolver vacio?(consultas)
```

¡Una forma piola de anotar lo que les falte!

1.2. Elección de estructuras

Ejercicio 2: Sistema de estadísticas

Se desea diseñar un sistema de estadísticas para la cantidad de personas que ingresan a un banco. Al final del día, un empleado del banco ingresa en el sistema el total de ingresantes para ese día. Se desea saber, en cualquier intervalo de días, la cantidad total de personas que ingresaron al banco. La siguiente es una especificación del problema.

TAD IngresosAlBanco

observadores básicos

totDias : iab ⟶ nat

cantPersonas : iab i × nat d × nat h ⟶ nat $\{1 \leq d \land d \leq h \land h \leq totDias(i)\}$

generadores

Comenzar : ⟶ iab

TerminaDia : iab × nat ⟶ iab

axiomas ...

totDias(Comenzar) ≡ 0

totDias(TerminaDia(i, n)) ≡ 1 + totDias(i)

cantPersonas(TerminaDia(i, n),d, h) ≡ **if** totDias(i) < h **then** n **else** 0 **fi** + **if** totDias(i) < d **then** 0 **else** cantPersonas($i, d,$ mín(h,totDias(i))) **fi**

Fin TAD

1. Dar una estructura de representación que permita que la función *cantPersonas* tome $O(1)$.
2. Calcular cuánta memoria usa la estructura, en función de la cantidad de días que pasaron n.
3. Si el cálculo del punto anterior fue una función que no es $O(n)$, piense otra estructura que permita resolver el problema utilizando $O(n)$ memoria.

2. Renombres de tipos

[1] Puede fallar.

3. Módulo Variante

Interfaz

se explica con: VARIANTE

géneros: `variante`.

Operaciones básicas de Variante

CREARVARIANTE(**in** *tamaño* : `nat`, **in** *cant_fichas* : `nat`, **in** *valor_letras* : `dicc(letra × nat)`, **in** *palabras_validas* : `conj(Lista(letra))`) → *res* : `variante`
Pre ≡ $\{n > 0 \land cant\_fichas > 0\}$
Post ≡ $\{res =_{\text{obs}} nuevaVariante(tamaño, cant\_fichas, valor\_letras, palabras\_validas)\}$
Complejidad: $\mathcal{O}(C * L_{max})$, donde C = cantidad de palabras
Descripción: crea un nueva variante
Aliasing: pasamos valor_letras y palabras_validas por referencia no modificable

TAMAÑODELTABLERO(**in** v : `variante`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} tamañoTablero(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tamaño del tablero de una variante del juego
Aliasing: no aplica

CANTFICHAS(**in** v : `variante`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} \#Fichas(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de fichas usada por cada jugador en la variante del juego
Aliasing: no aplica

PUNTAJEDELALETRA(**in** v : `variante`, **in** l : `letra`) → *res* : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} puntajeLetra(l, v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el puntaje asociado a una determinada letra l en la variante del juego
Aliasing: no aplica

ESPALABRAPERMITIDA(**in** v : `variante`, **in** *palabra* : `Lista(letra)`) → *res* : `bool`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} esPalabraLegitima?(v, palabra)\}$
Complejidad: $\mathcal{O}(L_{max})$
Descripción: determina si una palabra es válida para la variente del juego
Aliasing: no aplica

PALABRASPERMITIDAS(**in** v : `variante`, **in** *palabras* : `Cola(Lista(letra))`) → *res* : `DiccTrie(Lista(letra), Nat)`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} (\forall palabra : Lista(letra))(esta?(palabras, palabra) \land_L palabraPermitida(variante, palabra)))\}$
Complejidad: $\mathcal{O}(Tamao(palabras) * L_{max}) = \mathcal{O}(L_{max} * L_{max})$
Descripción: determina si unas palabras son válidaa para la variente del juego
Aliasing: toma por referencia a la variante y por copia a la cola de palabras

Representación

`variante` **se representa con** `var`

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

Rep : var ⟶ bool

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

Abs : var v ⟶ variante {Rep(v)}

Abs(v) $\equiv$ va : variante / (tamañoTablero(va) = v.tamaño) $\wedge$ (#Fichas(va) = v.cant_fichas) $\wedge$ (($\forall l$: letra)(puntajeLetra(va, l) = v.valor_letras[$ord^{-1}(l)$])) $\wedge$ (($\forall p$: palabra)(palabraLegitima?(va, p) = $p \in$ v.palabras_puntos))

Algoritmos

---

iCrearVariante(**in** n : `nat`, **in** f : `nat`, **in** *letraPuntaje* : `dicc(letra` $\times$ `nat)`, **in** *palabrasValidas* : `conj(List(letra))`) $\longrightarrow$ res : `var`

1: $tamaño \leftarrow n$ ▷ $\mathcal{O}(1)$
2: $cant\_fichas \leftarrow f$ ▷ $\mathcal{O}(1)$
3: VALOR_LETRAS = ARRAY[CLAVES(LETRAPUNTAJE).SIZE()]
4: **para** i = ord(l), con l en claves(letraPuntaje) **hacer**
5: valor_letras[i]= obtener(l,letraPuntaje)
6: **end para**

Complejidad:

---

iTamañoDelTablero(**in** v : `var`) $\longrightarrow$ res : `nat`

1: $res \leftarrow v.tamaño$ ▷ $\mathcal{O}(1)$

