Algoritmos y Estructuras de Datos II

Trabajo Práctico 1: Diseño:

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

Juego de Palabras

Integrante	LU	Correo electrónico
Higa, Leandro	1444/21	leanhiga12@gmail.com
Iturraspe, Santiago	130/17	sj_iturraspe@hotmail.com
Romero, Tomás Augusto	564/21	tomas.a.romero0711@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Diseño del Juego

Módulo Juego

Interfaz

parámetros formales
géneros α
función COPIAR(**in** a: α) $\rightarrow$ *res*: α
Pre $\equiv$ {true}
Post $\equiv$ {$res = a$}
Complejidad: $\Theta(copy(a))$
Descripción: función copia de α's

se explica con: JUEGO

géneros: `juego(`α`)`

operaciones básicas:

NUEVOJUEGO(**in** *cantJug*: `nat`, **in** v: `Variante` , **in** r: `cola(Letra)`) $\rightarrow$ *res*: `juego`
Pre $\equiv$ {longitud(r) $\geq$ (tamañoTablero(v)^2 + cantJug * #fichas(v)) }
Post $\equiv$ {$r\hat{e}s =_{obs}$ nuevoJuego(*cantJug, v, r*)}
Complejidad: $\mathcal{O}(N^2 + |\Sigma|K + FK)$
Descripción: Genera un nuevo juego
Aliasing: Res se devuelve por referencia.

UBICAR(**in/out** j: `juego`, **in** o: `ocurrencia`)
Pre $\equiv$ {jugadaValida?(j,o)}
Post $\equiv$ {ubicar(j,o)}
Complejidad: $\mathcal{O}(m)$, donde m es la cantidad de fichas que se ubican
Descripción: Genera instancia de juego ubicando las Letras en las ubicaciones que eligió un jugador en un turno determinado.
Aliasing: El juego entra y sale por referencia.

VARIANTE(**in** j: `juego`) $\rightarrow$ *res*: `variante`
Pre $\equiv$ {True}
Post $\equiv$ {$r\hat{e}s =_{obs}$ variante(j)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la variante del juego.
Aliasing: la variante se devuelve por referencia.

JUGADAVALIDA?(**in** j: `juego`) $\rightarrow$ *res*: `bool`
Pre $\equiv$ {True}
Post $\equiv$ {$r\hat{e}s =_{obs}$ jugadaValida?(j)}
Complejidad: $\mathcal{O}(L^2_{Max})$
Descripción: res es True si y sólo si la jugada es válida.
Aliasing: No presenta aspectos de aliasing.

TURNO(**in** j: `juego`) $\rightarrow$ *res*: `nat`
Pre $\equiv$ {True}
Post $\equiv$ {$r\hat{e}s =_{obs}$ turno(j)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve a quien le toca en el turno actual, devolverá el correspondiente al jugador que le corresponda jugar.

Aliasing: No presenta aspectos de aliasing.

PUNTAJE(**in** j: `juego`, **in** jug: `nat`) → res: `nat`
Pre ≡ {$jug \leq$ #jugadores(j)}
Post ≡ {$r\hat{e}s =_{obs}$ puntaje(j, jug)}
Complejidad: $\mathcal{O}(1 + m * L_{Max})$
Descripción: Devuelve el puntaje de un jugador válido en un juego.
Aliasing: No presenta aspectos de aliasing.

#JUGADORES(**in** j: `juego`) → res: `nat`
Pre ≡ {True}
Post ≡ {$r\hat{e}s =_{obs}$ #jugadores(j)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de jugadores del juego
Aliasing: No presenta aspectos de aliasing.

#FICHAS(**in** j: `juego`, **in** jug: `nat` , **in** l: `letra`) → res: `nat`
Pre ≡ {$jug \leq$ #jugadores(j)}
Post ≡ {$r\hat{e}s =_{obs}$ #fichas(j, jug, l)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve la cantidad de fichas de una letra que tiene un jugador en un juego.
Aliasing: No presenta aspectos de aliasing.

TABLERO(**in** j: `juego`) → res: `tablero`
Pre ≡ {True}
Post ≡ {$r\hat{e}s =_{obs}$ tablero(j)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el tablero del juego.
Aliasing: Devuelve el tablero por referencia.

FICHAS_JUGADORES(**in** j: `juego`) → res: `vector(vector(nat))`
Pre ≡ {True}
Post ≡ {(tam(res) = #Jugadores(j)) $\wedge_L$ ($\forall$ i : nat)(0 $\leq i <$ $tam(res) \rightarrow_L$ $r\hat{e}s[i] =_{obs}$ fichas(j,i))}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve un vector de vectores que indican la cantidad de fichas de cada letra que tiene cada jugador.
Aliasing: Devuelve res por referencia.

REPOSITORIO(**in** j: `juego`) → res: `cola(letra)`
Pre ≡ {True}
Post ≡ {$r\hat{e}s =_{obs}$ repositorio(j)}
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el repositorio del juego.
Aliasing: Devuelve el repositorio por referencia.

Implementación

Representación

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

donde `estr` **es** `tupla`(

tablero : `tablero`,
fichas_jugadores: `vector( vector(nat) )`,

puntaje_jugadores: `vector(nat)`,
turno: `nat`,
repositorio: `cola(letra)`,
variante: `variante`,
ocurrencias: `vector( cola(ocurrencia) )`

Rep : estr $\longrightarrow$ bool

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

1. universo de jugadores es entre 0 y longitud(e.fichas_Jugadores)
2. longitud(e.fichas_jugadores) = longitud(e.puntaje_jugadores) = longitud(e.ocurrencias)
3. universo de fichas es entre 0 y longitud de puntajeLetras(e.variante). Cada posición refiere a una letra.
4. ($\forall$ n: nat)(0 $\leq$ n <longitud(fichas_jugadores) $\rightarrow_L$ (($\forall$ i: nat)(0 $\leq$ i $<$ longitud(fichas_jugadores[n] $\rightarrow_L$ i $<$ longitud(puntajeLetras(e.variante)))))) (todas las fichas de cada jugador deben pertenecer al abecedario/palabras_Legitimas de variante)
5. la suma de la cantidad de fichas de los e.fichas_Jugadores y e.repositorio debe ser mayor o igual a la cantidad de coordenadas de un tablero más la cantidad de jugadores por la cantidad de fichas de variante (N^2 + K$*$F) //interfaz variante
6. todas las palabras que pueden formarse de manera vertical y horizontal en el tablero, a partir de las fichas ubicadas, deben ser palabrasLegitimas.
7. todas las ocurrencias de todos los jugadores deben ser jugadas válidas. (en e.ocurrencias[i], donde i puede ser cualquiera de los jugadores, y cada jugador tiene una cola de ocurrencias asociadas, a estas ocurrencias se hacen referencia)
8. tamaño(e.tablero) = tamañoTablero(e.variante)
9. ($\forall$ n: nat)(0 $\leq$ n $<$ longitud(fichas_jugadores) $\rightarrow_L$ longitud(fichas_jugadores[n] = #fichas(e.variante))) (la cantidad de fichas que posee cada jugador debe ser la misma que #fichas(e.variante))
10. cada ocurrencia de ocurrencias debe aparecer en el tablero ya que son jugadas validas realizadas por los jugadores en sus turnos.(en e.ocurrencias[i], donde i puede ser cualquiera de los jugadores, y cada jugador tiene una cola de ocurrencias asociadas, a estas ocurrencias se hacen referencia)
11. cada letra del e.repositorio debe estar entre 0 y longitud(puntajeLetras(e.variante)), ya que este sería el universo de letras/abecedario.
12. turno $\leq$ longitud(puntaje_jugadores)

