Tipos abstractos de datos básicos

Algoritmos y Estructuras de Datos II, DC, UBA.

Casillero es Nat

Movimiento es Nat

Continente es String

Jugador es Nat

1. TAD Tablero

TAD Tablero

géneros tablero

usa nat, bool, continente, casillero, multiconjExt(α), conj(α), movimiento

exporta tablero, generadores, observadores, continentes, todosLosMovs

igualdad observacional

$$(\forall t, t' : \text{tablero}) \left(t =_{\text{obs}} t' \iff \begin{pmatrix} (\#casilleros(t) =_{\text{obs}} \#casilleros(t')) \wedge_{\text{L}} \\ (\forall (c, c' : nat))c, c' \leq \#casilleros(t) \Rightarrow_{\text{L}} \\ (cont(c, t) =_{\text{obs}} cont(c, t') \wedge \\ movsDesdeHasta(c, c', t) =_{\text{obs}} movsDesdeHasta(c, c', t'))) \end{pmatrix} \right)$$

generadores

//Crea un tablero con dos casilleros. El 1 es del primer continente, el 2 del segundo; el primer movimiento va desde el primer casillero hasta el segundo, y el segundo movimiento hace lo opuesto.

crearTablero : continente $\times$ continente $\times$ mov $\times$ mov $\longrightarrow$ tablero

//Agrega un casillero del continente k conectado por el movimiento m al casillero c. El movimiento m' conecta c al casillero creado. Las restricciones en k aseguran que se cumpla el agrupamiento de continentes, las restricciones sobre m y m' aseguran que se cumpla la unicidad de movimientos y la necesidad de dos movimientos asegura que se cumpla la simetría.

agregarCasillero : casillero c $\times$ continente k $\times$ mov m $\times$ mov m' $\times$ tablero t $\longrightarrow$ tablero

$$\left\{ \begin{array}{l} c \leq \#\text{casilleros}(t) \wedge_{\text{L}} (k =_{\text{obs}} cont(c, t) \vee ((\forall c' : nat) c' \leq \#\text{casilleros}(t) \Rightarrow_{\text{L}} cont(c', t) \neq k)) \wedge_{\text{L}} \\ m' \notin \text{todosLosMovs}(c, t) \end{array} \right\}$$

//Conecta los casilleros pasados como parámetros con el movimiento m del primero al segundo y m' del segundo al primero.

conectar : casillero c $\times$ casillero c' $\times$ mov m $\times$ mov m' $\times$ tablero t $\longrightarrow$ tablero

$\{c, c' \leq \#\text{casilleros}(t) \wedge c \neq c' \wedge_{\text{L}} m \notin \text{todosLosMovs}(c, t) \wedge_{\text{L}} m' \notin \text{todosLosMovs}(c', t)\}$

//Agrega un movimiento en un solo sentido. Requiere que los casilleros ya estén conectados para que no se rompa la simetría.

agregarFlecha : casillero c $\times$ casillero c' $\times$ mov m $\times$ tablero t $\longrightarrow$ tablero

$\{c, c' \leq \#\text{casilleros}(t) \wedge_{\text{L}} \text{conectados?}(c, c', t) \wedge m \notin \text{todosLosMovs}(c, t)\}$

observadores básicos

#casilleros : tablero $\longrightarrow$ nat

cont : casillero c $\times$ tab t $\longrightarrow$ continente $\{c \leq \#\text{casilleros(t)}\}$

movsDesdeHasta : casillero c × casillero c' × tab t ⟶ conj(mov) $\{c, c' \leq \#\text{casilleros(t)}\}$

otras operaciones

//todosLosMovs: devuelve un conjunto con todos los movimientos que salen del casillero c a cualquier casillero del tablero.

todosLosMovs : casillero c × tab t ⟶ conj(mov) $\{c \leq \#\text{casilleros(t)}\}$

conectados? : casillero c × casillero c' × tab t ⟶ bool $\{c, c' \leq \#\text{casilleros(t)}\}$

continentes : tablero ⟶ conj(continente)

dameContinentes : nat × tablero ⟶ conj(continente)

axiomas ∀ t: tablero

#casilleros(crearTablero(k, k', m, m')) ≡ 2

#casilleros(agregarCasillero(c, k, m, m', t)) ≡ suc(#casilleros(t))

