DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico de Diseño - Scrabble

Integrante	LU	Correo electrónico
Malena Sol, Alamo	1620/21	malusalamo@gmail.com
Klimkowski, Victoria	1390/21	02vicky02@@gmail.com
Laria, Jeremias	1329/21	jeremiaslaria7@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Extras

1.1. Cola

Interfaz

Extiende el modulo cola

PROXIMOSN(**in** r : `cola(letra)`, **in** $cantFichas$: `nat`) $\rightarrow$ res : `vector(letra)`
Pre $\equiv$ {cantFichas $\leq tamanio(r)$}
Post$\equiv$ $\{vectorIgualMulticonj(proximosN(r, cantFichas), res)\}$
Complejidad: $O(cantFichas^2)$
Descripción: Nos extrae los primeros n elementos de una cola.
Aliasing: Se pasa la cola por referencia.

Algoritmos

---

iproximosN(**in** r : `estr`, **in** $cantFichas$: `nat`) $\rightarrow$ res : vector(letra)

```
1: res = []
2: for(int i=0; i<cantFichas; i++):
3:     agregarAtras(res, proximo(r))
4:     desencolar(r)
5: return res
```

1.2. Letra

letra es char

1.3. Palabra

palabra es secu(char)

2. Módulos de referencia

2.1. Módulo Tablero

Interfaz

se explica con: TABLERO

géneros: `tablero`

Operaciones básicas de tablero

CREARTABLERO(**in** n : `Nat`) $\rightarrow$ res : `tablero`
Pre $\equiv$ {n $\neq 0$}
Post$\equiv$ $\{res =_{\text{obs}} nuevoTablero(n)\}$
Complejidad: $O(n^2)$
Descripción: Crea un tablero vacío de dimension $n \times n$.

PONERLETRA(**in/out** tab : `tablero`, **in** i : `Nat`, **in** j : `Nat`, **in** l : `letra`)
Pre $\equiv$ $\{0 \leq i, j < tamanio(tab) \land \neg(hayLetra?(tab, i, j)) \land tab = tab_0\}$
Post$\equiv$ $\{tamanio(tab) =_{\text{obs}} tamanio(tab_0) \land hayLetra?(tab, i, j) =_{\text{obs}} true \land_L letra(tab, i, j) =_{\text{obs}} l\}$
Complejidad: $O(1)$
Descripción: En la posición (i,j) de la matriz se coloca una letra.
Aliasing: Si, en tab. Es mutable

UBICARFICHASTABLERO(**in/out** *tab*: `tablero`, **in** *o*: `ocurrencia`)
Pre ≡ {celdasLibres(tab,o) ∧ $tab = tab_0$}
Post≡ {$tamanio(tab) =_{\text{obs}} tamanio(tab_0) \wedge (\forall l : Nat)(l < long(o) \Rightarrow_{\text{L}} letra(tab, pi0(o[l]), pi1(o[l])) =_{\text{obs}} pi2(o[l]))$}
Complejidad: $O(long(o))$
Descripción: Se ubican todas las letras de la ocurrencia en sus respectivas posiciones.
Aliasing: Pasamos tab por referencia y es mutable

SACARLETRA(**in/out** *tab*: `tablero`, **in** *i*: `Nat`, **in** *j*: `Nat`, **in** *l*: `letra`)
Pre ≡ $\{0 \leq i, j < tamanio(tab) \wedge (hayLetra?(tab, i, j)) \wedge tab = tab_0\}$
Post≡ $\{tamanio(tab) =_{\text{obs}} tamanio(tab_0) \wedge hayLetra?(tab, i, j) =_{\text{obs}} false\}$
Complejidad: $O(1)$
Descripción: En la posición (i,j) de la matriz se saca una letra.
Aliasing: Si, en tab. Es mutable

OBTENERCONTENIDO(**in** *tab*: `tablero`, **in** *i*: `Nat`, **in** *j*: `Nat`) → *res* : `tupla(Bool, Letra)`
Pre ≡ $\{0 \leq i, j < tamanio(tab)\}$
Post≡ $\{res_0 =_{\text{obs}} hayLetra?(tab, i, j) \wedge_L (hayLetra?(tab, i, j) \rightarrow_L \text{res}_1 =_{\text{obs}} letra(tab, i, j)\}$
Complejidad: $O(1))$
Descripción: Se accede a la posición (i,j) del tablero.
Aliasing: Si, pero no es mutable.

CELDALIBRE(**in** *celda*: `tupla(nat,nat,letra)`, **in** *t*: `tablero`) → *res* : `bool`
Pre ≡ $\{\pi_0(celda), \pi_1(celda) < tamanio(t)\}$
Post≡ $\{res =_{\text{obs}}$ hayLetra?(t, $\pi_0(celda), \pi_1(celda))\}$
Complejidad:O(1)
Descripción: Se pregunta si en la posición (i,j) del tablero hay una letra
Aliasing: res es modificable si y solo si tab es modificable

secuIndex(s,i)= if i=0 then prim(s) else secuIndex(fin(s), i-1)

CELDASLIBRES(**in** *t*: `tablero`, **in** *o*: `ocurrencia`) → *res* : `bool`
Pre ≡ $\{(\forall i : nat)(i < long(o) \Rightarrow_{\text{L}} \pi_0(secuIndex(o, i)) < tamanio(t) \wedge \pi_1(secuIndex(o, i)) < tamanio(t))\}$
Post≡ $\{res =_{\text{obs}} (\forall i : nat)(i < long(s) \Rightarrow_{\text{L}} \neg hayLetra?(t, \pi_0(secuIndex(s, i))))\}$
Complejidad:O(long(o))
Descripción: Verificamos si todas las posiciones de la ocurrencia estan libres en el tablero.
Aliasing: Si, pero no es modificable.