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

Abs(e) $\equiv$ j : *juego* | variante(j) = e.variante $\wedge$
#jugadores(j) = longitud(e.fichas_Jugadores) $\wedge$
repositorio(j) = e.repositorio $\wedge$
tablero(j) = e.tablero $\wedge$
turno(j) = e.turno $\wedge$
($\forall$ jug: nat)(0 $\leq$ jug $\leq$ #jugadores(j) $\rightarrow_L$ puntaje(j, jug) = e.puntaje_Jugadores[jug] $\wedge$ fichas(j, jug) = e.fichas_Jugadores[jug])

Algoritmos

iNUEVOJUEGO(**in** *cantJug* : `nat`, **in** v : `variante`, **in** r : `cola(letra)`) $\longrightarrow$ *res* : `estr`

```
1:  variante ← v
2:  tablero ← nuevoTablero(tamañoTablero(v))                          ▷ O(n^2)
3:  turno ← 0
4:  ocurrencias ← vacio()
5:  fichas_Jugadores ← vacio()
6:  puntaje_Jugadores ← vacio()
7:  mientras longitud(fichas_Jugadores) <cantJug hacer                ▷ O(K)
8:      AgregarAtras( fichas_Jugadores, vacio():: vector())
9:      AgregarAtras( puntaje_Jugadores, 0 )
10:     AgregarAtras( ocurrencias, vacio():: cola() )
11: para (i ← 0, i <cantJug, i++) hacer                               ▷ O(|∑|K)
12:     para (j ← 0, j <longitud(puntajeLetras(v)), j++) hacer
13:         AgregarAtras( fichas_Jugadores[i], 0 )
14: para (i ← 0, i <cantJug, i++ hacer                                ▷ O(FK)
15:     para (j ← 0, j <#fichas(v), j++ hacer
16:         fichas_Jugadores[i][proximo(r)] ← fichas_Jugadores[i][proximo(r)] + 1
17:         desencolar(r)
18: repositorio ← r
19: res ← ⟨tablero, fichas_Jugadores, puntaje_Jugadores, turno, repositorio, ocurrencias⟩
    Complejidad: O(N^2 + |∑|K + FK)
```

iJUGADAVÁLIDA(**in** j_0 : `estr`, **in** o: `ocurrencia`) $\longrightarrow$ res : `bool`

1: $j \leftarrow ubicar(j_0, o)$ ▷ $\mathcal{O}(m)$, m siendo la cantidad de fichas que se ubican, m es menor o igual que L_Max, porque todas las letras de la jugada deben formar una palabra legítima. Paso por referencia j.
2: $palabraCoreIncluyeTodasLasFichas \leftarrow true$
3: **si** vacio?(o) **entonces**
4: $res \leftarrow true$
5: **else**
6: **si** cardinal(o) == 1 **entonces**
7: $res \leftarrow\neg$ hayLetra(j.tablero, siguiente(it)$_1$, $siguiente(it)_2$)&&$todasLegitimas?(j, palabrasUbicadas(\text{j}, o))$
8: **else**
9: $it \leftarrow crearIt(o)$
10: **si** siguiente(it)$_2$=siguiente(avanzar(it))$_2$ **entonces**
11: $it \leftarrow crearIt(o)$
12: $oHorizontal \leftarrow vacio() :: ocurrencia$
13: $prim \leftarrow siguiente(it)_1$
14: **para** ($i \leftarrow siguiente(it)_1$, $enTablero?(j.tablero, i, siguiente(it)_2)$ && $hayLetra?(j.tablero,\ i,\ siguiente(it)_2)$, $i--$ **hacer** ▷ $\mathcal{O}(L\_Max)$
15: $prim \leftarrow i$
16: **para** ($i \leftarrow prim$, $enTablero?(j.tablero,\ i,\ siguiente(it)_2)$ && $hayLetra?(j.tablero,\ i,\ siguiente(it)_2)$, $i++$ **hacer** ▷ $\mathcal{O}(L\_Max)$
17: $AgregarRapido(oHorizontal, <i, siguiente(it)_2, letra(j.tablero,\ i,\ siguiente(it)_2) >)$
18: $itO \leftarrow crearIt(o)$
19: **mientras** haySiguiente?(itO) **hacer** ▷ $\mathcal{O}(m * L\_Max)$
20: **si** $\neg(pertenece?(oHorizontal, siguiente(itO)))$ ***entonces***
21: $palabraCoreInlcuyeTodasLasFichas \leftarrow false$
22: $palabras \leftarrow palabrasUbicadas(j, oHorizontal)$ ▷ $\mathcal{O}(m * L\_Max)$, siendo m esta vez L_Max
23: **else**
24: $it \leftarrow crearIt(o)$
25: $oVertical \leftarrow vacio() :: conjunto_Lineal()$
26: $prim \leftarrow siguiente(it)_2$
27: **para** ($i \leftarrow siguiente(it)_2$, $enTablero?(j.tablero, siguiente(it)_1,\ i)$ && $hayLetra?(j.tablero,\ siguiente(it)_1, i)$, $i--$ **hacer** ▷ $\mathcal{O}(L\_Max)$
28: $prim \leftarrow i$
29: **para** ($i \leftarrow prim$, $enTablero?(j.tablero,\ siguiente(it)_1,\ i)$ && $hayLetra?(j.tablero,\ siguiente(it)_1, i)$, $i++$ **hacer** ▷ $\mathcal{O}(L\_Max)$
30: AgregarRapido(oVertical, <siguiente(it), i, letra(j.tablero, siguiente(it), i) >)
31: $itO \leftarrow crearIt(o)$
32: **mientras** haySiguiente?(itO) **hacer** ▷ $\mathcal{O}(m * L\_Max)$
33: **si** ¬pertenece?(oVertical, siguiente(itO)) **entonces**
34: $palabraCoreInlcuyeTodasLasFichas \leftarrow false$
35: $palabras \leftarrow palabrasUbicadas(j, oVerical)$ ▷ $\mathcal{O}(m * L_{Max})$, siendo m esta vez L_Max
36: $res \leftarrow (celdasLibres?(j.tablero, o)$&&$(palabraCoreInlcuyeTodasLasFichas$&&$todasLegitimas?(j, palabras)))$

Complejidad: $\mathcal{O}(L^2_{Max})$
Justificación: -armar oHorizontal o oVerical es $\mathcal{O}(L_Max)$, saber palabraCoreInlcuyeTodasLasFichas es $\mathcal{O}(m * L_Max)$, m siendo la longitud de o,
- conseguir las palabras ubicadas es $\mathcal{O}(L^2_{Max})$,
- verificar que todas sean legitimas es $\mathcal{O}(L^2_{Max})$ porque todasLegitimas?
es$\mathcal{O}(|os| * L_Max)$ y |os| en esta caso es a lo sumo L_{Max}, porque la palabra que incluye a todas las fichas, a lo sumo genera una L_{Max} más.
- celdasLibres? es $\mathcal{O}(L_{Max})$ por ser lineal con respecto a la longitud de o, que esta no puede ser mayor a L_{Max} ya que la jugada o todas las letras deben formar una palabra legítima en común.

iUBICAR(**in/out** j : `estr`, **in** o: `ocurrencia`)