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

---

iCantFichas(**in** v : `var`) $\longrightarrow$ res : `nat`

1: $res \leftarrow v.cant\_fichas$ ▷ $\mathcal{O}(1)$

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

---

iPuntajeDeLaLetra(**in** v : `var`, **in** l : `letra`) $\longrightarrow$ res : `nat`

1: $res \leftarrow v.valor\_letras$[ord$^{-1}(l)$]o ▷ $\mathcal{O}(1)$

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

---

iEsPalabraPermitida(**in** v : `var`, **in** p : `Lista(letra)`) $\longrightarrow$ res : `bool`

1: res = Definido?(v.palabras_puntos, p) ▷ $\mathcal{O}(L_{max})$

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

---

iEsPalabrasPermitidas(**in** v : `var`, **in** p : `Lista(letra)`) $\longrightarrow$ res : `bool`

1: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
2: **mientras** $EsVacia?(palabras)res$ **hacer** ▷ $\mathcal{O}(cantidaddepalabras)$
3: $palabra \leftarrow Proximo(palabras)$ ▷ $\mathcal{O}(1)$
4: Desencolar(alabras) ▷ $\mathcal{O}(1)$
5: $res \leftarrow palabraPermitida(variante, palabra)$ ▷ $\mathcal{O}(L_{max})$
6: **end mientras**

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

---

4. Módulo Tablero

Interfaz

se explica con: TABLERO

géneros: `tablero`

Operaciones básicas de Tablero

TAMAÑO(**in** t : `tablero`) $\rightarrow$ res : `nat`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} tamaño(t)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tamaño de un tablero
Aliasing: no aplica

HAYUNALETRAENPOS?(**in** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{enTablero?(t, fila, columna)\}$
Post $\equiv \{res =_{\text{obs}} hayLetra?(t, fila, columna)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: determina si un casillero indicado de un tablero está ocupado
Aliasing: no aplica

LETRAENPOS(**in** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\rightarrow$ res : `letra`
Pre $\equiv \{enTablero?(t, fila, columna) \wedge_L hayLetra?(t, fila, columna)\}$
Post $\equiv \{res =_{\text{obs}} letra(t, fila, columna)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la letra de un casillero del tablero
Aliasing: no aplica

CREARTABLERO(**in** n : `nat`) $\rightarrow$ res : `tablero`
Pre $\equiv \{n > 0\}$
Post $\equiv \{res =_{\text{obs}} nuevoTablero(n)\}$
Complejidad: $\mathcal{O}(N^2)$
Descripción: crea un tablero de tamaño n
Aliasing: no aplica

PONERLETRA(**in/out** t : `tablero`, **in** *fila* : `nat`, **in** *columna* : `nat`, **in** l : `letra`)
Pre $\equiv \{t = t_0 \wedge (enTablero?(t, fila, columna) \wedge_L \neg hayLetra?(t, fila, columna))\}$
Post $\equiv \{res =_{\text{obs}} ponerLetra(t_0, fila, columna, l)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tablero con la letra l colocada en el casillero indicado
Aliasing: no aplica

ENTABLERO(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} enTablero?(t, i, j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el casillero (i,j) está en el tablero
Aliasing: no aplica

ESTALIBRE(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} libre?(t, i, j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el casillero (i,j) de un tablero está libre
Aliasing: no aplica

ESTAOCUPADA(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} ocupada?(t, i, j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica si el casillero (i,j) de un tablero está ocupada
Aliasing: no aplica

PONERLETRAS(**in/out** t : `tablero`, **in** *ocurrencia* : `conjunto(tupla(nat, nat, letra))`)
Pre $\equiv \{t = t_o \wedge celdasLibres?(t, ocurrencia) \wedge ((\forall i, j : nat)(\forall l, l' : letra)((\langle i, j, l\rangle \in ocurrencia \wedge \langle i, j, l'\rangle \in ocurrencia) \Rightarrow l = l'))\}$
Post $\equiv \{res =_{\text{obs}} ponerLetras(t_0, ocurrencia)\}$
Complejidad: $\mathcal{O}(tamaño(ocurrencia))$
Descripción: ubica cada elemento de la ocurrencia en le tablero
Aliasing: Pasamos la ocurrencia por referencia no modificable

SONCELDASLIBRES?(**in** t : `tablero`, **in** *ocurrencia* : `conjunto(tupla(nat, nat, letra))`) $\rightarrow$ res : `bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} celdasLibres?(t, ocurrencia)\}$
Complejidad: $\mathcal{O}(tamaño(ocurrencia))$

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

Representación

`tablero` **se representa con** `tab`

donde tab es arreglo(arreglo(tupla $\langle$ *ocupado*: `bool` , *letr*: `letra` $\rangle$))

Rep : tab $\longrightarrow$ bool

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

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

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

Algoritmos

---

iTamaño(**in** t : `tab`) $\longrightarrow$ res : `nat`