TAMANIOTABLERO(**in** *t*: `tablero`) → *res* : `nat`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ tamanio(t)}
Complejidad: $O(1)$

PALABRAHORIZONTAL(**in** *o*: `ocurrencia`, **in** *t*: `tablero`) → *res* : `bool`
Pre ≡ {True}
Post ≡ $\{res =_{\text{obs}} true \iff secuenciaFormaPalabraHorizontal(ordenadaHorizontal(o))\}$
Complejidad: $O(Lmax))$
Descripción: Chequea si una palabra es horizontal

PALABRAVERTICAL(**in** *o*: `ocurrencia`, **in** *t*: `tablero`) → *res* : `bool`
Pre ≡ {True}
Post ≡ $\{res =_{\text{obs}} true \iff secuenciaFormaPalabraVertical(ordenadaVertial(o))\}$
Complejidad: $O(Lmax))$

Nota: ordenadaHorizontal ordena de menor a mayor según el primer elemento de la tupla(nat,nat,letra) y ordenadaVertical ordena de menor a mayor según el segundo elemento de la tupla(nat,nat,letra)

Representación

Representación de tablero

Un tablero contiene n x n casillas. Cada casilla puede o no contener una letra.

`tablero` **se representa con** `estr`

donde `estr` es `tupla`(*tamanio*: `nat`, *tab*: `vector(vector(tupla(bool, letra)))`)
Donde letra es char

Rep : estr e $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ $e.tamanio > 0 \wedge e.tamanio = long(e.tab) \wedge_L (\forall i : Z)(0 \leq i < e.tamanio \rightarrow_L$ long(e.tab[i])=e.tamanio))

Abs : estr e $\longrightarrow$ tablero {Rep(e)}

Abs(e) $\equiv$ $(\forall t : tablero)(Abs(e) =_{\text{obs}} t \iff e.tamanio =_{\text{obs}} tamanio(t) \wedge_L (\forall i, j : nat)(i, j < e.tamanio \Rightarrow_L hayLetra?(i, j, t) =_{\text{obs}} e.tab[i][j]_0) \wedge (\forall i, j : nat)(i, j < t.tamanio \wedge_L hayLetra?(i, j, t) \Rightarrow_L letra(i, j, t) =_{\text{obs}} \pi_2(t.tab[i][j])))$

Algoritmos

---

iCrearTablero(**in** *tamanio*: `nat`) $\rightarrow$ *res* : estr

```
1: vector(vector((bool, letra))) fila ← vacio
2: vector((bool, letra)) iFila ← vacio
3:
4: for (i = 0; i < tamanio; i++) :
5:     iFila.agregarAtras(tupla(false,''))
6:
7: for (i = 0; i < tamanio; i++) :
8:     fila.agregarAtras(iFila)
9:
10: res ← tupla(tamanio, fila)
11: return res
```

---

iPonerLetra(**in/out** *tablero*: `estr`, **in** i: `nat`, **in** j: `nat`, **in** l: `letra`)

```
1: tablero.tab[i][j] ← (true, l)
2:
```

---

iubicarFichasTablero(**in/out** *tablero*: `estr`, **in** o: `ocurrencia`)

```
1: for(k = 0; k < o.size(); k++)
2:     tablero.tab[get < 0 > o[k]][get < 1 > o[k]] = (true, get < 2 > o[k])
```

---

iSacarLetra(**in/out** *tablero*: `estr`, **in** i: `nat`, **in** j: `nat`, **in** l: `letra`)

```
1: tablero.tab[i][j] ← (false,'')
```

---

iObtenerContenido(**in** *tablero*: `estr`, **in** i: `nat`, **in** j: `nat`) $\rightarrow$ *res* : tupla(bool, letra)

```
1: return tablero.tab[i][j]
```

---

iCeldaLibre(**in** t: `estr`, **in** *celda*: `tupla(nat, nat, letra)` $\rightarrow$ *res* : bool

```
1:  return get < 0 > (ObtenerContenido(t, get < 0 > (celda), get < 1 > (celda)))
```

---

iCeldasLibres(**in** t: `estr`, **in** o: `ocurrencia`) → res : bool

```
1: res ← true
2: if (not esVacio?(o)) :
3:     for( i = 0; i < longitud(o); i++) :
4:         res=res and celdaLibre(t,o[i])
5: return res
```

iTamanioTablero(**in** t: `estr`) *to* res : nat

```
1: return t.tamanio
```

ipalabraHorizontal(**in** o: `ocurrencia`) → res : bool

```
1: int n = get<0>o[0]
2: for(int i; i<o.size();i++)
3:     if(get < 0 > o[i] ≠ n)
4:         return false
5: return true
```

ipalabraVertical(**in** o: `ocurrencia`) → res : bool

```
1: int n = get<1>o[0]
2: for(int i; i<o.size();i++)
3:     if(get < 1 > o[i] ≠ n)
4:         return false
5: return true
```

2.2. Módulo conjunto

Interfaz

se explica con: CONJ(PALABRA)

géneros: `conjunto`

Operaciones básicas de conjunto

NUEVOCONJUNTO(**in** c: `conjLin(palabra)`) → res : `conj(palabra)`
Pre ≡ {True}
Post ≡ {c =$_{\text{obs}}$ res}

PERTENECEACONJUNTO(**in** p: `palabra`, **in** c: `conj(palabra)`) → res : `Bool`
Pre ≡ {¬*vacio*?(p)}
Post≡ {$res =_{\text{obs}} p \in c$}
Complejidad: $O(Lmax)$
Descripción: Chequea si una palabra pertenece al conjunto

Representación

Representación de conj(palabra)

Representamos un conjunto de palabras a través de un conjunto de letras. Tomamos Lmax como la palabra más larga representada en el conjunto, que en la estructura definimos como altura.