1: encolar(j.ocurrencias[j.turno], o)
2: $aQuienLeToca \leftarrow (j.turno\ )$
3: $itO \leftarrow crearIt(o)$
4: **mientras** (haySiguiente(itO)) **hacer**
5: ponerLetra(j.tablero, siguiente(itO)$_1$, siguiente(itO)$_2$, siguiente(itO)$_3$)
6: $cantAnterior \leftarrow j.fichas\_Jugadores[aQuienLeToca][siguiente(itO)\ _3]$
7: $j.fichas\_Jugadores[aQuienLeToca][siguiente(itO)\ _3] \leftarrow cantAnterior - 1$
8: avanzar(itO)
9: $itO \leftarrow crearIt(o)$
10: **mientras** haySiguiente(itO) **hacer**
11: $cantAnterior \leftarrow j.fichas\_Jugadores[aQuienLeToca][proximo(j.repositorio)]$
12: $j.fichas\_Jugadores[aQuienLeToca][proximo(j.repositorio)] \leftarrow cantAnterior + 1$
13: desencolar(j.repositorio)
14: avanzar(itO)
15: $j.turno \leftarrow (j.turno\ +\ 1) mod\ j.\#jugadores$
=0

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

---

iVARIANTE(**in** j : `estr`) $\longrightarrow$ res : `variante`

1: $res \leftarrow$ j.variante
2: Complejidad: $\mathcal{O}(1)$
3: Justificación: res se devuelve por referencia. =0

---

iTURNO(**in** j : `estr`) $\longrightarrow$ res : `nat`

1: $res \leftarrow j.turno$
Complejidad: $\mathcal{O}(1)$

---

iPUNTAJE(**in** j : `estr`,**in** jug : `nat`) $\longrightarrow$ res : `nat`

1: $punt \leftarrow j.puntaje\_Jugadores[jug]$
2: **mientras** not(esVacia?(j.ocurrencias[jug])) **hacer**
3: $punt \leftarrow punt + puntajePalabras(j,\ palabrasUbicadas(j,\ proximo(j.ocurrencias[jug])))$ ▷ $\mathcal{O}(m * L\_Max)$, siendo m la cantidad de fichas ubicadas en una jugada determinada
4: desencolar(j.ocurrencias[jug])
5: $res \leftarrow punt$
Complejidad: $\mathcal{O}(1 + M * L\_Max)$, siendo M la cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación.
Justificación: va a operar puntajeObtendo tanto como fichas ubicadas desde la última vez que se invocó esta operación.

---

i#JUGADORES(**in** j : `estr`) $\longrightarrow$ res : `nat`

1: $res \leftarrow longitud(j.fichas\_Jugadores)$
Complejidad: $\mathcal{O}(1)$

---

i#FICHAS(**in** j : `estr`, **in** jug : `nat`, **in** l : `letra`) $\longrightarrow$ res : `nat`

1: $res \leftarrow j.fichas\_Jugadores[jug][l]$
Complejidad: $\mathcal{O}(1)$

---

iTODASLEGÍTIMAS?(**in** j : `estr`, **in** os : `conj(ocurrencia)`) $\longrightarrow$ res : `bool`

```
1:  res ← true
2:  itOs ← crearIt(os)
3:  mientras haySiguiente(itOs) hacer                                          ▷ O(|os| * L_Max)
4:      itO ← crearIt(siguiente(itOs))
5:      colaLetras ← vacio()
6:      si siguiente(itO)_2==siguiente(avanzar(itO))_2 entonces
7:          itO ← crearIt(siguiente(itOs))
8:          prim ← siguiente(itO)_1
9:          para (i ← siguiente(it)_1,
    enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i-- hacer   ▷ O(L_Max)
10:             prim ← i
11:         para (i ← siguiente(it)_1,
    enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i++ hacer   ▷ O(L_Max)
12:             encolar( colaLetras, letra(j.tablero, i ,siguiente(it)_2 ) )
13:     else
14:         itO ← crearIt(siguiente(itOs))
15:         prim ← siguiente(itO)_2
16:         para (i ← siguiente(itO)_2,
    enTablero?(j.tablero, siguiente(itO)_1,i) && hayLetra?(j.tablero, siguiente(itO)_1,i), i-- hacer   ▷ O(L_Max)
17:             prim ← i
18:         para (i ← prim,
    enTablero?(j.tablero, siguiente(itO)_1,i) && hayLetra?(j.tablero, siguiente(itO)_1,i), i++ hacer   ▷ O(L_Max)
19:             encolar( colaLetras, letra(j.tablero, siguiente(itO)_1 ,i ) )
20:     si ¬palabraLegitima?( j.variante, colaLetras ) entonces                ▷ O(L_Max)
21:         res ← false
22:     avanzar(itOs)
    Complejidad: O(|os| * L_Max)
```

iPALABRASUBICADAS(**in** j : `estr`, **in** o : `ocurrencia`) $\longrightarrow$ res : `conj(ocurrencia)`

```
1: si vacio?(o) entonces
2:     res ← vacio()
3: else
4:     si cardinal(o) = 1 entonces
5:         it ← crearIt(o)
6:         oHorizontal ← vacio() :: ocurrencia
7:         prim ← siguiente(it)_1   ▷ donde va a estar la primera tupla de la palabra para luego agregar las tuplas en orden respecto a la palabra.
8:         para (i ← siguiente(it)_1, enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i−−) hacer
9:             prim ← i
10:        para (i ← prim, enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i++) hacer
11:            AgregarRapido(oHorizontal, ⟨ i, siguiente(it)_2, letra(j.tablero, i, siguiente(it)_2) ⟩ )
12:        AgregarRapido(res, oHorizontal)
13:        oVertical ← vacio() :: ocurrencia
14:        prim ← siguiente(it)_2                          ▷ donde va a estar la segunda tupla de la palabra.
15:        para (i ← prim, enTablero?(j.tablero, i, siguiente(it)_1) && hayLetra?(j.tablero, i, siguiente(it)_1), i−−) hacer
16:            prim ← i
17:        para (i ← prim, enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i++) hacer
18:            AgregarRapido(oVertical, ⟨ i, siguiente(it)_1, letra(j.tablero, i, siguiente(it)_1) ⟩ )
19:        AgregarRapido(res, oVertical)                   ▷ hasta aca es O( L_max )
20:    else                                                ▷ Más de una letra ubicada
21:        it ← crearIt(o)
22:        si siguiente(it)_2 = siguiente(avanzar(it))_2 entonces   ▷ si la o es horizontal
23:            iPalabrasUbicadasOHorizontal(j, o, res)     ▷ O(m * L_Max)
24:        else                                            ▷ Si la o es vertical
25:            iPalabrasUbicadasOVertical(j, o, res)       ▷ O(m * L_Max)
26: devolver res
```

