Algoritmos y Estructuras de Datos II

Trabajo Práctico 1

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

Integrante	LU	Correo electrónico
Church, Alonso	1/20	alonso@iglesia.com
Lovelace, Ada	10/19	ada_de_los_dientes@tatooine.com
Null, Linda	100/18	null@null.null
Turing, Alan	314/16	halting@problem.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Dudas

- Igualdad observacional sin tenerla definida en los TADs.
- Nuevo Juego: ¿El repo se pasa por parámetro o se genera "aleatoriamente" dentro de la función como un comportamiento automático? Si es el 1º caso se tendría que copiar para coincidir con la complejidad del ejercicio. Si es el segundo ver cómo se genera "aleatoriamente".
- ¿Qué tipo es `letra`? Si es `char` la complejidad de nuevoJuego no se cumple (Σ sería muy grande). En caso de pasar por parámetro un conjunto de `chars` como alfabeto, habría que checkear que todas las fichas del repo inicial sean de ese alfabeto. Lo más probable es que sea un `enum`.
- ¿Se puede usar operaciones de interfaz dentro de **Pre** y **Post**? Por ejemplo para acceder a elemento de tablero, si es otro módulo.
- Si se quiere que `variante` sea un módulo, ¿qué tipo de parámetro de entrada se usan para la operación "nuevaVariante"? ¿Las ya definidas en la representación? ¿Puede ser un módulo que no tenga interfaz y solamente el **Rep** y **Abs**?
- ¿Hay que poner el item "usa" o sólo completar los "requiere"?
- Complejidad de operación #LETRATIENEJUGADOR.

2. Preámbulo

Antes de presentar los módulos, definimos las siguientes variables para las complejidades temporales:

- N — tamaño del tablero.
- K — cantidad de jugadores.
- $|\Sigma|$ — cantidad de letras en el alfabeto.
- F — cantidad de fichas por jugador.
- $L_{\texttt{máx}}$ — longitud de la palabra legítima más larga definida por la variante del juego de la que se trate.

3. Módulo Juego

3.1. Interfaz

se explica con: JUEGO

géneros: `juego`

usa: ??

operaciones:

NUEVOJUEGO(**in** k : `nat`, **in** v : `variante`) $\rightarrow$ res : `juego`
Pre $\equiv \{k > 0\}$
Post $\equiv \{\exists r : \texttt{cola(letra)} \mid res =_{\text{obs}} nuevoJuego(k, v, r)\}$
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: Dada una cantidad de jugadores y una variante de juego, se inicia un nuevo juego con el tablero vacío y con un repositorio de fichas acorde.
Aliasing: ??
Requiere: ??

JUGADAVÁLIDA?(**in** j : `juego`, **in** o : `occurrencia`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} jugadaVálida?(j, o)\}$
Complejidad: $O(L^2_{\texttt{máx}})$
Descripción: Determina si una jugada es válida.

Aliasing: ??

Requiere: ??

UBICAR(**in/out** j : `juego`, **in** o : `occurrencia`)
Pre $\equiv \{jugadaVálida(j, o) \land j =_{\text{obs}} J_0\}$
Post $\equiv \{j =_{\text{obs}} ubicar(J_0, o)\}$
Complejidad: $O(m)$, donde m es el número de fichas que se ubican.
Descripción: Ubica un conjunto de fichas en el juego.

Aliasing: ??

Requiere: ??

VARIANTE(**in** j : `juego`) $\rightarrow$ res : `variante`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} variante(j)\}$
Complejidad: $O(1)$
Descripción: Obtiene información sobre la variante del juego.

Aliasing: ??

Requiere: ??

TURNO(**in** j : `juego`) $\rightarrow$ res : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} turno(j)\}$
Complejidad: $O(1)$
Descripción: Obtiene al jugador del turno actual.

Aliasing: ??

Requiere: ??

PUNTAJE(**in** j : `juego`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} puntaje(j, i)\}$
Complejidad: $O(1 + m \cdot L_{\texttt{máx}})$, donde m es la cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación.
Descripción: Obtiene el puntaje de un jugador.

Aliasing: ??

Requiere: ??

CELDA(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `puntero(letra)`
Pre $\equiv \{enTablero?(tablero(J), i, j)\}$
Post $\equiv \{res =_{\text{obs}}$ **if** $hayLetra?(tablero(J), i, j)$ **then** $\&letra(tablero(J), i, j)$ **else** NULL **fi**$\}$
Complejidad: $O(1)$
Descripción: Obtiene el contenido del tablero en una coordenada (i, j).

Aliasing: ??

Requiere: ??

#LETRATIENEJUGADOR(**in** j : `juego`, **in** x : `letra`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} \#(x, fichas(j, i))\}$
Complejidad: $O(1)$?? $O(F)$
Descripción: Dada una cierta letra x del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra.