`conj(palabra)` **se representa con** `estr`

donde `estr` es `tupla`(*raiz*: `puntero(nodo)`, *altura*: `Nat`)

donde ndo es `tupla`(*flag*: `bool`, *hijos*: `vector(nodo)`)

Rep : estr c $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$ $long(c.raiz \rightarrow hijos) = tamanioAlfabeto \wedge nodosConTamanioCorrecto(c.raiz \rightarrow hijos) \wedge flagEnHojas(c.raiz) \wedge longCorrecta(c)$

longCorrecta: conj ->bool

longCorrecta(c) $\equiv altura(c.raiz) =_{\text{obs}} c.altura$

altura: puntero(nodo) ->nat

altura(c.raiz) $\equiv 1 + maxAlturaHijos(c.raiz \rightarrow hijos)$

maxAlturaHijos: secu(nodo) ->nat

$maxAlturaHijos(s) \equiv$ if vacia?(s) then 0 else max(altura(prim(s)),maxAlturaHijos(fin(s)))

nodosConTamanioCorrecto: secu(puntero) ->bool

nodosConTamanioCorrecto(v) $\equiv (\forall i : nat)(i < long(v) \wedge v[i] \neq NULL \rightarrow_L long(v[i] \rightarrow hijos) = tamanioAlfabeto)$

flagEnHojas: nodo $\rightarrow$ *bool*

flagEnHojas(n) $\equiv$if todosNULL(n.hijos) then

$n.flag = true$

else

$flagEnHojasHijos(v.hijos)$

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

Abs(c) $\equiv (\forall conj : conjunto(palabra))(Abs(c) =_{\text{obs}} conjunto(palabra) \Leftrightarrow (\forall p : palabra)(p \in conj \Leftrightarrow perteneceAconjunto(p, estr.raiz))$

perteneceAconjunto: palabra x conjunto ->bool
perteneceAconjunto(p,nodo) $\equiv$if vacio?(p) then

nodo.flag

else

if nodo->hijos[ord(prim(p))] == NULL then

false

else

perteneceAconjunto(fin(p),nodo->hijos[ord(prim(p))])

Algoritmos

inuevoConjunto(**in** *pl* : `conjLin(palabra)`) → *res* : estr

```
1:  res.raiz = new Nodo()
2:  Nodo* actual = res.raiz;
3:  for(Palabra p: pl)
4:     for(int l=0;l<p.size()-1;l++)
5:        if(actual->Nodo.hijos[ord(l)]== nullptr)
6:          actual->Nodo.hijos[ord(l)] = new Nodo()
7:          actual = actual->Nodo.hijos[ord(l)]
8:        else
9:          actual = actual->Nodo.hijos[ord(l)]
10:    actual->Nodo.flag = true
11:    if(p.size()>estr.altura)
12:       res.altura=p.size();
13: return res =0
```

iperteneceAconjunto(**in** *p* : `palabra`, **in** *t* : `estr`) → *res* : bool

```
1:  if (estr.raiz->hijos[ord(p[0])] == NULL) then
2:       false
3:  else
4:        nodo nodo = estr.raiz->hijos[ord(p(0))]
5:        for(i=1, i<|p|-1,i++)
6:           if nodo.hijos[ord(p[i])] == NULL then
7:                false
8:           else
9:                nodo = nodo.hijos[ord(p[i])]
10: return nodo.flag
```

2.3. Módulo Variante

Interfaz

se explica con: VARIANTE

género: `variante`

Operaciones básicas de variante

CREARVARIANTE(**in** *tamanioTab*: `nat`, **in** *fichas*: `nat`, **in** *puntajeLetra*: `vector(nat)`, **in** *palabrasLegitimas*: `conjunto(palabra)`) → *res* : `Variante`
Pre ≡ {tamanioTab>0 ∧$fichas > 0$}
Post≡ {$res =_{\text{obs}} nuevaVariante(tamanioTab, fichas, puntajeLetra, palabrasLegitimas$}
Complejidad: $O(1)$
Descripción: Crea variante
Aliasing: Sí en todas. Las pasamos por referencia y son mutables

TAMANIOTABVARIANTE(**in** *v* : `Variante`) → *res* : `Variante`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} tamanioTab(v)$}
Complejidad: $O(1)$
Descripción: Ve el tamanio por referencia
Aliasing: Sí. Las pasamos por referencia y no son mutables

CANTFICHAS(**in** *v* : `Variante`) → *res* : `Variante`

Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} fichas(v)\}$
Complejidad: $O(1)$
Descripción: Ve la cantidad de fichas que tiene cada jugador por referencia. A veces puede ser llamada como #fichas por problemas con latex
Aliasing: Sí. Las pasamos por referencia y no son mutables

PUNTAJELETRA(**in** $vector(nat)$: `Variante`) $\rightarrow$ res : `Variante`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} fichas(v)\}$
Complejidad: $O(1)$
Descripción: Ve la cantidad de fichas que tiene cada jugador por referencia
Aliasing: Sí. Las pasamos por referencia y no son mutables

ESPALABRALEGITIMA(**in** pal : `palabra`, **in** $v >$: `Variante`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} palabraLegitima?(pal, v)\}$
Complejidad: $O(1)$
Descripción: Ve si una palabra es legitima

Representación

Representación de variante
Una variante es información sobre alguna versión

`variante` **se representa con** `estr`

donde `estr` es `tupla`(*tamanioTab*: `nat`, *cantFichas*: `nat`, *puntajeLetra*: `vector(nat)`, *palabrasLegitimas*: `conjunto(letra)`)