1: $res \leftarrow t.tamaño$ ▷ $\mathcal{O}(1)$

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

---

iHayUnaLetraEnPos?(**in** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\longrightarrow$ res : `bool`

1: $res \leftarrow (t.casilleros[fila][columna]).ocupado$ ▷ $\mathcal{O}(1)$

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

---

iLetraEnPos(**in** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\longrightarrow$ res : `letra`

1: $res \leftarrow (t.casilleros[fila][columna]).letr$ ▷ $\mathcal{O}(1)$

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

---

iCrearTablero(**in** *tam* : `nat`) $\longrightarrow$ res : `tab`

1: $t.tamao \leftarrow tam$ ▷ $\mathcal{O}(1)$
2: $t.casilleros \leftarrow arreglo(arreglo(tupla < false, "" >)[n])[n]$ ▷ $\mathcal{O}(n^2)$
3: $res \leftarrow t$ ▷ $\mathcal{O}(1)$

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

---

iPonerLetra(**in/out** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`, **in** l : `letra`)

1: $t.casilleros[fila][columna]).casillero.letr \leftarrow l$ ▷ $\mathcal{O}(1)$
2: $t.casilleros[fila][columna]).casillero.ocupada \leftarrow true$ ▷ $\mathcal{O}(1)$

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

---

iEnTablero(**in** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\longrightarrow$ res : `bool`

1: $res \leftarrow (fila < t.tamao) \wedge (columna < t.tamao)$ ▷ $\mathcal{O}(1)$

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

---

iestaLibre(**in** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\longrightarrow$ res : `bool`

1: $res \leftarrow ienTablero(t, fila, columna) \wedge \neg t.casilleros[fila][columna]).casillero.ocupada$ ▷ $\mathcal{O}(1)$

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

---

iestaOcupada(**in** t : `tab`, **in** *fila* : `nat`, **in** *columna* : `nat`) $\longrightarrow$ *res* : `bool`

1: $res \leftarrow ienTablero(t, fila, columna) \wedge t.casilleros[fila][columna]).casillero.ocupada$ ▷ $\mathcal{O}(1)$

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

iponerLetras(**in/out** t : `tab`, **in** *ocurrencia* : `conjuntoLineal(tupla(nat,nat,letra))`)

1: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
2: **mientras** ihaySiguiente(it) **hacer** ▷ $\mathcal{O}$(tamaño(ocurrencia))
3: **si** iestaOcupada(tab, ficha0.fila, ficha0.columna) **entonces**
4: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
5: **end si**
6: iavanzar(it) ▷ $\mathcal{O}(1)$
7: **end mientras**

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

iSonCeldasLibres?(**in** t : `tab`, **in** *ocurrencia* : `conjuntoLineal(tupla(nat,nat,letra))`) $\longrightarrow$ *res* : `bool`

1: $it \leftarrow crearIt(o)$ ▷ $\mathcal{O}(1)$
2: $res \leftarrow true$ ▷ $\mathcal{O}(1)$
3: **mientras** haySiguiente(it) **hacer** ▷ ($\mathcal{O}(tamao(ocurrencia))$
4: **si** estaOcupada(tab, ficha0.fila, ficha0.columna) **entonces**
5: $res \leftarrow false$ ▷ $\mathcal{O}(1)$
6: **end si**
7: iavanzar(it) ▷ $\mathcal{O}(1)$
8: **end mientras**

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

5. Módulo Juego

Interfaz

se explica con: JUEGO

géneros: `juego`.

Operaciones básicas de Juego

CREARNUEVOJUEGO(**in** *jugadores* : `nat`, **in** *varianteDelJuego* : `variante`, **in** *bolsaDeFichas*: `cola(letra)`) $\rightarrow$ *res* : `juego`
Pre $\equiv$ {long(bolsaDeFichas) $\leq tamañoTablero(v) * tamañoTablero(v) + k * fichas(v) \wedge k > 0$}
Post$\equiv$ $\{res = nuevoJuego(jugadores, varianteDelJuego, bolsaDeFichas)\}$
Complejidad: $\mathcal{O}(N^2 + F * K + \Sigma * K)$
Descripción: crea un nuevo juego
Aliasing: No aplica

UBICARFICHAS(**in/out** j : `juego`, **in** *ocurrencia* : `o`, **in** *nuevasPalabras* : `conj(palabra)`) $\rightarrow$ *res* : `juego`
Pre $\equiv$ $\{j = j0 JugadaValida?(j, o) \wedge nuevasPalabras = palabrasUbicadas(ocurrenciasDePalabras(ponerLetras(tablero(j), o))$
Post $\equiv$ $\{res = ubicar(j0, o)\}$
Complejidad: $\mathcal{O}(cardinal(o))$
Descripción: ubica las fichas en el tablero, pasa el turno, actualiza los puntos del jugador y repone la cantidad de fichas ubicadas
Aliasing: se pasa el juego por referencia modificable

VARIANTEDELJUEGO(**in** j : `juego`) $\rightarrow$ *res* : `variante`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res = variante(j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la variante del juego
Aliasing: res por referencia no modificable

CANTJUGADORES(**in** j : `juego`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res = \#Jugadores(j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de jugadores
Aliasing: No aplica