#casilleros(conectar(c, c', m, m', t)) ≡ #casilleros(t)

#casilleros(agregarFlecha(c, c', m, t)) ≡ #casilleros(t)

cont(c,crearTablero(k, k', m, m')) ≡ **if** $c = 1$ **then** k **else** k' **fi**

cont(c,agregarCasillero($\tilde{c}, k, m, m', t$)) ≡ **if** c = suc(#casilleros(t)) **then** k **else** cont(c, t) **fi**

cont(c,conectar($\tilde{c}, \tilde{c}', m, m', t$)) ≡ cont($c, t$)

cont(c,agregarFlecha($\tilde{c}, \tilde{c}', m, t$)) ≡ cont($c, t$)

```
movsDesdeHasta(c, c', crearTablero(k, k', m, m')) ≡ if c = 1 ∧ c' = 2 then
                                                       {m}
                                                    else
                                                       if c = 2 ∧ c' = 1 then {m'} else ∅ fi
                                                    fi
movsDesdeHasta(c, c', agregarCasillero(c̃, k, m, m', t)) ≡ if c = suc(#casilleros(t)) then
                                                             if c' = c̃ then {m} else ∅ fi
                                                          else
                                                             if c = c̃ ∧ c' = suc(#casilleros(t)) then
                                                                {m'}
                                                             else
                                                                movsDesdeHasta(c, c', t)
                                                             fi
                                                          fi
movsDesdeHasta(c, c', conectar(c̃, c̃', m, m', t)) ≡ if c = c̃ ∧ c' = c̃' then
                                                     {m} ∪ movsDesdeHasta(c, c', t)
                                                  else
                                                     if c = c̃' ∧ c' = c̃ then
                                                        {m'} ∪ movsDesdeHasta(c, c', t)
                                                     else
                                                        movsDesdeHasta(c, c', t)
                                                     fi
                                                  fi
movsDesdeHasta(c, c', agregarFlecha(c̃, c̃', m, t)) ≡ if c = c̃ ∧ c' = c̃' then
                                                      {m} ∪ movsDesdeHasta(c, c', t)
                                                   else
                                                      movsDesdeHasta(c, c', t)
                                                   fi
```

todosLosMovs(c,crearTablero(k, k', m, m')) ≡ **if** $c = 1$ **then** $\{m\}$ **else** $\{m'\}$ **fi**

todosLosMovs(c,agregarCasillero($\tilde{c}, k, m, m', t$)) $\equiv$ **if** $c =$ suc(#casilleros(t)) **then**
    $\{m\}$
else
    if $c = \tilde{c}$ **then**
        $\{m'\} \cup$ todosLosMovs(c, t)
    else
        todosLosMovs(c, t)
    fi
fi

todosLosMovs(c,conectar($\tilde{c}, \tilde{c}', m, m', t$)) $\equiv$ **if** $c = \tilde{c}$ **then**
    $\{m\} \cup$ todosLosMovs(c, t)
else
    if $c = \tilde{c}'$ **then**
        $\{m'\} \cup$ todosLosMovs(c, t)
    else
        todosLosMovs(c, t)
    fi
fi

todosLosMovs(c,agregarFlecha($\tilde{c}, \tilde{c}', m, t$)) $\equiv$ **if** $c = \tilde{c}$ **then**
    $\{m\} \cup$ todosLosMovs(c, t)
else
    todosLosMovs(c, t)
fi

conectados?(c, c', t) $\equiv$ $\neg\emptyset$?(movsDesdeHasta(c, c', t))

continentes(t) $\equiv$ dameContinentes(#casilleros(t), t)

dameContinentes(n, t) $\equiv$ **if** $n = 0$ **then** $\emptyset$ **else** $\{$cont(n, t)$\} \cup$ dameContinentes($n - 1, t$) **fi**