iPALABRASUBICADASOHORIZONTAL(**in** j : `estr`, **in** o : `ocurrencia`, **in/out** res : `conj(ocurrencia)`)

```
1: it ← crearIt(o)
2: oHorizontal ← vacio() :: ocurrencia
3: prim ← siguiente(it)_1   ▷ donde va a estar la primera tupla de la palabra para luego agregar las tuplas en orden respecto a la palabra.
4: para (i ← prim, enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i−−) hacer
5:     prim ← i
6: para (i ← prim, enTablero?(j.tablero, i, siguiente(it)_2) && hayLetra?(j.tablero, i, siguiente(it)_2), i++) hacer
7:     AgregarRapido(oHorizontal, ⟨ i, siguiente(it)_2, letra(j.tablero, i, siguiente(it)_2) ⟩ )
8: AgregarRapido(res, oHorizontal)
9: mientras haySiguiente?(it) hacer                         ▷ O( m ) <= O( L_max ), |o|= m
10:    oVertical ← vacio() :: ocurrencia                    ▷ armo oVerical, que se formó con siguiente(it)
11:    prim ← siguiente(it_2)
12:    para ( i ← siguiente(it)_2, enTablero?(j.tablero, siguiente(it)_1, i) && hayLetra?(j.tablero, siguiente(it)_1, i), i−−) hacer
13:        prim ← i
14:    para ( i ← prim, enTablero?(j.tablero, siguiente(it)_1, i) && hayLetra?(j.tablero, siguiente(it)_1, i), i++) hacer
15:        AgregarRapido(oVertical, ⟨ siguiente(it)_1, i, letra(j.tablero, siguiente(it)_1, i) ⟩ )   ▷ O(copy(tupla))
16:    AgregarRapido(res, oVertical)
17:    avanzar(it)                                          ▷ O(m * L_Max)
```

iPALABRASUBICADASOVERTICAL(**in** j : `estr`, **in** o : `ocurrencia`, **in/out** res : `conj(ocurrencia)`)

1: $it \leftarrow$ crearIt(o)
2: $oVertical \leftarrow$ vacio() :: ocurrencia
3: $prim \leftarrow$ siguiente$(it)_2$ ▷ donde va a estar la segunda tupla de la palabra para ir agregando.
4: **para** (i $\leftarrow$ siguiente(it)$_2$, enTablero?(j.tablero, i, siguiente(it)$_1$) && hayLetra?(j.tablero, i, siguiente(it)$_1$), i−−) **hacer**
5: $prim \leftarrow$ i
6: **para** (i $\leftarrow$ prim, enTablero?(j.tablero, i, siguiente(it)$_1$) && hayLetra?(j.tablero, i, siguiente(it)$_1$), i++) **hacer**
7: AgregarRapido(oVertical, ⟨ i, siguiente(it)$_1$, letra(j.tablero, i, siguiente(it)$_1$) ⟩)
8: AgregarRapido(res, oVertical) ▷ hasta aca $\mathcal{O}(L_{max})$
9: **mientras** haySiguiente?(it) **hacer**
10: $oHorizontal \leftarrow$ vacio() :: ocurrencia
11: $prim \leftarrow$ siguiente$(it)_1$
12: **para** (i $\leftarrow$ siguiente(it)$_1$, enTablero?(j.tablero, siguiente(it)$_2$, i) && hayLetra?(j.tablero, siguiente(it)$_2$, i), i−−) **hacer**
13: $prim \leftarrow$ i
14: **para** (i $\leftarrow$ prim, enTablero?(j.tablero, siguiente(it)$_2$, i) && hayLetra?(j.tablero, siguiente(it)$_2$, i), i++) **hacer**
15: AgregarRapido(oHorizontal, ⟨ siguiente(it)$_2$, i, letra(j.tablero, siguiente(it)$_2$, i) ⟩) ▷ $\mathcal{O}$(copy(tupla))
16: AgregarRapido(res, oHorizontal)
17: avanzar(it)

iPUNTAJEPALABRAS(**in** j : `estr`, **in** os : `conj(ocurrencia)`) $\longrightarrow$ res : `nat`

1: **si** vacio?(os) **entonces**
2: $res \leftarrow 0$
3: **else**
4: $it \leftarrow$ crearIt(os)
5: **mientras** (haySiguiente(it)) **hacer** ▷ $\mathcal{O}(|\text{os}|)$ cantidad de palabras ubicadas
6: $res \leftarrow res +$ puntajePalabra(j, j.tablero, siguiente(it)) ▷ $\mathcal{O}(L_{Max})$
7: Avanzar(it)
8: **devolver** res

iPUNTAJEPALABRA(**in** j : `estr`, **in** o : `ocurrencia`) $\longrightarrow$ res : `nat`

1: $res \leftarrow 0$
2: $it \leftarrow$ crearIt(o)
3: **mientras** (haySiguiente(it)) **hacer** ▷ $\mathcal{O}(L_{Max})$
4: $res \leftarrow res +$ puntajeLetra(j.variante, siguiente(it)$_3$) ▷ $\mathcal{O}(\ |\text{letra}|\ ) =>\mathcal{O}(1)$, busca en un vector, el puntaje de la letra dado por la posición.
5: Avanzar(it)
6: **devolver** res

iTABLERO(**in** j : `estr`) $\longrightarrow$ res : `tablero`

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

iFICHAS_JUGADORES(**in** j : `estr`) $\longrightarrow$ res : `vector(vector(nat))`

1: $res \leftarrow j.fichas\_jugadores$
Complejidad: $\mathcal{O}(1)$

iREPOSITORIO(**in** j : `estr`) $\longrightarrow$ res : `cola(letra)`

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

Servicios usados

TABLERO, VARIANTE, VECTOR, COLA, CONJUNTO LINEAL

Módulos auxiliares

Módulo Tablero

Interfaz

se explica con: TABLERO

géneros: `tablero`

Operaciones básicas de tablero

NUEVOTABLERO(**in** n : `nat`) $\rightarrow$ res : `tablero`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ nuevoTablero(n)}
Complejidad: $\mathcal{O}(n^2)$
Descripción: genera un nuevo tablero de tamaño n
Aliasing: no presenta aspectos de aliasing

PONERLETRA(**in/out** t : `tablero`, **in** i : `nat`, **in** j : `nat`, **in** l : `letra`)
Pre $\equiv$ {$t_0 = t \wedge$ enTablero?$(t, i, j) \wedge_L \neg$hayLetra?(t, i, j)}
Post $\equiv$ {$t =_{\text{obs}}$ ponerLetra(t_0, i, j, l)}
Complejidad: $\mathcal{O}(1)$
Descripción: ubica una letra en la posición (i, j)
Aliasing: l se ubica por copia. Tablero entra y sale por referencia.

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 del tablero
Aliasing: no presenta aspectos de aliasing

HAYLETRA?(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `bool`
Pre $\equiv$ {enTablero?(t, i, j)}
Post $\equiv$ {$res =_{\text{obs}}$ hayLetra?(t, i, j)}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve true si y sólo si hay una letra en la posición (i, j)
Aliasing: no presenta aspectos de aliasing

LETRA(**in** t : `tablero`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `letra`
Pre $\equiv$ {enTablero?$(t, i, j) \wedge_L$ hayLetra?(t, i, j)}
Post $\equiv$ {$res =_{\text{obs}}$ letra(t, i, j)}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la letra en la posición (i, j)
Aliasing: no presenta aspectos de aliasing

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: devuelve true si y sólo si la posición (i, j) es válida
Aliasing: no presenta aspectos de aliasing

Implementación

Representación

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

donde `estr` es `tupla`(*tamaño*: `nat` , *tablero*: `vector(vector(puntero de letra))`)

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ longitud(e.tablero) = e.tamaño $\wedge$
($\forall$ i: nat)(0$\leq$ i <longitud(e.tablero) $\rightarrow_L$ longitud(e.tablero[i]) = e.tamaño)