REPOSITORIODELJUEGO(**in** j : `juego`) $\rightarrow$ *res* : `cola(letras)`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res = repositorio(j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el repositorio del juego
Aliasing: res por referencia (modificable)

TABLERO(**in** j : `juego`) $\rightarrow$ *res* : `tablero`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res = tablero(j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tablero
Aliasing: se pasa por referencia el tablero (modificable)

POSDETABLERO(**in** j : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ *res* : `tupla(bool, letra)`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =< ocupada?(tablero(j), i, j), letra(tablero(j), i, j) >\}$
Complejidad: $\mathcal{O}(1)$
Descripción: dada una posición del tablero, devuelve si está ocupada y la letra que lo ocupa (si está vacía, devuelve el caracter vacío)
Aliasing: se pasa por referencia el tablero (modificable)

TURNOACTUAL(**in** j : `juego`) $\rightarrow$ *res* : `nat`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res = turno(j)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve a quien le toca jugar
Aliasing: No aplica

FICHASDEJUGADOR(**in** j : `juego`, **in** i : `nat`) $\rightarrow$ *res* : `arreglo(nat)`
Pre $\equiv$ $\{i < cantJugadores(j)\}$
Post $\equiv$ $\{(\forall f : ficha)(f \in fichas(i, j) \Rightarrow_L f \in \pi 1(res) \wedge (f, fichas(i, j) = \pi 2(res))\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de fichas de cada letra que tiene el jugador
Aliasing: No aplica

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

Pre $\equiv \{i < cantJugadores(j)\}$
Post $\equiv \{res = cardinal(jugadores(j)).puntos.ocurrencias)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el puntaje del jugador
Aliasing: No aplica

JUGADAVALIDA(**in** j : `juego`, **in** o : `ocurrencia`) $\rightarrow$ res : $bool$
Pre $\equiv \{i < cantJugadores(j)\}$
Post $\equiv \{res = (l, fichasDeJugador(j, i))\}$
Complejidad: $\mathcal{O}(L_{max})$
Descripción: dice si la jugada es válida
Aliasing: No aplica

PALABRASJUGADAS(**in** j : `juego`, **in** o : `ocurrencia`) $\rightarrow$ res : $cola(lista(letra))$
Pre $\equiv \{jugadaValida(juego, o) \wedge_L ubicadas0 = palabrasUbicadas(ocurrenciasDePalabras(ponerLetras(tablero(juego), o)), o)\}$
Post $\equiv \{Cardinal(ubicadas0) = Longitud(res) \wedge_L (\forall o : ocurrencia)(o \in ubicadas0 \Rightarrow_L (\exists let : letra)(est?(letra, palabra) \Rightarrow_L \pi_3(o) = palabra[i])) \wedge_L (\forall palabra : lista(letra))(est?(palabra, res) \Rightarrow_L (\exists o : ocurrencia)(o \in ubicadas0 \wedge_L cardinal(o) = Longitud(palabra) \wedge_L (\forall letr : letra)(est?(letr, palabra) \Rightarrow_L (\exists f : fichas)(f \in o \Rightarrow_L \pi_3(f) = letr))))\}$
Complejidad: $\mathcal{O}(L_{max})$
Descripción: devuelve todas las palabras jugadas dada una ocurrencia
Aliasing: todos los datos de entrada se pasan por referencia

Representación

`juego` **se representa con** `jueg`

donde `jueg` es `tupla`(*variante*: `var`, *repositorio*: `cola(letra)`, *tablero*: `tab)`, *jugadores*: `array[jugador])`, *turnoDe*: `nat)`)

donde `jugador` es `tupla`(*fichas*: `Arreglo(Nat)`, *puntos*: `, puntaje : Nat, ocurrencias : Cola(palabra)`)

Rep : juego $\longrightarrow$ bool

Rep($jueg$) $\equiv$
true $\iff$ (Longitud(j.repositorio) >= Tamao(j.tablero) * Tamao(j.tablero) + Longitud(j.jugadores) * fichas(j.var)) $\wedge$ 0 < (Longitud(j.jugadores) $\wedge$ j.turno$_de$ < ($Longitud(j.jugadores) \wedge (\forall i : nat)(i < longitud(j.jugadores) \Rightarrow_L cantFichas(j.variante) = cantFichasTiene(j, i))$

Abs : juego ju $\longrightarrow$ jueg {Rep(ju)}

Abs(ju) $\equiv$

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

Algoritmos

---

iCrearNuevoJuego(**in** *cantJugadores* : `nat`, **in** *varianteDelJuego* : `variante`, **in** *bolsaDeFichas* : `cola(letra)`) $\longrightarrow$ *res* : `jueg`

1: $jueg.var \leftarrow varianteDelJuego$ ▷ $\mathcal{O}(1)$
2: $jueg.repositorio \leftarrow bolsaDeFichas$ ▷ $\mathcal{O}(1)$
3: $tam \leftarrow itamañoDelTablero(var)$ ▷ $\mathcal{O}(1)$
4: $jueg.tablero \leftarrow icrearTablero(tam)$ ▷ $\mathcal{O}$(N*N)
5: $fichas \leftarrow iCantFichas(var)$ ▷ $\mathcal{O}(1)$
6: $jueg.jugadores \leftarrow array[](cantJugadores)$ ▷ $\mathcal{O}$(k)
7: $i \leftarrow 0$ ▷ $\mathcal{O}(1)$
8: **mientras** $i < cantJugadores(j)$ **hacer** ▷ $\mathcal{O}$(tamao(palabra))
9: $jueg.jugadores[i] \leftarrow jugador$ ▷ $\mathcal{O}(1)$
10: $repositorioDelJugador \leftarrow array[fichas](fichas)$ ▷ $\mathcal{O}$(F)
11: $fichasPorJugador \leftarrow array[nat](letrasDelAbecedario)$ ▷ $\mathcal{O}(|\sigma|)$ jugador.puntos.puntaje ← 0 ▷ $\mathcal{O}(1)$