Aliasing: ??

Requiere: ??

3.2. Implementación

Representación

juego se representa con juego_estr

donde juego_estr es tupla(

tablero: array_dimensionable(array_dimensionable(puntero(letra)))
, *jugadores*: array_dimensionable(tupla(*puntaje*: nat, *mazo*: array_dimensionable(letra)))
, *jugadorActual*: nat
, *repositorio*: cola(letra)
, *variante*: variante

)

Invariante de Representación

Rep : estr $\longrightarrow$ bool

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

Función de Abstracción

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

Abs(e) $=_{\text{obs}}$ p: foo | bar

Algoritmos

hacerGuia(**in** A : guia, **in** *parámetroInútil* : Nat) $\longrightarrow$ bool

```
1:  i ← 0                                                    ▷ esto es Θ(1)
2:  n ← guia.cantEjercicios()                                ▷ O(1)
3:  consultas ← diccVacio
4:  prepararMate()                                           ▷ Ω(n^n)
5:  while i < n do
6:      pensarEjercicio(i)
7:      if tengoConsultas(i) then
8:          escribirConsultasEjercicio(i,consultas)
9:      else
10:         comerBizochito()
11:     comerBizochito()
12: for miVariable do
13:     hacer algo
14: return vacio?(consultas)
```

4. Módulo Servidor

4.1. Interfaz

se explica con: SERVIDOR

géneros: `servidor`

usa: ??

operaciones:

NUEVOSERVIDOR(**in** k : `nat`, **in** v : `variante`) $\rightarrow$ res : `servidor`
Pre $\equiv$ {true}
Post $\equiv$ $\{\exists r : \texttt{cola(letra)} \mid res =_{\text{obs}} nuevoServidor(k, v, r)\}$
Complejidad: $O(N^2 + |\Sigma|K + FK)$
Descripción: Dada una cantidad de jugadores y una variante de juego, se inicia un nuevo servidor y una nueva partida de juego.
Aliasing: ??
Requiere: ??

CONECTAR(**in/out** s : `servidor`)
Pre $\equiv$ $\{\neg empezó?(s) \land s =_{\text{obs}} S_0\}$
Post $\equiv$ $\{s =_{\text{obs}} conectarCliente(S_0)\}$
Complejidad: $O(1)$
Descripción: Conecta un cliente a un servidor.
Aliasing: ??
Requiere: ??

CONSULTAR(**in/out** s : `servidor`, **in** cid : `nat`)
Pre $\equiv$ $\{cid \leq \#conectados(s) \land s =_{\text{obs}} S_0\}$
Post $\equiv$ $\{s =_{\text{obs}} consultar(S_0, cid)\}$
Complejidad: $O(n)$, donde n es la cantidad de mensajes en la cola de dicho cliente.
Descripción: Consulta la cola de notificaciones de un cliente (lo cual vacía dicha cola).
Aliasing: ??
Requiere: ??

RECIBIR(**in/out** s : `servidor`, **in** cid : `nat`, **in** o : `ocurrencia`)
Pre $\equiv$ $\{cid \leq \#conectados(s) \land s =_{\text{obs}} S_0\}$
Post $\equiv$ $\{s =_{\text{obs}} recibirMensaje(S_0, cid, o)\}$
Complejidad: ??
Descripción: Recibe un mensaje de un cliente.
Aliasing: ??
Requiere: ??

CLIENTESESPERADOS(**in** s : `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \#esperados(s)\}$
Complejidad: $O(1)$
Descripción: Obtiene el número de clientes esperados.
Aliasing: ??
Requiere: ??

CLIENTESCONECTADOS(**in** s : `servidor`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \#conectados(s)\}$
Complejidad: $O(1)$
Descripción: Obtiene el número de clientes conectados.
Aliasing: ??
Requiere: ??

PARTIDA(**in** s : `servidor`) $\rightarrow$ res : `juego`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} juego(s)\}$
Complejidad: $O(1)$
Descripción: Obtiene el juego que se está jugando en el servidor.
Aliasing: ??
Requiere: ??

4.2. Implementación

Representación

`servidor` **se representa con** `servidor_estr`

donde `servidor_estr` es `tupla`(

cantJugadores: `nat`
, *mensajes*: `array_dimensionable(ocurrencia)`
, *notificaciones*: `dicc(cliente,cola(notificacion))`

??