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

Abs(e) $\equiv$ t : *tablero* | tamaño (t) = e.tamaño $\wedge$
($\forall i, j$: *nat*)(enTablero(t,i,j) $\Rightarrow_L$ hayLetra?(t, i, j) = $\neg$((e.*tablero*[i][j])= NULL) $\wedge$
letra(t, i, j) = *(e.tablero[i][j]))

Algoritmos

---

inuevoTablero(**in** n : `nat`) $\rightarrow$ res : estr
  res.tamaño $\leftarrow$ n
  tablero $\leftarrow$ vector::Vacía()
  $i \leftarrow 0$
  mientras i <n **hacer** ▷ $\mathcal{O}(n)$
    $j \leftarrow 0$
    $fila \leftarrow$ vector::Vacía()
    mientras j <n **hacer** ▷ $\mathcal{O}(n)$
      $agregarAtras(fila, NULL)$
      $j++$
    $agregarAtras(tablero, fila)$
    $i++$
  $res.tablero \leftarrow tablero$
  Complejidad: $\mathcal{O}(n^2)$

---

iponerLetra(**in/out** t : `estr`, **in** i : `nat`, **in** j : `nat`, **in** l : `nat`)
  $t.tablero[i][j] \leftarrow \&l$
  Complejidad: $\mathcal{O}(1)$

---

itamaño(**in** t : `estr`) $\rightarrow$ res : nat
  $res \leftarrow$ t.tamaño
  Complejidad: $\mathcal{O}(1)$

---

ihayLetra?(**in** t : `estr`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : bool
$res \leftarrow t.tablero[i][j]! = NULL$
Complejidad: $\mathcal{O}(1)$

---

iletra(**in** t : `estr`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : letra
$res \leftarrow *(t.tablero[i][j])$
Complejidad: $\mathcal{O}(1)$

---

ienTablero?(**in** t : `estr`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : bool
$res \leftarrow i <$ t.tamaño && $j <$t.tamaño
Complejidad: $\mathcal{O}(1)$

Servicios usados

VECTOR

Módulo Variante

Interfaz

se explica con: VARIANTE

géneros: `variante`

Operaciones básicas de variante

NUEVAVARIANTE(**in** n: `nat`, **in** f: `nat`, **in** $puntajeLetras$: `vector(nat)`, **in** $palabraLegitima?$: `conj(secu(letra))`) $\rightarrow$ res : `variante`
Pre $\equiv$ $\{n > 0 \land f > 0\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{nuevaVariante}(n, f, puntajeLetras, palabraLegitima?)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: genera una nueva variante
Aliasing: Res se devuelve por referencia.

TAMAÑOTABLERO(**in** v : `variante`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{tamañoTablero}(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve el tamaño del tablero en la variante
Aliasing: no presenta aspectos de aliasing

#FICHAS(**in** v : `variante`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \#\text{fichas}(v)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de fichas que tienen que tener los jugadores en todo momento en esta variante
Aliasing: no presenta aspectos de aliasing

PUNTAJELETRA(**in** v : `variante`, **in** l : `letra`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{puntajeLetra}(v, l)\}$

Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de puntos que vale la letra dentro de la variante
Aliasing: no presenta aspectos de aliasing

PUNTAJELETRAS(**in** v : `variante`) $\rightarrow$ res : `vector(nat)`
Pre $\equiv$ {true}
Post $\equiv$ {$(\forall$ l : nat$)(0 \leq l < tam(v.puntajeLetras) \rightarrow_L \hat{res}[l] =_{obs}$ puntajeLetra(v,l))}
Complejidad: $\mathcal{O}(1)$
Descripción: devuelve la cantidad de puntos que vale cada letra dentro de la variante
Aliasing: no presenta aspectos de aliasing

PALABRALEGITIMA?(**in** v : `variante`, **in** ls : `cola(letra)`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ palabraLegitima?(v, ls)}
Complejidad: $\mathcal{O}(L_{max})$
Descripción: devuelve true si y sólo si la palabra esta definida en la variante
Aliasing: no presenta aspectos de aliasing

Implementación

Representación

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

donde `estr` es `tupla`(*tamañoTablero*: `nat` , *#fichas*: `nat` , *puntajeLetras*: `vector(nat)` , *palabraLegitima?*: `diccDigital(cola(letra), bool)`)

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ $(\forall cl : cola(letra))(cl \in claves(e.palabraLegitima?) \rightarrow_L elementosEnRango(cl, longitud(e.puntajeLetras)))$

elementosEnRango : cola(letra) $\times$ nat $\longrightarrow$ bool

elementosEnRango(cl, n) $\equiv$ **if** vacia?(cl) **then** true **else** $0 <$ proximo(cl) $< n \wedge$ elementosEnRango(desencolar(cl), n) **fi**

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

Abs(e) $\equiv$ $v : variante|$ tamañoTablero(v) = e.tamañoTablero $\wedge$
fichas(v) = e.fichas $\wedge$
$(\forall l : letra)(0 \leq l <)$ longitud(e.puntajeLetras) $\rightarrow_L$ puntajeLetra(v, l) = e.puntajeLetras[l] $\wedge$
$(\forall ls : secu(letra))(definido?(e.palabraLegitima?, ls) \rightarrow_L$ palabraLegitima?(v,ls) = significado(e.palabraLegitima?,ls))

Algoritmos

---

inuevaVariante(**in** n: `nat`, **in** f: `nat`, **in** *puntajeLetras*: `vector(nat)`, **in** *palabraLegitima?*: `diccDigital(vector(letra), bool)`) $\rightarrow$ res : estr
$res.tamanoTablero \leftarrow n$
$res.fichas \leftarrow f$
$res.puntajeLetras \leftarrow puntajeLetras$
$res.palabraLegitima? \leftarrow palabraLegitima?$
Complejidad: $\mathcal{O}(1)$

---

itamañoTablero(**in** v : `estr`) $\rightarrow$ res : nat
$res \leftarrow$ v.tamañoTablero
Complejidad: $\mathcal{O}(1)$

---

i#fichas(**in** v : `estr`) $\rightarrow$ res : nat
$res \leftarrow v.\#fichas$
Complejidad: $\mathcal{O}(1)$

---

ipuntajeLetra(**in** v : `estr`, **in** l : `letra`) $\rightarrow$ res : nat
$res \leftarrow v.puntajeLetras[l]$
Complejidad: $\mathcal{O}(1)$

---

ipuntajeLetras(**in** v : `estr`) $\rightarrow$ res : vector(nat)
$res \leftarrow v.puntajeLetras$
Complejidad: $\mathcal{O}(1)$

---

ipalabraLegitima?(**in** v : `estr`, **in** ls : `cola(letra)`) $\rightarrow$ res : bool
$res \leftarrow definido?(v.palabraLegitima?, ls)$ ▷ $\mathcal{O}(L_{max})$
Complejidad: $\mathcal{O}(L_{max})$

Servicios usados

Diccionario Digital, Vector, Cola

Módulo Diccionario Digital(κ, σ)

Interfaz

parámetros formales
géneros α
función Copiar(**in** a: α) $\rightarrow$ res: α
Pre $\equiv$ {true}
Post $\equiv$ {$res = a$}
Complejidad: $\Theta(copy(a))$
Descripción: función copia de α's

se explica con: Diccionario(κ, σ)

géneros: `diccDigital(`κ, σ`)`

operaciones básicas:

Vacio(**in** A : `nat`, **in** L_{max} : `nat`) $\rightarrow$ res: `diccDigital(`κ, σ`)`
Pre $\equiv$ {True}
Post $\equiv$ {$r\hat{e}s =_{obs}$ vacio}
Complejidad: $\mathcal{O}(\sum_{n=0}^{L_{max}} A^{L_{max}})$
Descripción: Genera un Diccionario Digital vacío.
Aliasing: No presenta aspectos de aliasing.