Rep : estr e $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$ $e.tamanioTab > 0 \land e.cantFichas > 0$
Abs : estr v $\longrightarrow$ variante {Rep(v)}
Abs(v) $\equiv$ $(\forall var : variante)$
(Abs(v) $=_{\text{obs}}$ variante $\Leftrightarrow$ $v.tamanioTab =_{\text{obs}} tamanioTablero(var) \land$
$v.cantFichas =_{\text{obs}} fichas(var) \land$
$(a : letra)(def?(v.puntajeLetra, a) \Rightarrow_{\text{L}} obtener(a, v.puntajeLetra) =_{\text{obs}} puntajeLetra(a, var)) \land$
$(a : palabra)(palabraLegitima?(a, var) =_{\text{obs}} av.palabrasLegitimas))$

Algoritmos

---

icrearVariante(**in** $tamaoTab$: `nat`, **in** $fichas$: `nat`, **in** $puntajeLetra$: `vector(nat)`, **in** $palabrasLegitimas$: `conjunto(palabra)` $\rightarrow$ res : estr

1: $res.tamanioTab = tamanioTab$
2: $res.fichas = fichas$
3: $res.puntajeLetra = puntajeLetra$
4: $res.palabrasLegitimas = palabrasLegitimas$
5: **return** res

---

itamanioTabVariante(**in** v : `estr`) $\rightarrow$ res : nat

1: res = v.tamanioTab

---

ifichas(**in** v : `estr`) $\rightarrow$ res : nat

1: res = v.fichas

---

ipuntajeLetra(**in** a : `letra`,**in** v : `estr`) → res : nat
1: res = obtener(a, v.puntajeLetra)

iesPalabraLegitima(**in** pal : `palabra`,**in** v : `estr`) → res : bool
1: res = perteneceAconjunto(pal, v.palabrasLegitimas)

2.4. Módulo Juego

Interfaz

se explica con: JUEGO

géneros: `juego`

Operaciones básicas de juego

NUEVOJUEGO(**in** $cantJug$: `nat`, **in** v : `variante`, **in** r : `cola(letra)`) → res : `juego`
Pre ≡ {cantJug>0 ∧ $tamanio(r) \geq tamanioTablero(v) * tamanioTablero(v) + cantJug * cantFichas(v)$}
Post≡ {$res =_{\text{obs}} nuevoJuego(cantJug, v, r)$}
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: "N" en la complejidad = tamanio del tablero de la variante, "K" = cantJug, $|\Sigma|$ = cant letras en el alfabeto, "F" es la cant fichas por jugador (dado en la variante)

VERVARIANTE(**in** j : `juego`) → res : `variante`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} variante(j)$}
Complejidad: $O(1)$
Descripción: Ve la variante
Aliasing: Si y no es mutable. Se pasa por referencia

VERTABLERO(**in** j : `juego`) → res : `tablero`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} tablero(j)$}
Complejidad: $O(1)$
Descripción: Ve el tablero
Aliasing: Si y es mutable el tablero. Lo veo por referencia

TURNO(**in** j : `juego`) → res : `nat`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} turno(j)$}
Complejidad: $O(1)$
Descripción: Ve el turno
Aliasing: Si y no es mutable

TAMANIOTABLERO(**in** j : `juego`) → res : `nat`
Pre ≡ {True}
Post ≡ {$res =_{\text{obs}} tamanio(j)$}
Complejidad: $O(1)$
Descripción: Me devuelve el tamanio del tablero.

JUGADAVALIDA(**in** o : `ocurrencia`, **in** j : `juego`) → res : `bool`
Pre ≡ {True}
Post ≡ { $res =_{\text{obs}}$ jugadaValida?(j,o)}
Complejidad: $O(lmax^2)$
Descripción: Devuelve true si y solo si: la ocurrencia es menor a la logitud de la palabra valida mas larga, que el jugador tenga en su poder las fichas que quiere poner (lo cual no estaba en el TAD pero nos parecía sumamente necesario), que las celdas a ocupar existan y estén libres, que la ocurrencia sea vertical u horizonal y que todas las palabras a formar sean validas

Aliasing: Sí, es mutable el tablero del juego pero no deberían generarse cambios al terminar la funcipon

secuAmulticonj(s) = if vacio?(s) then vacio else Ag(prim(s), secuAmulticonj(fin(s)))

UBICARFICHAS(**in/out** j : `juego`, **in** o: `ocurrencia`)
Pre ≡ {celdasLibres(j.tablero,o), **in** jug: `nat` }
Post ≡ $\{(\forall i, j : Nat)(min(\pi_0(o) \leq i \leq max(\pi_0(o) \wedge (min(\pi_1(o) \leq j \leq max(\pi_1(o) \longrightarrow_L (hayLetra?(j.tablero, i, j) =_{\text{obs}} true \wedge_L letra(tab, i, j) =_{\text{obs}} \pi_2(o))\}$
Complejidad: $O(long(o))$
Descripción: En la posición (i,j) de la matriz se coloca una letra.
Aliasing: Si, en tab. Alteramos la estructura de juego cambiando el historial de jugadas

OBTENERPUNTAJE(**in** i: `nat`, **in** j: `juego`) → res : `nat`
Pre ≡ {True}
Post ≡ $\{res =_{\text{obs}} puntaje(j, i)\}$
Complejidad: $O(1)$
Descripción: Me devuelve el puntaje del jugador. Por cambios en la estructura, ObtenerPuntaje termina siendo O(1).

CANTIDADFICHASLETRA(**in** j: `estr`, **in** cid: `nat`, **in** l: `letra`) → res : `nat`
Pre ≡ {cid<j.cantJugadores}
Post ≡ $\{res =_{\text{obs}} cantApariciones(l, fichas(cid, j))\}$
Complejidad: $O(1)$
Descripción: Nos devuelve la cantidad que tiene un jugador de determinada ficha.