13: $jugador.puntos.ocurrencia \leftarrow cola < palabras >$ ▷ $\mathcal{O}(1)$
14: $j \leftarrow 0$ ▷ $\mathcal{O}(1)$
15: **mientras**
16: **hacer** $fichaDelJugador \leftarrow iproximo(jueg.repositorio)$ ▷ $\mathcal{O}(1)$
17: $repositorioDelJugador[j] \leftarrow fichaDelJugador$ ▷ $\mathcal{O}(1)$
18: $jueg.repositorio \leftarrow idesencolar(jueg.repositorio)$ ▷ $\mathcal{O}(1)$
19: $letraDeLaFicha \leftarrow fichaDelJugador.letra$ ▷ $\mathcal{O}(1)$
20: $fichasPorJugador[ord^{-}1(letraDeLaFicha)] \leftarrow fichasPorJugador[ord^{-}1(letraDeLaFicha)] + 1$ ▷ $\mathcal{O}(1)$
21: $j \leftarrow j + 1$ ▷ $\mathcal{O}(1)$
22: **end mientras**
23: $jugador.fichas \leftarrow fichasPorJugador$ ▷ $\mathcal{O}(1)$
24: $i \leftarrow i + 1$ ▷ $\mathcal{O}(1)$
25: **end mientras**
26: $jueg.turnoDe \leftarrow 0$ ▷ $\mathcal{O}(1)$
27: $res \leftarrow jueg$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(N^2 + F * K + \Sigma * K)$

---

iUbicarFichas(**in/out** j : `juego`, **in** o : `ocurrencia`, **in** *nuevasPalabras* : `cola(lista(letra))`) $\longrightarrow$ *puntajePrevio* : `nat`

H

1: $jugador0 \leftarrow j.jugadores[turno_d e(j)]$ ▷ $\mathcal{O}(1)$
2: $it \leftarrow crearIT(o)$ ▷ $\mathcal{O}(1)$
3: ponerLetras(t, o) ▷ $\mathcal{O}$(cardinal(o))
4: $nroNuevaLetra \leftarrow ord - 1(iproxima(juego.repositorio))$ ▷ $\mathcal{O}(1)$
5: $jugado0[nroNuevaLetra] \leftarrow jugado0[nroNuevaLetra] + 1$ ▷ $\mathcal{O}(1)$
6: **mientras** $\neg EsVaca?(nuevasPalabras)$ **hacer** ▷ $\mathcal{O}$(m+1) = $\mathcal{O}$(m+1) = $\mathcal{O}$(Lm)
7: $palabra0 \leftarrow Proximo(nuevasPalabras)$ ▷ $\mathcal{O}(1)$
8: Encolar(jugador0.puntos.ocurrencias, palabra0) ▷ $\mathcal{O}(1)$
9: Desencolar(nuevasPalabras) ▷ $\mathcal{O}(1)$
10: **end mientras**
11: $turno \leftarrow j.turnoDe + 1$ ▷ $\mathcal{O}(1)$
12: **si** cantJugadores(j) == turno **entonces**
13: $turno \leftarrow turno - cantJugadores(j)$ ▷ $\mathcal{O}(1)$
14: **end si**
15: $puntajePrevio \leftarrow jugador0.puntos.puntaje$

Complejidad :O(cardinal(o))

---

iVarianteDelJuego(**in** j : `juego`) $\longrightarrow$ *res* : `variante`

$res \leftarrow j.variante$ ▷ $\mathcal{O}(1)$

---

iRepositorioDelJuego(**in** j : `juego`) $\longrightarrow$ *res* : `cola(letras)`

$res \leftarrow j.repositorio$ ▷ $\mathcal{O}(1)$

---

iTablero(**in** j : juego) $\longrightarrow$ res : tab

$res \leftarrow j.tablero$	▷ $\mathcal{O}(1)$

iCantJugadores(**in** j : juego) $\longrightarrow$ res : nat

$res \leftarrow Longitud(j.jugadores)$	▷ $\mathcal{O}(1)$

iPosDeTablero(**in** j : juego, **in** i : nat , **in** j : nat) $\longrightarrow$ res : tupla(bool, letra)

$res \leftarrow j.tablero[i][j]$	▷ $\mathcal{O}(1)$

iTurnoActual(**in** j : juego) $\longrightarrow$ res : nat

$res \leftarrow j.turnoDe$	▷ $\mathcal{O}(1)$

iFichasDeJugador(**in** j : juego, **in** i : nat) $\longrightarrow$ res : array(nat)

$res \leftarrow j.jugador[i].fichas$	▷ $\mathcal{O}(1)$

iPuntaje(**in** j : juego, **in** i : nat) $\longrightarrow$ res : nat

$puntosACalcular \leftarrow 0$	▷ $\mathcal{O}(1)$
$c \leftarrow j.jugadores[i]).puntos.ocurrencias$	▷ $\mathcal{O}(1)$
mientras $\neg iesVacia?(c)$ **hacer** p $\leftarrow iproximo(c)$	▷ $\mathcal{O}(1)$
$c \leftarrow idesencolar(c)$	▷ $\mathcal{O}(1)$
$puntosDeLaPalabra \leftarrow 0$	
$j \leftarrow 0$	▷ $\mathcal{O}(1)$
mientras j <ilong(p)-1 **hacer**	
$puntajeLetra \leftarrow ipuntajeLetra(p[i])$	▷ $\mathcal{O}(1)$
$puntosACalcular \leftarrow puntosACalcular + puntajeLetra$	▷ $\mathcal{O}(1)$
end mientras	
$puntaje \leftarrow j.jugadores[i]).puntos.puntaje$	▷ $\mathcal{O}(1)$
$puntaje \leftarrow puntaje + puntosACalcular$	▷ $\mathcal{O}(1)$
$res \leftarrow puntaje$ =0	

iPalabrasJugadas(**in** j : juego, **in** o : ocurrencia) $\longrightarrow$ res : cola(lista(letra))

$tablero \leftarrow tablero(juego)$	▷ $\mathcal{O}(1)$
$esHorizontal, esVertical \leftarrow HorizontalOVertical(o)$	▷ $\mathcal{O}$(o.cardinal)
$sentido \leftarrow true$	▷ $\mathcal{O}(1)$
si esVertical **entonces**	
$sentido \leftarrow false$	▷ $\mathcal{O}(1)$
end si	
$res \leftarrow PalabrasTransversales(tablero, o, sentido)$	▷ $\mathcal{O}$(????)
$palabraPrincipal \leftarrow formarPalabraPrincipal(tablero, o, sentido)$	▷ $\mathcal{O}$(???????)
$Encolar(res, palabraPrincipal)$	▷ $\mathcal{O}(1)$
si Cardinal(o) = 1 **entonces**	
$palabraHorizontal \leftarrow formarPalabraPrincipal(tablero, o, sentido)$	▷ $\mathcal{O}$(???????)
si longitud(palabraPrincipal) = 1 $\wedge longitud(palabraHorizontal = 1)$ **entonces** $Desencolar(res)$	
si	
end si entonces	

6. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `servidor`.