DEFINIR(**in/out** d : `diccDigital(`κ, σ`)`, **in** $k : \kappa$)
Pre $\equiv \{|k| \leq L_{max}\}$
Post $\equiv \{\hat{d} =_{obs}$ definir($\hat{d}$, $\hat{k}$, True)$\}$
Complejidad: $\mathcal{O}(|k| + copy(s))$
Descripción: Marca como definida una palabra en el diccionario.
Aliasing: No presenta aspectos de aliasing.

DEFINIDO?(**in** d : `diccDigital(`κ, σ`)`, **in** $k : \kappa$) $\rightarrow$ *res*: `bool`
Pre $\equiv$ {True}
Post $\equiv \{r\hat{e}s =_{obs}$ definido?($\hat{d}$, $\hat{k}$)$\}$
Complejidad: $\mathcal{O}(|k|)$
Descripción: Res es true sí sólo sí k está definido en el diccionario.
Aliasing: Presenta aspectos de aliasing.

Implementación

Representación

`diccDigital(`κ, σ`)` **se representa con** `estr`

donde `estr` **es** `tupla(`

rama : `vector(puntero(estr))`,
definida?: `bool`,
palabra: `vector(letra)`)

Rep: estr $\rightarrow$ bool

(∀ e : `estr`, p : `puntero(estr)`, pal : `vector(letra)`, n : `nat`) Rep(e) ≡ true ⇔ ((0 <n <long(e.rama) $\wedge_L$ e.rama[n] = p) $\rightarrow_L$ (long(e.rama) = long(*p.rama) ∧ *p.palabra = agregarAtras(e.palabra, n)))

Abs: $e\hat{s}tr \rightarrow$ dicc(κ, σ) ▷ {Rep(e)}

(∀ e : `estr`) Abs(e) $=_{obs}$ d : dicc(κ, σ) ⇔ (∀ k : κ)(def?(k,d) ⇔ definido?(k,e) ∧ obtener(k,d) $=_{obs}$ definido?(k,e))

Algoritmos

iVACIO(**in** A : `nat`, **in** L_{max} : `nat`) $\longrightarrow$ *res* : `estr`

1: $raiz \leftarrow \langle vector(Null * A), False, vacia() \rangle$ ▷ Tupla con un vector de A punteros a Null, el bool False, y un vector vacío
2: $res \leftarrow completarArbol(raiz)$

Complejidad: $O(\sum_{n=0}^{L_{max}} A^{L_{max}})$ La cantidad de nodos del Trie, que es un arbol completo

iCOMPLETARARBOL(**in/out** *nodo* : `estr`, **in** A : `nat`, **in** L_{max} : `nat`)

1: **si** (tam(nodo$_3$)< L$_{max}$) **entonces**
2: $hijo \leftarrow nodo$ ▷ copia del nodo padre
3: **para** ($l \leftarrow 0$, $l <$ A, $l++$) **hacer**
4: $nodo_1[l] \leftarrow \&hijo$
5: $nodo_1[l]_3 \leftarrow agregarAtras(nodo_3, l)$
6: completarArbol(nodo$_1$[l])

Complejidad: $O(\sum_{n=0}^{L_{max}} A^{L_{max}})$ La cantidad de nodos del Trie, que es un arbol completo

iDEFINIR(**in/out** d : `estr`, **in** $k : \kappa$, **in** $s : \sigma$)

1: $itK \leftarrow$ `crearIt`(k)
2: $nodo \leftarrow_{Ref} d$
3: **mientras** (haySiguiente(itK)) **hacer** ▷ $\mathcal{O}(|k|)$
4: $nodo \leftarrow_{Ref} {}^*nodo_1$[siguiente($itK$)]
5: avanzar(itK)
6: $nodo_2 \leftarrow$ true
7: $=0$

Complejidad: $O(L_{max})$ ya que puede recorrer a lo sumo la palabra mas larga definida

iDEFINIDO?(**in** d : `estr`, **in** $k : \kappa$) $\longrightarrow$ res : `bool`

1: $itK \leftarrow$ `crearIt`(k)
2: $nodo \leftarrow_{Ref} d$
3: **mientras** (haySiguiente(itK) && $\neg$(${}^*nodo_1$[siguiente(itK)] == NULL)) **hacer** ▷ $\mathcal{O}(|k|)$
4: $nodo \leftarrow_{Ref} {}^*nodo_1$[siguiente($itK$)]
5: avanzar(itK)
6: **si** ${}^*nodo_1$[siguiente(itK)] == NULL)) **entonces** ▷ Si |k| > L_{max}
7: $res \leftarrow$ false
8: **else**
9: $res \leftarrow nodo_2$
10: **devolver** res

Complejidad: $O(L_{max})$ ya que puede recorrer a lo sumo la palabra mas larga definida

Servicios usados

VECTOR

2. Diseño del Servidor

Módulo Servidor

Interfaz

parámetros formales
géneros α
función COPIAR(**in** a: α) $\rightarrow$ res: α
Pre $\equiv$ {true}
Post $\equiv$ {$res = a$}
Complejidad: $\Theta(copy(a))$
Descripción: función copia de α's

se explica con: SERVIDOR

géneros: `servidor`

operaciones básicas:

NUEVOSERVIDOR(**in** v: `variante`, **in** $\#e$: `nat` , **in** r: `cola(Letra)`) $\rightarrow$ res: `servidor`
Pre $\equiv$ {longitud(r) $\geq$ tamañoTablero(v)^2 + $\#e$ * #fichas(v) }
Post $\equiv$ {$\hat{res} =_{obs}$ nuevoServidor($\#\hat{e}, \hat{v}, \hat{r}$)}
Complejidad: $\mathcal{O}(N^2 + |\Sigma|K + FK)$
Descripción: Genera un nuevo servidor con un juego nuevo en base a la variante, el repositorio, y la cantidad de

jugadores esperados.
Aliasing: Presenta aspectos de aliasing al pasar por referencia al nuevo juego.

CONECTARCLIENTE(**in/out** s: `servidor`)
Pre $\equiv \{s_0 = s \wedge_L \neg$ empezo?$(s)\}$
Post $\equiv \{\hat{s} =_{obs}$ conectarCliente$(\hat{s_0})\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Agrega un cliente al servidor y prepara su cola de notificaciones.
Aliasing: El servidor entra y sale por referencia.

EMPEZO?(**in** s: `servidor`) $\rightarrow$ res: `bool`
Pre $\equiv$ {True}
Post $\equiv \{\hat{res} =_{obs}$ empezo?$(\hat{s})\}$
Complejidad: $\mathcal{O}(1)$
Descripción: res es true si sólo si empezo el juego del servidor.
Aliasing: No presenta aspectos de aliasing.

CONSULTAR(**in/out** s: `servidor`, **in** id: `cid`)
Pre $\equiv \{s_0 = s \wedge id \leq s.\#$conectados$)\}$
Post $\equiv \{\hat{s} =_{obs}$ consultar$(\hat{s_0}, \hat{id})\}$
Complejidad: $\mathcal{O}(n)$ ▷ n es |cola(notif)| que tenga el id
Descripción: Consultar la cola de notificaciones de un cliente (lo cual vacía dicha cola).
Aliasing: El servidor entra y sale por referencia.