Representación

Representación de juego

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

donde `estr` es `tupla`(*variante*: `variante`, *cantJugadores*: `nat`, *repositorio*: `cola(letra)`, *tablero*: `tablero`, *puntaje*: `vector(nat)` , *cantidadLetraJugador*: `vector(vector(nat))`, *turno*: `nat`)

Donde letra es string

Rep : estr j $\longrightarrow$ bool

Rep(e) ≡ true $\iff$ $j.cantJugadores > 0 \wedge tamanio(j.tablero) =_{\text{obs}} tamanioTablero(j.variante) \wedge long(j.cantidadLetraJugador) =_{\text{obs}} j.cantJugadores \wedge long(j.puntaje) =_{\text{obs}} j.cantJugadores \wedge long(j.cantLetraJugador) =_{\text{obs}} j.cantJugadores \wedge (\forall i : nat)(i < j.cantJugadores \Rightarrow_{\text{L}} sumaLetras(cantLetraJugador[i]) =_{\text{obs}} cantFichas(j.variante)$

sumaLetras: secu(nat) ->nat

sumaLetras(s) ≡if vacio?(s) then 0 *else*

prim(s) + sumaLetras(fin(s))

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

Abs(j) ≡ jue:juego |(j.variante $=_{\text{obs}}$ variante(jue) $\wedge j.cantJugadores =_{\text{obs}} jugadores(jue) \wedge j.repositorio =_{\text{obs}} repositorio(jue) \wedge j.tablero =_{\text{obs}} tablero(jue) \wedge (\forall i : nat)(i < long(cantidadLetraJugador) \Rightarrow_{\text{L}} vectorIgualMulticonj(j.fichas[i], fichas(jue))) \wedge (\forall i : nat)(0 \leq i < long(j.puntaje) \longrightarrow_L puntaje(jue, i) =_{\text{obs}} j.puntaje[i]))$

vectorIgualMulticonj: secu(letra) x multiconj(letra) → *bool*

vectorIgualMulticonj(s,m) ≡
if long(s) $=_{\text{obs}}$ m then
if vacio?(s)
then true
else
if prim(s) ∈ m
then vectorIgualMulticonj(fin(s), m-prim(s))

else

false

else

false

Algoritmos

---

iNuevoJuego(**in** *cantJug* : `nat`, **in** *v* : `variante`, **in** *r* : `cola(letra)`)→ *res* : estr

```
1:  res.variante = v
2:  res.tablero = crearTablero(tamanioTabVariante(v))
3:  res.cantJugadores = cantJug
4:  res.repositorio = r
        /*Inicializa Puntaje y Fichas */
5:  res.turno = 0
6:  res.fichas = []
7:  res.puntaje = []
8:  for(int jugador = 0; jugador<cantJug, jugador++)
9:      agregarAtras(res.fichas, [])
10:     agregarAtras(res.puntaje, 0)
        /*En el i-jugador creamos cuantas letras tiene*/
11:     for(int letraDeAlfabeto = 0; letraDeAlfabeto <tamanioAlfabeto; letrADeAlfabeto++)
12:         agregarAtras(res.fichas[jugador],0)
13: for(int i=0; i <cantJug; i++)
14:     for(int j=0; j<cantFichas; j++)
15:         res.fichas[i][ord(proximo(r))] = res.fichas[i][ord(proximo(r))]+1
16:         desencolar(r)
```

---

iVerVariante(**in** *j* : `estr`)→ *res* : variante

return *j.variante*

---

iVerTablero(**in** *j* : `estr`)→ *res* : tablero

return *j.tablero*

---

iVerTurno(**in** *j* : `estr`)→ *res* : nat

return *j.turno*

---

iTamanioTablero(**in** *j* : `estr`)→ *res* : nat

return *tamanio*(*j.tablero*)

---

iJugadaValida(**in** o: `ocurrencia`, **in** j: `estr`, **in** jug: `nat`) → res : nat

```
1: /* Chequeo si la longitud de la ocurrencia es menor a la palabra legitima más larga */
2:
3: if(|o| > altura(palabrasLegitimas(j.variante)))
4:    returnfalse
5:
6: /* Verifico que las fichas esten en mano, que las celdas estén libres y que el rango pertenezca al tablero */
7: /* La complejidad es de O(m) con m acotado por lmax, por lo que es O(lmax) */
8: for(i=0, i<o.size(), i++)
9:    if(j.cantidadLetrasJugador[jug][ord(get<2>o[i])] == 0 || !celdaLibre(j.tablero) ||
10:   get < 0 > o[i] ≥ tamanioTablero(j.tablero) || get < 0 > o[i] < 0 ||
11:   get < 1 > o[i] ≥ tamanioTablero(j.tablero) || get < 1 > o[i] < 0)
12:       return false
13:
14: /*Chequeo que sea horizontal o vertical. O(lmax+lmax)*/
15: bool palHor = palabraHorizontal(o)
16: bool palVer = palabraVertical(o)
17: if(!palHor(o) || !palVer(o))
18:       return false
19:
20: /*Pongo las fichas en el tablero O(lmax)*/
21: for(i=0, i<o.size(), i++)
22:    ponerLetra(j.tablero,get<0>o[i],get<1>o[i],get<2>o[i])
23:
24: /* Chequeo palabras validas si es horizontal. O(lmax^2) */
25: if(palHor)
26:    if(esValidaHorizontal(j.tablero,o[0]))
27:    res = true
28:       for(i=0, i<o.size(), i++)
29:             if(!esValidaVertical(j.tablero,o[i]))
30:                res = false
31:
32: /* Chequeo palabras validas si es vertical. O(lmax^2) */
33: if(palVer)
34:    if(esValidaVertical(j.tablero,o[0]))
35:    res = true
36:       for(i=0, i<o.size(), i++)
37:             if(!esValidaHorizontal(j.tablero,o[i]))
38:                res = false
39:
40: /* Saco las fichas. O(lmax) */
41: for(i=0, i<o.size(), i++)
42:    sacarLetra(j.tablero,get<0>o[i],get<1>o[i],get<2>o[i])
43:
44: return res
```