Operaciones básicas de Servidor

#JUGESPERADOS(**in** s : `servidor`) → res : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} \#esperados(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica la cantidad de jugadores esperados en el servidor
Aliasing: no aplica

#JUGCONECTADOS(**in** s : `servidor`) → res : `nat`
Pre ≡ $\{true\}$
Post ≡ $\{res =_{\text{obs}} \#conectados(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica la cantidad de jugadores conectados en el servidor
Aliasing: no aplica

CONFIGURACIONDELSERVIDOR(**in** s : `servidor`) → res : `tupla(variante, cola(letra))`
Pre ≡ $\{\neg empezo?(s)\}$
Post ≡ $\{res =_{\text{obs}} configuracion(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: indica el juego que se está jugando en el servidor
Aliasing: no aplica

JUEGODELSERVIDOR(**in** s : `servidor`) → res : `juego`
Pre ≡ $\{empezo?(s)\}$
Post ≡ $\{res =_{\text{obs}} juego(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la variante y el repositorio de un servidor
Aliasing: no aplica

NOTISDELJUGADOR(**in** s : `servidor`, **in** $jugador$: `nat`) → res : `cola(notif)`
Pre ≡ $\{jugador < \#conectados(s)\}$
Post ≡ $\{res =_{\text{obs}} notificaciones(s, jugador)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cola de notificaciones de un jugador
Aliasing: no aplica

CREARNUEVOSERVIDOR(**in** $cantJugadores$: `nat`, **in** $variante$: `var`, **in** $bolsaDeFichas$: `cola(letras)`) → res : `s`
Pre ≡ $\{long(bolsaDeFichas) \geq tamavãroTablero(variante) * tamaoTablero(variante) + cantJugadores * \#fichas(variante)\}$
Post ≡ $\{s = nuevoServidor(cantJugadores, variante, bolsaDeFichas)\}$
Complejidad: $\mathcal{O}(N^2 + |\Sigma| + \text{FK})$
Descripción:$creaunnuevoservidor$
Aliasing:$sepasantodoslosinputsporreferencia$

CONECTARCLIENTEALSERVIDOR(**in/out** s : `servidor`)
Pre ≡ $\{s = s_0 \wedge \neg empezo?(s)\}$
Post ≡ $\{s = conectarCliente(s)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve un servidor con un cliente que se conecta
Aliasing: pasamos el servidor por referencia

CONSULTARNOTIS(**in/out** s : `servidor`, **in** $jugador$: `nat`)
Pre ≡ $\{s = s_0 \wedge jugador < \#conectados(s)\}$
Post ≡ $\{esVacia?(notificaciones(s, jugador)) \wedge s = consultar(s_0, jugador)\}$
Complejidad: $\mathcal{O}(n)$, donde n es la cantidad de mensajes en la cola del jugador
Descripción: devuelve el servidor una vez que un jugador consulta sus notificaciones
Aliasing: pasamos al servidor por referencia

RECIBIRMENSAJE(**in/out** s : `servidor`, **in** cid : `nat`, **in** o : `ocurrencia`)
Pre ≡ $\{(s = s_0) \wedge (0 \leq cid) \wedge (cid < \#conectados(s))\}$
Post ≡ $\{s = recibirMensaje(s_0, cid, o)\}$
Complejidad: $\mathcal{O}(fpj * ln * K)$
Descripción: devuelve un servidor con un nuevo mensaje al jugador cid tras realizar una ocurrencia
Aliasing: no aplica

EMPEZOJUEGO(**in** s : `servidor`) → res : `bool`
Pre ≡ $\{true\}$
Post ≡ $\{\#conectados(s) =_{\text{obs}} \#esperados(s)\}$
Complejidad: $\mathcal{O}(1)$

IJUGADAVALIDA(**in** j : `juego`, **in** o : `ocurrencia`) $\longrightarrow$ res : `bool`

si Cardinal(o) == 0 **entonces** ▷ $\mathcal{O}(1)$
  devolver false ▷ $\mathcal{O}(1)$
end si

$res \leftarrow true$ ▷ $\mathcal{O}(1)$
$palabrasFormadas \leftarrow cola(lista(letra))Vaca$ ▷ $\mathcal{O}(1)$
$varJ \leftarrow VarianteDelJuego(jueg)$ ▷ $\mathcal{O}(1)$
$largoMax \leftarrow longitudMax(palabras_puntos(varJ))$ ▷ $\mathcal{O}(1)$

si res $\wedge_L$ (cantFichas(varJ) < cardinal(o) $\vee_L$ largoMax < cardinal(o)) **entonces** ▷ $\mathcal{O}(1)$
  $res \leftarrow false$ ▷ $\mathcal{O}(1)$
end si

$fichasJugador \leftarrow arreglo(nat)Copia(Fichas(juego.jugadores[turno\_de(juego)])$ ▷ $\mathcal{O}|\Sigma|$
si res $\wedge_L$ $\neg tieneLasFichas(fichasJugador, o)$ **entonces** ▷ $\mathcal{O}$(cardinal(o))
  $res \leftarrow false$
end si

si res $\wedge_L$ $\neg celdasLibres?(Tablero(juego), o)$ **entonces** ▷ $\mathcal{O}$(cardinal(o))
  $res \leftarrow false$
end si

$esHorizontal \leftarrow false$
$esVertical \leftarrow false$

si res **entonces** $esHorizontal, esVertical \leftarrow HorizontalOVertical(o)$ ▷ $\mathcal{O}$(cardinal(o)
  si $\neg(esHorizontal \vee_L$ esVertical)) **entonces**
    $res \leftarrow false$ ▷ $\mathcal{O}(1)$
  end si
end si