RECIBIRMENSAJE(**in/out** s: `servidor`, **in** id: `cid`, **in** o: `ocurrencia`)
Pre $\equiv \{s_0 = s \wedge id \leq s.\#$conectados$)\}$
Post $\equiv \{\hat{s} =_{obs}$ recibirMensaje$(\hat{s_0}, \hat{id}, \hat{o})\}$
Complejidad: $\mathcal{O}(n)$ ▷ n es |cola(notif)| que tenga el id
Descripción: Recibe un mensaje de un cliente y agrega notificaciones a las colas de los jugadores
Aliasing: El servidor entra y sale por referencia.

#ESPERADOS(**in** s: `servidor`) $\rightarrow$ res: `nat`
Pre $\equiv$ {True}
Post $\equiv \{\hat{res} =_{obs}$ #esperados$(\hat{s})\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el numero de jugadores necesarios para iniciar el juego.
Aliasing: No presenta aspectos de aliasing.

#CONECTADOS(**in** s: `servidor`) $\rightarrow$ res: `nat`
Pre $\equiv$ {True}
Post $\equiv \{\hat{res} =_{obs}$ #conectados$(\hat{s})\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el número de clientes conectados al servidor.
Aliasing: No presenta aspectos de aliasing.

JUEGO(**in** s: `servidor`) $\rightarrow$ res: `juego`
Pre $\equiv$ {True}
Post $\equiv \{\hat{res} =_{obs}$ juego$(\hat{s})\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve el juego que se está jugando en el servidor.
Aliasing: Res es modificable si sólo si s.j es modificable.

Implementación

Representación

`servidor` se representa con `estr`

donde **`estr` es `tupla`**(

#conectados : `nat`,
#esperados: `nat`,
configuracion: `tupla(`v`: variante,` *repositorio*`: cola(letra))`,
j: `juego`,
notificacionesId: `vector(lista(tupla(orden, notificacion)))`,
notificacionesGen: `lista(tupla(orden, notificacion) )`,
contadorNotif: `nat`,
ultimaConsulta: `vector(nat)`)

Rep: estr → bool

(∀ e : `estr`) Rep(e) ≡ true ⇔ e.#conectados ≤ e.#esperados ∧
letrasRepositorioEnPuntajeLetras(e.configuracion.repositorio, puntajeLetras(e.configuracion.v)) ∧
longitud(e.notificacionesId) = e.#esperados ∧ longitud(ultimaConsulta) = e.#esperados ∧
variante(e.j) = e.configuracion.v ∧
desencolarN(e.configuracion.repositorio, longitud(e.configuracion.repostiorio) - longitud(repositorio(e.j))) = repositorio(e.j) ∧ (∀ n:nat)(0 ≤ n <e.#esperados →$_L$ 0 ≤ ultimaConsulta[n] ≤ e.contadorNotif)

LetrasRepositorioEnPuntajeLetras : cola(letra) × vector(nat) ⟶ bool

LetrasRepositorioEnPuntajeLetras($repo, puntajes$) ≡ **if** vacia?($repo$) **then** true **else** 0 ≤ proximo($repo$) < *longitud*($puntajes$) ∧$_L$ LetrasRepositorioEnPuntajeLetras(desencolar($repo$), $puntajes$) **fi**

Abs: $\hat{estr}$ → servidor ▷ {Rep(e)}

(∀ e : `estr`) Abs($\hat{e}$) =$_{obs}$ s : servidor ⇔ #esperados(s)=$_{obs}$ e.#esperados ∧
#conectados(s)=$_{obs}$ e.#conectados ∧
configuracion(s) =$_{obs}$ e.configuracion ∧
juego(s) =$_{obs}$ e.juego ∧
(∀ id: cid)(0 ≤ id <#esperados(s) →$_L$ notificaciones(s, id) =$_{obs}$ consultar(e, id))

Algoritmos

iNUEVOSERVIDOR(**in** v : `variante`, $\#e$: `nat`, r : `cola(letra)`) ⟶ res : `estr`

1: $j \leftarrow$ nuevoJuego($\#e, v, r$) ▷ $\mathcal{O}(N^2 + |\Sigma|K + FK)$
2: $notificacionesId \leftarrow$ vacio() :: vector()
3: $notificacionesGen \leftarrow$ vacio() :: lista()
4: $ultimaConsulta \leftarrow$ vacio() :: vector()
5: **para** (i ← 0, i <#e, i++) **hacer**
6: agregarAtras($notificacionesId$, vacio() :: lista) ▷ $\mathcal{O}(n)$
7: agregarAtras($ultimaConsulta$, 0)
8: $res \leftarrow_{Ref}$ ⟨ 0, $\#e$, $\langle v, r \rangle$, j, $notificacionesId$, $notificacionesGen$, 0, $ultmaConsulta$ ⟩ ▷ $\mathcal{O}(n^2)$
9: **devolver** res

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

iCONECTARCLIENTE(**in**/**out** s : `estr`)

```
1: notificacionesDeId ← vacio():: lista(tupla(orden,notificacion))
2: agregarAdelante(s.notificacionesId[s.#conectados], ⟨ contadorNotif, idCliente(s.#conectados )⟩)
3: contadorNotif++
4: s.#conectados ← s.#conectados + 1
5: si empezo?(s) entonces
6:     agregarAdelante(s.notificacionesGen,⟨ contadorNotif, empezar(tamañoTablero(configuracion_1) )⟩)
7:     contadorNotif++
8:     agregarAdelante(s.notificacionesGen,⟨ contadorNotif, turnoDe(0) ⟩)
9:     contadorNotif++
```

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

iEMPEZO?(**in** s : `estr`) $\longrightarrow$ res : `bool`

```
1: res ← s.#conectados == s.#esperados
2: devolver res
```

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

iCONSULTAR(**in/out** s : `estr`, **in** id : `cid`,) $\longrightarrow$ res : `cola(notificacion)`

1: $notifGen \leftarrow$ vacia() :: lista()
2: $itList \leftarrow$ crearIt(s.notificacionesGen)
3: **para** (i $\leftarrow$ ultimaConsulta[id], i <contadorNotif && haySiguiente(itList), i++) **hacer** ▷ a lo sumo tantas veces como notificaciones por consultar
4: AgregarAdelante(notifGen, siguiente(itList)$_1$)
5: Avanzar(itList)
6: $ultimaConsulta[id] \leftarrow contadorNotif + 1$
7: $notifId \leftarrow s.notificacionesId[id]$
8: $arrNotifId \leftarrow vacio()$
9: $itLid \leftarrow crearIt(notifId)$
10: **mientras** haySiguiente(itLid) **hacer** ▷ $\mathcal{O}(n)$
11: AgregarAtras(arrNotifId, siguiente(itLid))
12: $arrNotifGen \leftarrow vacio()$
13: $itLg \leftarrow crearIt(notifGen)$
14: **mientras** haySiguiente(itLg) **hacer** ▷ $\mathcal{O}(n)$
15: AgregarAtras(arrNotifGen , siguiente(itLg))
16: $itVid \leftarrow crearIt(arrNotifId)$
17: $itVg \leftarrow crearIt(arrNotifGen)$
▷ ambas quedaron ordenadas en base al orden, porque las genéricas, las últimas notificaciones estaban al principio, y quedaron al final. Las notificaciones de id estaban ordenadas.
18: **mientras** haySiguiente(itVid) ó haySiguiente(itVg) **hacer** ▷ $\mathcal{O}(n)$
19: **si** siguiente(itVid).orden <siguiente(itVg).orden **entonces**
20: AgregarAtras(res, siguiente(itVid))
21: avanzar(itVid)
22: **else**
23: AgregarAtras(res, siguiente(itVg))
24: avanzar(itVg)
25: **si** haySiguiente(itVid) **entonces**
26: **mientras** haySiguiente(itVid) **hacer** ▷ $\mathcal{O}(n)$
27: AgregarAtras(res, siguiente(itVid))
28: avanzar(itVid)
29: **else**
30: **si** haySiguiente(itVg) **entonces**
31: **mientras** haySiguiente(itVg) **hacer** ▷ $\mathcal{O}(n)$
32: AgregarAtras(res, siguiente(itVg))
33: avanzar(itVg)
34: $s.notificaciones[id] \leftarrow$ vacio()::lista()
35: **devolver** res