)

5. Módulos auxiliares

5.1. Módulo Variante

5.1.1. Interfaz

se explica con: VARIANTE

géneros: `variante`

usa: ??

operaciones:

NUEVAVARIANTE(

in n : `nat`,
in f : `nat`,
in *puntajes* : `dicc(letra, nat)`,
in *legítimas* : `conj(secu(letra))`

) $\rightarrow$ *res* : `variante`
Pre $\equiv \{n > 0 \land f > 0\}$
Post $\equiv \{res =_{\text{obs}} nuevaVariante(n, f, puntajes, legítimas)\}$
Complejidad: $O(1)$
Descripción: Dada una cantidad de jugadores y una variante de juego, se inicia un nuevo juego con el tablero vacío y con un repositorio de fichas acorde.
Aliasing: ??
Requiere: ??

JUGADAVÁLIDA?(**in** j : `juego`, **in** o : `occurrencia`) $\rightarrow$ *res* : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} jugadaVálida?(j, o)\}$
Complejidad: $O(L^2_{\texttt{máx}})$
Descripción: Determina si una jugada es válida.
Aliasing: ??
Requiere: ??

UBICAR(**in/out** j : `juego`, **in** o : `occurrencia`)
Pre $\equiv \{jugadaVálida(j, o) \land j =_{\text{obs}} J_0\}$
Post $\equiv \{j =_{\text{obs}} ubicar(J_0, o)\}$
Complejidad: $O(m)$, donde m es el número de fichas que se ubican.
Descripción: Ubica un conjunto de fichas en el juego.
Aliasing: ??
Requiere: ??

VARIANTE(**in** j : `juego`) $\rightarrow$ *res* : `variante`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} variante(j)\}$
Complejidad: $O(1)$
Descripción: Obtiene información sobre la variante del juego.
Aliasing: ??
Requiere: ??

TURNO(**in** j : `juego`) $\rightarrow$ *res* : `nat`
Pre $\equiv \{\text{true}\}$

Post $\equiv \{res =_{\text{obs}} turno(j)\}$
Complejidad: $O(1)$
Descripción: Obtiene al jugador del turno actual.
Aliasing: ??
Requiere: ??

PUNTAJE(**in** j : `juego`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} puntaje(j, i)\}$
Complejidad: $O(1 + m \cdot L_{\texttt{máx}})$, donde m es la cantidad de fichas que ubicó el jugador desde la última vez que se invocó a esta operación.
Descripción: Obtiene el puntaje de un jugador.
Aliasing: ??
Requiere: ??

CELDA(**in** J : `juego`, **in** i : `nat`, **in** j : `nat`) $\rightarrow$ res : `puntero(letra)`
Pre $\equiv \{enTablero?(tablero(J), i, j)\}$
Post $\equiv \{res =_{\text{obs}}$ **if** $hayLetra?(tablero(J), i, j)$ **then** $\&letra(tablero(J), i, j)$ **else** NULL **fi**$\}$
Complejidad: $O(1)$
Descripción: Obtiene el contenido del tablero en una coordenada (i, j).
Aliasing: ??
Requiere: ??

#LETRATIENEJUGADOR(**in** j : `juego`, **in** x : `letra`, **in** i : `nat`) $\rightarrow$ res : `nat`
Pre $\equiv \{i < \#jugadores(j)\}$
Post $\equiv \{res =_{\text{obs}} \#(x, fichas(j, i))\}$
Complejidad: $O(1)$?? $O(F)$
Descripción: Dada una cierta letra x del alfabeto, conocer cuántas fichas tiene un jugador de dicha letra.
Aliasing: ??
Requiere: ??

5.1.2. Implementación

Representación `juego` se representa con `juego_estr`

donde `juego_estr` es `tupla`(

tablero: `array_dimensionable(array_dimensionable(puntero(letra)))`
, *jugadores*: `array_dimensionable(tupla(`*puntaje*: `nat`, *mazo*: `array_dimensionable(letra)))`
, *jugadorActual*: `nat`
, *repositorio*: `cola(letra)`
, *variante*: `variante`

)

Invariante de Representación

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\iff$ foo

Función de Abstracción

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

Abs(e) =$_{\text{obs}}$ p: foo | bar

Algoritmos

HACERGUIA(**in** A : `guia`, **in** *parámetroInútil* : `Nat`) $\longrightarrow$ `bool`

```
1:  i ← 0                                                          ▷ esto es Θ(1)
2:  n ← guia.cantEjercicios()                                      ▷ O(1)
3:  consultas ← DICCVACIO
4:  PREPARARMATE()                                                 ▷ Ω(n^n)
5:  while i < n do
6:      PENSAREJERCICIO(I)
7:      if TENGOCONSULTAS(i) then
8:          ESCRIBIRCONSULTASEJERCICIO(i,consultas)
9:      else
10:         COMERBIZOCHITO()
11:     COMERBIZOCHITO()
12: for miVariable do
13:     hacer algo
14: return VACIO?(consultas)
```