Fin TAD

2. TAD Partida

TAD Partida

géneros partida

exporta partida, generadores, observadores, jugadoresActivos, jugadoresEliminados, terminada?, ganador, casillerosDominados, casillerosDisputados, casillerosVacíos

usa Bool, Casillero, Jugador, Tablero, Movimiento, Nat, Multiconj(α), Conj(α), Continente

igualdad observacional

$$(\forall p, p' : \text{partida}) \left(p =_{\text{obs}} p' \Longleftrightarrow \begin{pmatrix} (\text{tablero}(p) =_{\text{obs}} \text{tablero}(p') \wedge \\ \#\text{jugadores}(p) =_{\text{obs}} \#\text{jugadores}(p') \wedge \\ ((\forall c : nat) c \leq \#\text{casilleros}(\text{tablero}(p)) \Rightarrow_{\text{L}} \\ \text{fichasEnCasillero}(c, p) =_{\text{obs}} \text{fichasEnCasillero}(c, p')) \wedge \\ ((\forall j : nat) j \leq \#\text{jugadores}(p) \Rightarrow_{\text{L}} \\ (\text{misión}(j, p) =_{\text{obs}} \text{misión}(j, p)) \wedge \\ \text{fichasPuestas}(j, p) =_{\text{obs}} \text{fichasPuestas}(j, p')) \end{pmatrix} \right)$$

observadores básicos

tablero : partida $\longrightarrow$ tablero

#jugadores : partida $\longrightarrow$ nat

//fichasEnCasillero: las fichas de cada jugador están representadas por su cardinal en un multiconjunto. Una aparición de j en fichasEnCasillero representa una ficha de j en el casillero dado. Esta convención con los multiconjuntos se mantendrá durante toda la especificación para representar las fichas en cada casillero.

fichasEnCasillero : casillero c × partida p ⟶ multiconjExt(jugador) $\{c \leq \#\text{casilleros}(\text{tablero}(p))\}$

misión : jugador j × partida p ⟶ continente $\{j \leq \#\text{jugadores}(p)\}$

//fichasPuestas: devuelve la cantidad de fichas que puso j a lo largo de todo el partido.

fichasPuestas : jugador j × partida p ⟶ nat $\{j \leq \#\text{jugadores}(p)\}$

generadores

//crearPartida: crea una nueva partida cuyo tablero es t. La cantidad de jugadores es js y las secuencias cs y ks indican el casillero donde coloca la primera ficha y el continente que cada jugador debe conquistar respectivamente. Al i-ésimo jugador le corresponde la (i-1)-ésima posición de cada secuencia, ya que el primer jugador es el 1 y los índices de las secuencias empiezan en 0.

crearPartida : tablero t × nat js × secuExt(casillero) cs × secuExt(continente) ks ⟶ partida

$$\left\{\begin{array}{l} 2 \leq js \wedge \text{long}(cs) = \text{long}(ks) = js \wedge \text{sinRepetidos}(cs) \wedge ((\forall k : string)\text{está?}(k, ks) \Rightarrow \\ k \in \text{dameConts}(t)) \wedge ((\forall c : nat)\text{está?}(c, cs) \Rightarrow c \leq \#\text{casilleros}(t)) \end{array}\right\}$$

//agregarFicha: agrega una ficha del jugador j en el casillero c. Requiere que esté vacío o dominado por j.

agregarFicha : jugador j × casillero c × partida p ⟶ partida

$$\left\{\begin{array}{l} j \leq \#\text{jugadores}(p) \wedge_{\text{L}} \text{estáActivo?}(j, p) \wedge \neg\text{terminada?}(p) \wedge c \leq \#\text{casilleros}(\text{tablero}(p)) \wedge_{\text{L}} \\ \text{jugadoresEnCasillero}(c, p) = \emptyset \vee \text{jugadoresEnCasillero}(c, p) = \{j\} \end{array}\right\}$$

//mover: el jugador j realiza la acción de movimiento m con n fichas. El funcionamiento se detalla más profundamente en los axiomas.