Complejidad: $\mathcal{O}(n)$ donde n es la longitud de la cola de notificación del cliente id

iRECIBIRMENSAJE(**in/out** s : `estr`, **in** id : `cid`, o : `ocurrencia`)

1: **si** empezo?(s) && jugadaValida?(s.j, o) && turno(s.j) == id **entonces**
2: AgregarAdelante(s.notificacionesGen, ⟨ contadorNotif, ubicar(id,o) ⟩)
3: contadorNotif++
4: AgregarAdelante(s.notificacionesGen, ⟨ contadorNotif, sumaPuntos(cid, puntajeObtenido(s.j,o)) ⟩)
5: contadorNotif++
6: $fichas$ ← proximosN(s.j.repositorio, longitud(o))
7: AgregarAtras(s.notificacionesId[id], reponer(fichas)))
8: contadorNotif++
9: AgregarAdelante(s.notificacionesGen, ⟨ contadorNotif, turnoDe(id+1 mod #esperados)⟩)
10: contadorNotif++
11: **else**
12: AgregarAtras(s.notificacionesId[id], mal())
13: contadorNotif++

Complejidad: $\mathcal{O}(n)$ donde n es la longitud de la cola de notificación del cliente id

i#ESPERADOS(**in** s : `estr`) ⟶ res : `nat`

1: res ← s.#esperados
2: **devolver** res

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

3: =0

i#CONECTADOS(**in** s : `estr`) ⟶ res : `nat`

1: res ← s.#conectados
2: **devolver** res

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

iJUEGO(**in** s : `estr`) ⟶ res : `juego`

1: res $\leftarrow_{Ref}$ s.j
2: **devolver** res

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

Servicios usados

NOTIFICACIÓN, VARIANTE, COLA, JUEGO, VECTOR, LISTA

Módulos auxiliares

Módulo Notificación

Interfaz

TAD CID **es** NAT
TAD TIPONOTIF **es** ENUM(`IdCliente`, `Empezar`, `TurnoDe`, `Ubicar`, `Reponer`, `SumaPuntos`, `Mal`)

se explica con: NOTIFICACIÓN

géneros: `notificacion`

operaciones básicas:

idCliente(**in** id: `cid`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ idCliente(id)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificación de tipo idCliente
Aliasing: no presenta aspectos de aliasing

empezar(**in** n: `nat`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ empezar(n)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificacion de tipo empezar
Aliasing: no presenta aspectos de aliasing

turnoDe(**in** id: `cid`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ turnoDe(id)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificación de tipo turnoDe
Aliasing: no presenta aspectos de aliasing

ubicar(**in** id: `cid`, **in** o: `ocurrencia`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ ubicar(id, o)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificacion de tipo ubicar
Aliasing: no presenta aspectos de aliasing

reponer(**in** f : `cola(letra)`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ reponer(f)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificacion de tipo reponer
Aliasing: no presenta aspectos de aliasing

sumaPuntos(**in** id: `cid`, **in** n: `nat`) $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ sumaPuntos(id, n)}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificacion de tipo sumaPuntos
Aliasing: no presenta aspectos de aliasing

mal() $\rightarrow$ res : `notificacion`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ mal()}
Complejidad: $\mathcal{O}(1)$
Descripción: genera una notificacion de tipo mal
Aliasing: no presenta aspectos de aliasing

Implementación

Representación

`notificacion` **se representa con** `estr`

donde `estr` **es** `tupla`(

tipo : `lista(letra)`,
id: `cid`,
n: `nat`,
repositorio: `cola(letra)`,
ocurrencia: `conj(tupla(nat, nat, letra))`
tiposValidos: `conj(lista(letra))`)

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ $e.tipo =$ "*idCliente*" $\vee$ $e.tipo =$ "*empezar*" $\vee$ $e.tipo =$ "*turnoDe*" $\vee$ $e.tipo =$ "*ubicar*" $\vee$ $e.tipo =$ "*reponer*" $\vee$ $e.tipo =$ "*sumaPuntos*" $\vee$ $e.tipo =$ "*mal*"

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

Abs(e) $\equiv$ $(\forall n : notificacion)(datos(n) = e)$

Algoritmos

---

iidCliente(**in** *id*: `cid`) $\rightarrow$ *res* : estr
res $\leftarrow$ $\langle$ "*idCliente*", *cid*, 0, $\emptyset$, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

---

iempezar(**in** n: `nat`) $\rightarrow$ *res* : estr
res $\leftarrow$ $\langle$ "*empezar*", 0, n, $\emptyset$, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

---

iturnoDe(**in** *id*: `cid`) $\rightarrow$ *res* : estr
res $\leftarrow$ $\langle$ "*turnoDe*", *id*, 0, $\emptyset$, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

---

iubicar(**in** *id*: `cid`, **in** o: `ocurrencia`) $\rightarrow$ *res* : estr
res $\leftarrow$ $\langle$ "*ubicar*", *id*, 0, $\emptyset$, $o\rangle$
Complejidad: $\mathcal{O}(1)$

---

ireponer(**in** f : `cola(letra)`) $\rightarrow$ *res* : estr
res $\leftarrow$ $\langle$ "*reponer*", 0, 0, f, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

---

isumaPuntos(**in** id : `cid`, **in** n : `nat`) $\rightarrow$ res : estr
$res \leftarrow \langle$ "$sumaPuntos$", id, n, $\emptyset$, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

imal() $\rightarrow$ res : estr
$res \leftarrow \langle$ "mal", 0, 0, $\emptyset$, $\emptyset\rangle$
Complejidad: $\mathcal{O}(1)$

Servicios usados

Lista, Conjunto Lineal, Cola

Decisiones tomadas

- consideramos a las letras como números naturales desde 0 hasta la longitud del alfabeto menos uno.
- las coordenadas del tablero se ordenan de arriba hacia abajo y de izquierda a derecha. Las palabras deben seguir este orden para ser legítimas.
- asociamos a cada mensaje un valor "orden" que va aumentando con el desarrollo del juego, para que tengan asignado un orden de entrada al servidor.
- **TAD** Jugador **es** Nat
- **TAD** Puntaje **es** Nat
- **TAD** Ocurrencia **es** Conjunto(Tupla(Nat, Nat, Letra))
- **TAD** Jugador **es** Nat
- **TAD** Puntaje **es** Nat
- **TAD** Orden **es** Nat
- **TAD** Cid **es** Nat