[Comentario: Se chequea como primer paso que la ocurrencia este acotada por lmax (longitud de la palabra valida mas larga). En el caso de que no lo sea, la jugada es invalida. En el caso de que lo sea, la ocurrencia esta acotada por lmax. Después verifica que el jugador tenga todas las fichas que quiere poner en su dominio (lo cual no estaba en el TAD pero nos parecía sumamente necesario) y que las celdas que quiere ocupar existan y estén libres. Después chequea que la ocurrencia tenga todas fichas horizontales o todas verticales. Luego pone las fichas. Después, es es horizontal, chequea que la palabra formada sea válida y que cada una de las palabras verticales también lo sea. Análogo para cuando la ocurrencia es vertical. Si alguna no lo fuese, no es una jugada válida. Finalmente saca todas las fichas]

iUbicarFichas(**in**/**out** j : estr, **in** o : ocurrencia, **in** jug : nat)

```
1: ubicarFichasTablero(j.tablero,o)
2: for(Celda c:o)
3:     j.cantLetraJugador[j.turno][ord(get<2>(c)]= j.cantLetraJugador[j.turno][ord(get<2>(c))] - 1;
4:     j.[j.turno][ord(proximo(j.repositorio)] = cantLetraJugador[j.turno][ord(proximo(j.repositorio))] + 1
5:   desencolar(j.repositorio)
6: j.puntaje[j.turno] = j.puntaje[j.turno] + calcularPuntaje(o)
7: j.turno = j.turno + 1 mod j.cantJugadores
```

[De esta manera, conseguimos el puntaje en O[1] pero la complejidad de ubicar termina siendo O($m^2 * Lmax$)))]

iEsValidaHorizontal(**in** j : estr, **in** *celda* : tupla(nat,nat,letra)) $\rightarrow$ *res* : bool

```
1: t = j.tablero
2: pivote = celda
3: inicio = pivote
4: fin = pivote
5: i = get<0>(inicio)
6: j = get<0>(fin)
7:
8: while(i ≥ 0 and not celdaLibre(t, inicio)) :
9:     inicio = tupla(i, get<1>(celda), get<2>(obtenerContenido(t, i, get<1>(celda))))
10:    i − −
11:
12: while(j < tamanioTablero(t) and not celdaLibre(t, fin)) :
13:    fin = tupla(j, get<1>(celda), get<2>(obtenerContenido(t, i, get<1>(celda))))
14:    j + +
15: palabra = []
16:
17: for (k = i; k < j; k + +) :
18:    agregarAtras(palabra, get<2>(obtenerContenido(t, k, get<1>(celda))))
19:
20: if (esPalabraLegitima(palabra, variante(j)))
21:    return true
22: else :
23:    return false
```

iEsValidaVertical(**in** j : `estr`, **in** *celda* : `tupla(nat,nat,letra)`)→ *res* : bool

```
1: t = j.tablero
2: pivote = celda
3: inicio = pivote
4: fin = pivote
5: i = get<1>(inicio)
6: j = get<1>(fin)
7:
8: while(i ≥ 0 and not celdaLibre(t, inicio)) :
9:    inicio = tupla(get<0>(celda), i, get<2>(obtenerContenido(t, i, get<1>(celda))))
10:   i − −
11:
12: while(j < tamanioTablero(j.tablero) and not celdaLibre(t, fin)) :
13:   fin = tupla(get<1>(celda), j, get<2>(obtenerContenido(t, i, get<1>(celda))))
14:   j + +
15:
16: palabra = []
17:
18: for (k = i; k < j; k + +) :
19:   agregarAtras(palabra, get<2>(obtenerContenido(t, get<1>(celda), k)))
20:
21: if (esPalabraLegitima(palabra, variante(j)))
22:   return true
23: else :
24:   return false
```

iObtenerPuntaje(**in** *jug* : `nat`, **in** j : `estr`)→ *res* : nat

```
1: return j.puntaje[jug]
```

iCalcularPuntaje(**in** j : `estr`, **in** o : `ocurrencia`)→ *res* : nat

```
1: puntaje = 0
2:
3: for(i<|o|, i++)
4:    puntaje=puntaje + puntajeLetra(o[i],j.variante)
5:
6: if(palabraHorizontal(o))
7:    for(i<|o|, i++)
8:       puntaje=puntaje + sumarPuntajesVerticales(o[i],j)
9: else
10:   for(i<|o|,i++)
11:      puntaje=puntaje+sumarPuntajesHorizontales(o[i],j)
12: return puntaje
```

iSumarPuntajesVerticales(**in** *posicion* : `tupla(nat,nat,letra)`, **in** *j* : `estr`)→ *res* : nat

```
1: /* Creamos conjunto con el elemento posicion */
2: conjunto(tupla(nat,nat,letra)) palabra = posicion
3:
4: for(i<Lmax)
5:     if((0 ≤ i < n and get < 0 > (j.tablero[get < 0 > posicion][get < 1 > posicion − i]) == true)
6:       agregar(palabra,j.tablero[get<0>posicion][get<1>posicion-1]
7:
8:     if((0 ≤ i < n and get < 0 > (j.tablero[get < 0 > posicion][get < 1 > posicion + i]) == true)
9:       agregar(palabra,j.tablero[get<0>posicion][get<1>posicion+1]
10:
11: puntajepalabra = 0
12: for(i<|palabra|,i++)
13:     puntajePalabra = puntajePalabra + puntajeLetra(palabra[i],j.variante)
14:
15: return puntajePalabra
```

iSumarPuntajesHorizontales(**in** *posicion* : `tupla(nat,nat,letra)`, **in** *j* : `estr`)→ *res* : nat

```
1: /* Creamos conjunto con el elemento posicion */
2: conjunto(tupla(nat,nat,letra)) palabra = posicion
3:
4: for(i<Lmax)
5:     if((0 ≤ i < n and get < 0 > (j.tablero[get < 0 > posicion − i][get < 1 > posicion]) == true)
6:       agregar(palabra,j.tablero[get<0>posicion-i][get<1>posicion]
7:
8:     if((0 ≤ i < n and get < 0 > (j.tablero[get < 0 > posicion + i][get < 1 > posicion]) == true)
9:       agregar(palabra,j.tablero[get<0>posicion+i][get<1>posicion]
10:
11: puntajepalabra = 0
12: for(i<|palabra|,i++)
13:     puntajePalabra = puntajePalabra + puntajeLetra(palabra[i],j.variante)
14:
15: return puntajePalabra
```

iCantidadDeFichasLetra(**in** *j* : `juego`,**in** *cid* : `nat`, **in** *l* : `letra`)→ *res* : nat

```
1: return j.cantidadLetraJugador[cid][ord(l)]
```