mover : jugador j × movimiento m × nat n × partida p ⟶ partida

$\{j \leq \#\text{jugadores}(p) \wedge \text{estáActivo?}(j, p) \wedge \neg\text{terminada?}(p)\}$

otras operaciones

Funciones requeridas por la empresa

jugadoresActivos : partida ⟶ conj(jugador)

jugadoresEliminados : partida ⟶ conj(jugador)

terminada? : partida ⟶ bool

ganador : partida ⟶ jugador $\{\text{terminada?}(p)\}$

casillerosDominados : partida ⟶ conj(tupla(casillero,conj(jugador)))

casillerosDisputados : partida ⟶ conj(tupla(casillero,conj(jugador)))

casillerosVacíos : partida ⟶ conj(casillero)

Funciones auxiliares

//fichasVecinasDeJ: dado un casillero c, devuelve un multiconjunto con todas las fichas de j de todos los casilleros del tablero que con el movimiento m podrían llegar a c.

fichasVecinasDeJ : casillero c × jugador j × movimiento m × partida p ⟶ multiconjExt(jugador)

$\{c \leq \#\text{casilleros}(\text{tablero}(p))\}$

estáActivo? : jugador j × partida p ⟶ bool $\{j \leq \#\text{jugadores}(p)\}$

tieneFichasEnAlgúnCasillero? : jugador j × nat n × partida p ⟶ bool $\{j \leq \#\text{jugadores}(p)\}$

dameActivos : partida × nat ⟶ conj(jugador)

dameEliminados : partida × nat ⟶ conj(jugador)

algunoCompletóLaMisión? : nat × partida ⟶ bool

completóLaMisión? : jugador j × partida p ⟶ bool $\{j \leq \#\text{jugadores}(p)\}$

cuántosLeFaltan : jugador j × partida p ⟶ nat $\{j \leq \#\text{jugadores}(p)\}$

//contarNoDominadosHasta: esta función sirve de auxiliar para saber si el jugador j completó o no su misión. Al hacer recursión sobre el parámetro n, devuelve la cantidad de casilleros numerados entre 1 y n que pertenecen al continente que j debe conquistar y no son dominados por j.

contarNoDominadosHasta : jugador j × nat n × partida p ⟶ nat {$j \leq$ #jugadores(p)}

estáDominado? : casillero c × partida p ⟶ bool {$c \leq$ #casilleros(tablero(p))}

estáDisputado? : casillero c × partida p ⟶ bool {$c \leq$ #casilleros(tablero(p))}

jugadoresEnCasillero : casillero c × partida p ⟶ conj(jugador) {$c \leq$ #casilleros(tablero(p))}

dominadoPor : jugador j × casillero c × partida p ⟶ bool
{$j \leq$ #jugadores(p) $\wedge$ $c \leq$ #casilleros(tablero(p))}

dameDominados : nat × partida ⟶ conj(casillero)

dameDisputados : nat × partida ⟶ conj(casillero)

dameVacíos : nat × partida ⟶ conj(casillero)

maxiFourcade : nat × partida ⟶ jugador

axiomas ∀ p: partida

Observadores

tablero(crearPartida(t, js, cs, ks)) ≡ t

tablero(agregarFicha(j, c, p)) ≡ tablero(p)

tablero(mover(j, m, n, p)) ≡ tablero(p)

#jugadores(crearPartida(t, js, cs, ks)) ≡ js

#jugadores(agregarFicha(j, c, p)) ≡ #jugadores(p)

#jugadores(mover(j, m, n, p)) ≡ #jugadores(p)

fichasEnCasillero(c, crearPartida(t, js, cs, ks)) ≡ **if** está?(c, cs) **then** {suc(posición(c, cs))} **else** ∅ **fi**

fichasEnCasillero(c, agregarFicha($j, \tilde{c}, p$)) ≡ fichasEnCasillero(c, p) ∪ (**if** $c = \tilde{c}$ **then** {j} **else** ∅ **fi**)