si res $\wedge_L$ $\neg ocurrenciaSinPosicionesRepetidas(o)$ **entonces** ▷ $\mathcal{O}(Lm^2)$
  $res \leftarrow false$
end si
$sentido \leftarrow true$
si res $\wedge_L$ esVertical **entonces**
  $sentido \leftarrow false$
end si
si res $\wedge_L$ $\neg DistanciaNoMayorALongMax(o, sentido, largoMax)$ **entonces**
res $\leftarrow false$
end si
si res $\wedge_L$ Cardinal(o) == 1 **entonces**
  $iter \leftarrow CreatIT(o)$
  $ficha0 \leftarrow Siguiente(iter)$
  $palabraVertical \leftarrow PalabraTransversalVertical(tablero(juego), ficha0)$
  si $\neg palabraLegitima(varJ, palabraVertical)$ **entonces** res $\leftarrow false$
end si
end si
$palabraHorizontal \leftarrow PalabraTransversalHorizontal(tablero(juego), ficha0)$
si res $\wedge_L$ $\neg palabraLegitima(varJ, palabraHorizontal)$ **entonces** $res \leftarrow false$
end si
end si
si res $\wedge_L$ $\neg Cardinal(o)$!=1 **entonces**
  palabraPrincipal $\leftarrow formarPalabraPrincipal(tablero, o, sentido)$

si EsVacia?(palabraPrincipal) $\vee_L \neg$ palabraLegitima?(varJ, palabraPrincipal) **entonces**
  $res \leftarrow false$
else
  $palabras \leftarrow PalabrasTransversales(tablero(juego), o, sentido)$
  si $\neg PalabrasLegtimas?(varJ, palabras)$ **entonces**
    res $\leftarrow false$

=0

iCantFichasTiene(**in** j : `juego`, **in** i : `nat`, **in** l : `letra`) $\longrightarrow$ res : `nat`

$res \leftarrow 0$ ▷ $\mathcal{O}(1)$
para j in longitud(j.jugadores[i].fichas) **hacer** ▷ $\mathcal{O}(\|\Sigma\|)$
res $\leftarrow res + jugadores[i].fichas[j]$ ▷ $\mathcal{O}(1)$

$i++$ ▷ $\mathcal{O}(1)$
Complejidad: $\mathcal{O}(\|\Sigma\|)) = 0$
Pre{i <cantJugadores(j)}
Post{ res es la cantidad de fichas en total que tiene el jugador i)}

iTieneLasFichas(**in** *fichas* : `arreglo(nat)`, **in** o : `ocurrencia`) $\longrightarrow$ res : `nat`

$res \leftarrow true$
$it \leftarrow crearIT(o)$
mientras HaySiguiente(it) $\wedge res$ **hacer** orden $\leftarrow ord^{-}1(Siguiente(o).letra)$
$fichas[orden] \leftarrow fichas[orden] - 1$
si fichas[orden] <0 **entonces**
$res \leftarrow false$
end si
$it \leftarrow avanzar(it)$

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

Representación

`servidor` se representa con `serv`

donde `serv` es `tupla`(*#jugadoresConectados*: `nat`, *#jugadoresEsperados*: `nat`, *notificacionesPersonales*: `arreglo(cola(notif))`, *notificacionesParaTodos*: `tupla` ⟨ `cantidadVistos : arreglo(nat), notis: vector(notif)` ⟩)

Rep : serv $\longrightarrow$ bool

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

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

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

Algoritmos

i#jugEsperados(**in** s : `serv`) $\longrightarrow$ res : `nat`

1: $res \leftarrow s.\#jugadoresEsperados$ ▷ $\mathcal{O}(1)$

Complejidad: $\mathcal{O}(1) = 0$

i#jugConectados(**in** s : `serv`) $\longrightarrow$ res : `nat`

1: $res \leftarrow s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$

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

iconfiguracionDelServidor(**in** s : `serv`) $\longrightarrow$ res : `tupla(variante, cola(letra))`

1: $res \leftarrow upla < ivarianteDelJuego(s.juego), irepositorioDelJuego(s.juego) >$ ▷ $\mathcal{O}(1)$

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

ijuegoDelServidor(**in** s : `serv`) $\longrightarrow$ res : `juego`

1: $res \leftarrow s.juego$ ▷ $\mathcal{O}(1)$

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

INOTIDELJUGADOR(**in** s : `serv`, **in** cid : `nat`) $\longrightarrow$ res : `vector<notif>`

1: $notisPers \leftarrow s.notificacionesPersonales[cid-1]$ ▷ $\mathcal{O}(1)$
2: $notisTodos \leftarrow s.notificacionesParaTodos.notis$ ▷ $\mathcal{O}(1)$
3: $n \leftarrow s.notificacionesParaTodos.cantidadVistos[cid-1]$ ▷ $\mathcal{O}(1)$
4: **mientras** ¬ iVacía(notisPers) **hacer** ▷ $\mathcal{O}$(longitud(notisPers))
5: AgregarAtras(res, proximo(notisPers)) ▷ $\mathcal{O}$(longitud(res))
6: Desencolar(notisPers) ▷ $\mathcal{O}(1)$
7: **end mientras**
8: $l \leftarrow ilongitud(notisTodos) - 1$ ▷ $\mathcal{O}(1)$
9: **si** N $\leq l$ **entonces** ▷ $\mathcal{O}$(l-n)
10: **para** i = n .. l **hacer** ▷ $\mathcal{O}$(longitud(res))
11: AgregarAtras(res, notisTodos[i])
12: **end para**
13: **end si**