2.5. Módulo Notificacion

Interfaz

se explica con: Notificaion

género: `Notificación`

Operaciones básicas de notificación

IdCliente(**in** *cid* : `nat`) → *res* : `Notificacion`
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} IdCliente(cid)$}
Complejidad: $O(1)$

Empezar(**in** *tamanio* : `nat`) → *res* : `Notificacion`
Pre ≡ {n>0}
Post ≡ {$res =_{\text{obs}} Empezar(n)$}
Complejidad: $O(1)$

TurnoDe(**in** *cid*: nat) → *res* : Notificacion
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} TurnoDe(cid)$}
Complejidad: $O(1)$

Ubicar(**in** *cid*: nat, **in** *o*: ocurrencia) → *res* : Notificacion
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} Ubicar(cid, o)$}
Complejidad: $O(1)$

Reponer(**in** *f*: multiconj(letra)) → *res* : Notificacion
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} Reponer(f)$}
Complejidad: $O(1)$

SumaPuntos(**in** *cid*: nat, **in** *puntos*: nat) → *res* : Notificacion
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} SumaPuntos(cid, puntos)$}
Complejidad: $O(1)$

Mal() → *res* : Notificacion
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} Mal$}
Complejidad: $O(1)$

Representación

Representación de Notificación
Notificaciones al servidor

Notificacion **se representa con** estr

donde estr es tupla(*tipoNotif*: string, *idCliente*: nat, *cantTablero*: nat, *fichas*: multiconj(ocurrencia), *o*: ocurrencia)

Rep : estr e $\longrightarrow$ bool
Rep(e) ≡ true $\iff$ $e.tipoNotif(IdCliente, Empezar, TurnoDe, Ubicar, Reponer, SumaPuntos, Mal)$
Abs : estr n $\longrightarrow$ notificacion {Rep(n)}
Abs(n) ≡ $(\forall not : notificacion)$
(Abs(n) $=_{\text{obs}}$ notificacion $\Leftrightarrow$ $n =_{\text{obs}} datos(not)$)

Algoritmos

iIdCliente(**in** *cid*: IdCliente)→ *res* : estr
1: **return** (IdCliente, cid, 0, vacio(), vacio())

iEmpezar(**in** *n*: tamanioTablero)→ *res* : estr
1: **return** (Empezar, 0, n, vacio(), vacio())

iTurnoDe(**in** *cid*: tamanioTablero)→ *res* : estr
1: **return** (TurnoDe, cid, 0, vacio(), vacio())

iUbicar(**in** *cid*: `nat, in o: ocurrencia)` → *res* : estr
1: **return** (Ubicar, cid, 0, vacio(), o)

iReponer(**in** *f* : `vector(letra))` → *res* : estr
1: **return** (Reponer, 0, 0, f, vacio())

iSumaPuntos(**in** *cid*: `nat)` → *res* : estr
1: **return** (SumaPuntos, cid, n, vacio(), vacio())

iMal→ *res* : estr
1: **return** (Mal, 0, 0, vacio(), vacio())

2.6. Módulo Servidor

Interfaz

se explica con: SERVIDOR

géneros: `servidor`

Operaciones básicas de servidor

NUEVOSERVIDOR(**in** *cantJug*: `nat`, **in** *v*: `variante`, **in** *r*: `cola(letra)`) → *res* : `servidor`
Pre ≡ {cantJug>0 ∧ $tamanio(r) > 0$}
Post≡ {$res =_{\text{obs}} nuevoServidor(cantJug, v, r)$}
Complejidad: $(tamanio^2 + |\Sigma|K + FK)$
Descripción: crea un servidor donde los clientes esperados es la cantJugadores, la configuración es la tupla de la variate y el repositorio y las notificaciones están vacías.

CONSULTARCOLA(**in/out** *s*: `servidor`, **in** *cid*: `nat`) → *res* : `cola(notificaciones)`
Pre ≡ {empezo(s) ∧$cid < s.clientesConectados \wedge s = S_0$}
Post≡ {s =obs consultarCola(s,cid) ∧res =obs notificaciones(cid)}
Complejidad:O(s.notificacionesIndivicuales[cid)][$Desencolalasnotificacionesindividualesymueveelitreadorenlasge$

empezo(s)= $s.clientesConectados == s.clientesEsperados$

CONECTARCLIENTE(**in** *s*: `servidor`) → *res* : `servidor`
Pre ≡ {s.clientesConectados <s.clientesEsperados}
Post ≡ {$res =_{\text{obs}} conectarCLiente(s)$}
Complejidad: $O(1)$)
Descripción: Si, en clientes conectados.
Aliasing: suma un cliente a los que había y si es igual a los esperados manda una notificacion

ENVIARMENSAJE(**in/out** *s*: `servidor`, **in** *cid*: `nat`, **in** *o*: `ocurrencia`)
Pre ≡ {cid <s.clientesConectados}
Post ≡ {$res =_{\text{obs}} recibirMensaje(s, cid, o)$}
Complejidad: $O(Lmax^2 + long(o))$
Descripción: Mensajes del servidor