//fichasEnCasillero: si el casillero es dominado por el jugador que hizo el movimiento, tendrá n fichas menos de éste, o 0 en caso de que hubiera menos de n fichas. Si el casillero estaba disputado, algún jugador tendrá una ficha menos. En cualquier caso se agregan todas las fichas posibles de j a través de los casilleros dominados por j que se conecten a c con el movimiento m.

fichasEnCasillero(c, mover(j, m, n, p)) ≡ **if** estáDominado?(c, p) **then**
  if j ∈ fichasEnCasillero(c,p) ∧ m ∈ todosLosMovs(c,tablero(p))
  then
    (fichasEnCasillero(c, p) - agNVeces($j, n, \emptyset$)) ∪
    fichasVecinasDeJ(c, j, m, p)
  else
    fichasEnCasillero(c, p) ∪ fichasVecinasDeJ(c, j, m, p)
  fi
else
  if estáDisputado?(fichasEnCasillero(c, p)) **then**
    sinUno(fichasEnCasillero(c, p)) ∪ fichasVecinasDeJ(c, j, m, p)
  else
    fichasVecinasDeJ(c, j, m, p)
  fi
fi

misión(j,crearPartida(t, js, cs, ks)) ≡ $ks[j-1]$

misión(j,agregarFicha(j, c, p)) ≡ misión(j, p)

misión(j,mover(j, m, n, p)) ≡ misión(j, p)

fichasPuestas(j,crearPartida(t, js, cs, ks)) ≡ 1

fichasPuestas(j,agregarFicha($\tilde{j}, c, p$)) ≡ fichasPuestas(j, p) + (**if** $j = \tilde{j}$ **then** 1 **else** 0 **fi**) fichasPuestas(j,mover(t, js, cs, ks)) $\equiv$ fichasPuestas(j, p)

Funciones requeridas por la empresa

jugadoresActivos(p) $\equiv$ dameActivos(p,#jugadores(p))

jugadoresEliminados(p) $\equiv$ dameEliminados(p,#jugadores(p))

terminada?(p) $\equiv$ #(jugadoresActivos(p))= 1 $\vee$ algunoCompletóLaMisión?(#jugadores(p),p)

ganador(p) $\equiv$ maxiFourcade(#jugadores(p),p)

casillerosDominados(p) $\equiv$ dameDominados(#casilleros(tablero(p)),p)

casillerosDisputados(p) $\equiv$ dameDisputados(#casilleros(tablero(p)),p)

casillerosVacíos(p) $\equiv$ dameVacíos(#casilleros(tablero(p)),p)

Funciones auxiliares

fichasVecinasDeJ(j, cn, m, c, n, p) $\equiv$ **if** $0 < cn$ **then**
    if dominadoPor(j, cn, p) $\wedge$ $m \in$ movsDesdeHasta(cn, c,tablero(p))
    then
        agNVeces(j,minimo(#(j,fichasEnCasillero(cn,tablero(p))), n), $\emptyset$)
        $\cup$ fichasVecinasDeJ($j, cn - 1, m, c, n, p$)
    else
        fichasVecinasDeJ($j, cn - 1, m, c, n, p$)
    fi
else
    $\emptyset$
fi

estáActivo?(j) $\equiv$ tieneFichasEnAlgúnCasillero(j,#casilleros(tablero(p)),p)

tieneFichasEnAlgúnCasillero(j, n, p) $\equiv$ **if** $n = 0$ **then**
    false
else
    $0 <$ fichasEnCasillero(j, n, p)$\vee$
    tieneFichasEnAlgúnCasillero($j, n - 1, p$)
fi

dameActivos(p, n) $\equiv$ **if** $n = 0$ **then**
    $\emptyset$
else
    if estáActivo?(n, p) **then**
        Ag(n,dameActivos($p, n - 1$))
    else
        dameActivos($p, n - 1$)
    fi
fi

dameEliminados(p, n) $\equiv$ **if** $n = 0$ **then**
    $\emptyset$
else
    if $\neg$ estáActivo?(n, p) **then**
        Ag(n,dameEliminados($p, n - 1$))
    else
        dameEliminados($p, n - 1$)
    fi
fi