Complejidad: $\mathcal{O}(longitud(notisPers) + cantidaddemensajesnovistosdelasnotificacionesGenerales)$

ICREARNUEVOSERVIDOR(**in** $cantJugadores$: `nat`, **in** $variante$: `var`, **in** $bolsaDeFichas$: `cola(letra)`) $\longrightarrow$ res : `servidor`

1: $res.\#jugadoresConectados \leftarrow 0$ ▷ $\mathcal{O}(1)$
2: $res.\#jugadoresEsperados \leftarrow cantJugadores$ ▷ $\mathcal{O}(1)$
3: $res.notificacionesPersonales \leftarrow array[cantJugadores]$ ▷ $\mathcal{O}$(K)
4: $res.notificacionesParaTodos.cantidadVistos \leftarrow array[cantJugadores]$ ▷ $\mathcal{O}$(K)
5: $res.notificacionesParaTodos.notis \leftarrow ivacio()$ ▷ $\mathcal{O}(1)$
6: $res.juego \leftarrow icrearNuevoJuego(cantJugadores, variante, bolsaDeFichas)$ ▷ $\mathcal{O}(N^2 + |\Sigma| + \text{FK})$

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

ICONECTARCLIENTEALSERVIDOR(**in/out** s : `serv`)

1: $\#conectados \leftarrow s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$
2: $\#esperados \leftarrow s.\#jugadoresEsperados$ ▷ $\mathcal{O}(1)$
3: #conectados++ ▷ $\mathcal{O}(1)$
4: iencolar(s.notificacionesPersonales[#conectados-1],iIDCliente[#conectados]) ▷ $\mathcal{O}(1)$
5: **si** #CONECTADOS <#ESPERADOS **entonces** ▷ $\mathcal{O}(1)$
6: iAgregarAtras(s.notificacionesParaTodos.notis,iEmpezar(s.juego.variante.tamaño)) ▷ $\mathcal{O}(1)$
7: iAgregarAtras(s.notificacionesParaTodos.notis, iTurnoDe(0)) ▷ $\mathcal{O}(1)$
8: **end si**

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

ICONSULTARNOTIS(**in/out** s : `serv`, **in** cid : `nat`)

1: $notisPers \leftarrow s.notificacionesPersonales[cid-1]$ ▷ $\mathcal{O}(1)$
2: $notisTodos \leftarrow s.notificacionesParaTodos.notis$ ▷ $\mathcal{O}(1)$
3: $n \leftarrow s.notificacionesParaTodos.cantidadVistos[cid-1]$ ▷ $\mathcal{O}(1)$
4: **mientras** ¬ iVacía(notisPers) **hacer** ▷ $\mathcal{O}$(#notificacionesPersonales)
5: desencolar(notisPers) ▷ $\mathcal{O}(1)$
6: **end mientras**
7: **para** i = n .. ilongitud(notisTodos)-1 **hacer** ▷ $\mathcal{O}$(#notificaciones para todos - #notificaciones de todos ya vistas por el jugador)
8: s.notificacionesParaTodos.cantidadVistos[cid-1]++
9: **end para**

Complejidad: $\mathcal{O}(A + B)$, donde A es #notificacionesPersonales y B es #notificaciones para todos - #notificaciones de todos ya vistas por el jugador

---

IRECIBIRMENSAJENUEVO (**in/out** s : `serv`, **in** cid : `nat`, **in** o : `ocurrencia`)

1: $fichasAnteriores \leftarrow ifichasDeJugador(s.juego, cid)$ ▷ $\mathcal{O}(1)$
2: $esValida \leftarrow iempez?(s) \wedge$ (cid == s.juego.turno_de) ▷ $\mathcal{O}(1)$
3: $palabrasFormadas \leftarrow Cola(lista(letra))Vacia()$ ▷ $\mathcal{O}(1)$
4: **si** ESVALIDA **entonces**
5: $jugValida.palabrasFormadas \leftarrow jugadaValida(s.juego, o)$
6: $fichasAnteriores \leftarrow s.juego.jugadores.fichas$ ▷ $\mathcal{O}(1)$
7: $puntajePrevio \leftarrow ibucarFichas(s.juego, o, palabrasFormadas)$ ▷ $\mathcal{O}$(K)
8: $fichasPostJugada \leftarrow s.juego.jugadores.fichas$ ▷ $\mathcal{O}(1)$
9: iAgregarAtras(s.notificacionesParaTodos.notis, iUbicar(cid, o)) ▷ $\mathcal{O}$(ln), donde ln = long(s.notificacionesParaTodos.noti)
10: iAgregarAtras(s.notificacionesParaTodos.notis, iSumarPuntos(cid, ipuntaje(s.juego, cid-1, out) - puntajePrevio)) ▷ $\mathcal{O}$(ln)
11: iAgregarAtras(s.notificacionesParaTodos.notis, iTurnoDe(s.j.turno_de)) ▷ $\mathcal{O}$(ln)
12: $fichasRepuestas \leftarrow ivacio()$ ▷ $\mathcal{O}(1)$
13: **para** i = 0 .. longitud(fichasPostJugada)-1 **hacer** ▷ $\mathcal{O}$(fpj), donde longitud(fichasPostJugada)
14: **si** FICHASANTERIORES[I] <FICHASPOSTJUGADA[I] **entonces**
15: $cant \leftarrow fichasPostJugada[i] - fichasAnteriores[i]$ ▷ $\mathcal{O}(1)$
16: iAgregarRapido(ord^{-1}(i), cant) ▷ $\mathcal{O}(1)$
17: **end si**
18: **end para**
19: iencolar(s.notificacionesPersonales[#conectados-1], iReponer(fichasRepuestas) ▷ $\mathcal{O}(1)$
20: **else**iencolar(s.notificacionesPersonales[cid-1], iMal) ▷ $\mathcal{O}(1)$
21: **end si**

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

IEMPEZOJUEGO(**in** s : `serv`) $\longrightarrow$ res : `bool`

1: $res \leftarrow s.\#jugadoresEsperados = s.\#jugadoresConectados$ ▷ $\mathcal{O}(1)$

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