Aliasing: Si en servidor

Representación

Representación de servidor

`servidor` **se representa con** `estr`

donde `estr` es `tupla`(*clientesEsperados*: `nat` , *clientesConectados*: `nat` , *j*: `juego` , *configuracion*: `tupla(var: variante, r:cola(letra))` , *notificacionesIndividuales*: `vector(cola(tupla(notificacion, nat))` , *notificacionesGenerales*: `vector(tupla(notificacion, nat))` , *iterador*: `vector(nat)` , *contador*: `nat`)

Rep : estr e $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ e.clientesEsperados>0 $\wedge verVariante(e.j) =_{\text{obs}} get < 0 > (e.configuracion) \wedge$
$verRepositorio(e.j) =_{\text{obs}} get < 1 > (e.configuracion) \wedge$
$(\forall cid : nat)(cid \leq e.clientesConectados) \longrightarrow_L$
$((\forall i : nat)(0 \leq i < |consultarCola(cid)| - 1 \longrightarrow_L \pi_2(consultarCola(cid)[i]) < \pi_2(consultarCola(cid)[i+1])) \wedge e.contador > \pi_2(consultarCola(cid)[i])))$
contadoresCola(e.notificacionesIndividuales[cid]) $\wedge$
$contadoresCola(aCola(e.notificacionesGenerales))$

aCola: $vector(tupla(notificacion, contador)) \rightarrow cola(tupla(notificacion, contador))$

aCola(v) $\equiv if$ vacio?(v) $then$ vacio

encolar(prim(v),aCola(fin(v)))

contadoresCola: cola(tupla(notificacion,contador)) $\rightarrow$ *bool*
$contadoresCola(c) \equiv if cantidad(c) = 1 then$

true

else

$\pi_1(proximo(c)) < \pi_1(proximo(desencolar(c)) \wedge contadoresCola(desencolar(c))$

Abs : estr e $\longrightarrow$ tablero {Rep(e)}

Abs(e) $\equiv$ $(\forall ser : servidor)$(Abs(e) $=_{\text{obs}}$ ser $\iff$ $s.clientesEsperados =_{\text{obs}} esperados(ser) \wedge s.clientesConectados(s) =_{\text{obs}} conectados(ser) \wedge s.configuracion =_{\text{obs}} configuracion(ser) \wedge s.juego =_{\text{obs}} juego(ser) \wedge (\forall i : nat)(i < conectados(ser) \Rightarrow_{\text{L}} ordenarSegunSegundaCoordenada(merge(subseq(s.notificacionesGenerales, iterador[i], long(s.notificacionesGenerales)), s.notificacionesIndividuales[i])) = obs\ consultarCola(ser, i)$

colaASecuencia(c)= if vacio?(c) then <>else proximo(c)· colaASecuencia(desencolar(c))

Algoritmos

---

iNuevoServidor(**in**/**out** *cantJugadores*: `nat`, **in** v: `variante`, **in** r: `cola(letra)`)

1: $res.clientesEsperados = cantJugadores$
2: $res.configuracion = (v, r)$
3: $res.clientesConectados = 0$
4: $res.j = nuevoJuego(s.clientesEsperados, get < 0 > (s.configuracion), get < 1 > (s.configuracion))$
5: $res.notificacionesGenerales = vacio$
6: $res.notificacionesIndividuales = vacio$
7: $res.iterador = vacio$
8: $res.contador = 0$ =0

---

iConsultarCola(**in** s: `servidor`, **in** cid: `nat`)→ res : cola(notificaciones)

```
1: n = (s.notificacionesIndividuales[cid].size() +
2:    subseq(s.notificacionesGenerales,s.iterador[cid],s.notificacionesGenerales.size())).size()
3: res = vacio()
4: for i = 0 to n
5:    if get<1>(proximo(s.notificacionesIndividuales[cid])) < get<1>s.notificacionesGenerales[s.iterador[cid]]
6:        res.encolar(get < 0 > (proximo((s.notificacionesIndividuales[cid]))
7:        desencolar(s.notificacionesIndividuales[cid])
8:    else
9:       res.encolar(get < 0 > (s.notificacionesGenerales[s.iterador[cid]])
10:      s.iterador[cid] = s.iterador[cid] + 1
11: return res
```

[Comentario: En la estructura, el vector de iteradores nos devuelve en O(1) la ultima posicion consultada por ese jugador en el vector de notificaciones generales].

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

```
1: s.clientesConectados = s.clientesConectados + 1
2: s.notificacionesIndividuales.pushback(vacia())
3: notificacionesIndividuales[res.clientesConectados − 1].pushback(IdCliente(res.clientesConectados −
   1), s.contador)
4: iterador.pushback(0)
5: if (s.clientesConectados =obs s.clientesEsperados) :
6:    notificacionesGenerales.pushback(empezar(tamanio(s.j)), s.contador)
7: s.contador = s.contador + 1
```

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

```
1: if (jugadaValida(o, s.j) and turno(s.j) == cid and s.clientesConectados == s.clientesEsperados) :
2:    notificacionesIndividuales[cid].pushback(reponer(proximosN(get<1>s.configuracion, o.size)),s.contador)
3:    s.contador = s.contador + 1
4:    ubicarFichas(s.j, o)
5:    notificacionesGenerales.pushback(turnoDe(Turno(s.j)), s.contador)
6:    s.contador = s.contador + 1
7:    notificacionesGenerales.pushback(Ubicar(cid, ocurrencia), s.contador)
8:    s.contador = s.contador + 1
9:    notificacionesGenerales.pushback(SumaPuntos(cid, calcularPuntaje(s.j, o)), s.contador)
10:   s.contador = s.contador + 1
11: else
12:   notificacionesIndivuales[cid].pushback(Mal(), s.contador)
13:   s.contador = s.contador + 1
```