algunoCompletóLaMisión?(n, p) $\equiv$ **if** $n = 0$ **then**
    false
else
    completóLaMisión?(n, p) $\vee$ algunoCompletóLaMisión?($n - 1, p$)
fi

completóLaMisión?(j, p) $\equiv$ cuántosLeFaltan(j, p) $= 0$

cuántosLeFaltan(j, p) $\equiv$ contarNoDominadosHasta(j,#casilleros(tablero(p)),p)

contarNoDominadosHasta(j, n, p) $\equiv$ **if** $\neg$ dominadoPor$(j, n, p) \wedge$ cont(n,tablero(p)) = misión(j, p) **then**
suc(contarNoDominadosHasta$(j, n-1, p)$)
else
contarNoDominadosHasta$(j, n-1, p)$
fi

dominadoPor(j, n, p) $\equiv$ estáDominado?(c, p) $\wedge j \in$ jugadoresEnCasillero(c, p)

estáDominado?(c, p) $\equiv$ #(jugadoresEnCasillero(c, p)) = 1

estáDisputado?(c, p) $\equiv$ #(jugadoresEnCasillero(c, p)) > 1

jugadoresEnCasillero(c, p) $\equiv$ aConj(fichasEnCasillero(c, p))

dameDominados(n, p) $\equiv$ **if** $n = 0$ **then**
$\emptyset$
else
if estáDominado?(n,p) **then**
$\{\langle n, \text{jugadoresEnCasillero}(n, p)\rangle\} \cup$ dameDominados$(n-1, p)$
else
dameDominados$(n-1, p)$
fi
fi

dameDisputados(n, p) $\equiv$ **if** $n = 0$ **then**
$\emptyset$
else
if estáDisputado?(n,p) **then**
$\{\langle n, \text{jugadoresEnCasillero}(n, p)\rangle\} \cup$ dameDisputados$(n-1, p)$
else
dameDisputados$(n-1, p)$
fi
fi

dameVacíos(n, p) $\equiv$ **if** $n = 0$ **then**
$\emptyset$
else
if $\emptyset$?(jugadoresEnCasillero(n, p)) **then**
$\{n\} \cup$ dameVacíos$(n-1, p)$
else
dameVacíos$(n-1, p)$
fi
fi

//La función maxiFourcade te devuelve al más winner de todos. Si no lo conocés, googlealo.

maxiFourcade(n, p) $\equiv$ **if** $n = 1$ *or* completóLaMisión?(n, p) **then** n **else** maxiFourcade$(n-1, p)$ **fi**

Fin TAD

3. Extensiones de otros TADs

3.1. TAD SecuExt extiende Secuencia

TAD SecuExt

(...)

otras operaciones

$\bullet[\bullet]$: secuExt(α) $\times$ nat $\longrightarrow$ α $\{n < \text{long}(s)\}$

sinRepetidos? : secuExt(α) $\longrightarrow$ bool

axiomas

s[n] $\equiv$ **if** $n = 0$ **then** prim(s) **else** fin(s)[$n-1$] **fi**

sinRepetidos?(s) $\equiv$ **if** vacía?(s) **then** true **else** $\neg$ está?(prim(s),fin(s)) $\wedge$ sinRepetidos?(fin(s)) **fi**

Fin TAD

3.2. TAD MulticonjExt extiende Multiconjunto

TAD multiconjExt

(...)

otras operaciones

agNVeces : $\alpha \times$ nat $\times$ multiconjExt(α) $\longrightarrow$ multiconjExt(α)

aConj : multiconjExt(α) $\longrightarrow$ conj(α)

axiomas

agNVeces(a, n, c) $\equiv$ **if** $n = 0$ **then** c **else** Ag(a,agNVeces($a, n-1, c$)) **fi**

aConj(c) $\equiv$ **if** $\emptyset$?(c) **then** $\emptyset$ **else** Ag(dameUno(c),aConj(sinUno(c))) **fi**

Fin TAD