DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Recuperatorio Trabajo Práctico 1

Especificación

25 de septiembre de 2016

Algoritmos y Estructuras de Datos II
Segundo Cuatrimestre de 2016

Grupo "Algo Habrán Hecho (por las Estructuras de Datos)"

Integrante	LU	Correo electrónico
Barylko, Roni Ariel	750/15	rbarylko@dc.uba.ar
Giudice, Carlos	694/15	cgiudice@dc.uba.ar
Szperling, Sebastián Ariel	763/15	sszperling@dc.uba.ar
Tarrío, Ignacio	363/15	itarrio@dc.uba.ar

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. TADs Auxiliares

1.1. Renombres

TAD Posicion es Tupla(NatExtendido, NatExtendido)
TAD Pokemon es String
TAD Jugador es Nat

1.2. Extensiones

TAD NatExtendido

exporta $\bullet \div \bullet$, $\sqrt{\bullet}$

extiende Nat

otras operaciones

$\bullet \div \bullet$: nat n × nat m $\longrightarrow$ nat $\{0 < m\}$
$\sqrt{\bullet}$: nat $\longrightarrow$ nat
encontrarRaiz : nat n × nat m $\longrightarrow$ nat $\{m \leq n\}$

axiomas $\forall\ a, b$: nat
a ÷ b ≡ **if** a < b **then** 0 **else** 1 + ((a-b) ÷ b) **fi**
$\sqrt{a}$ ≡ encontrarRaiz(a,a)
encontrarRaiz(a, b) ≡ **if** a < b × b **then** encontrarRaiz(a, pred(b)) **else** b **fi**

Fin TAD

TAD MulticonjuntoExtendido(α)

usa Conjunto(α)

exporta partes, $\bullet - \bullet$

extiende Multiconjunto(α)

otras operaciones

partes : multiconj(α) $\longrightarrow$ conj(multiconj(α))
agregarATodos : α × conj(multiconj(α)) $\longrightarrow$ conj(multiconj(α))
$\bullet - \bullet$: multiconj(α) × multiconj(α) $\longrightarrow$ multiconj(α)

axiomas $\forall a$: α, $\forall\ as, bs$: multiconj(α), $\forall\ cas$: conj(multiconj(α))

partes(as) ≡ **if** vacio?(as) **then**
Ag(∅,∅)
else
agregarATodos(dameUno(as), partes(sinUno(as))) ∪ partes(sinUno(as))
fi

agregarATodos(a, cas) ≡ **if** vacio?(cas) **then**
∅
else
Ag(Ag(a, dameUno(cas)), agregarATodos(a, sinUno(cas))
fi

as − bs ≡ **if** vacio?(bs) **then** as **else** (as − {dameUno(bs)}) − sinUno(bs) **fi**

Fin TAD

2. TAD Mapa

TAD MAPA

géneros mapa

exporta mapa, generadores, observadores, conexion?, distancia

usa NATEXTENDIDO, POSICION, BOOL, CONJUNTO(α)

igualdad observacional

$$(\forall m, m' : \text{mapa}) \left(m =_{\text{obs}} m' \iff \begin{pmatrix} \text{posiciones(m)} =_{\text{obs}} \text{posiciones(m')} \wedge \\ (\forall\ \text{p} : \text{posiciones(m)})\ (\text{conexionesDirectas(p,m)} =_{\text{obs}} \text{conexionesDirectas(p,m')}) \end{pmatrix} \right)$$

generadores

vacio : $\longrightarrow$ mapa

AgregarPos : posicion p $\times$ conj(posicion) ps $\times$ mapa m $\longrightarrow$ mapa $\{\neg(\text{p} \in \text{posiciones(m)}) \wedge (\text{ps} \subseteq \text{posiciones(m)}\}$

observadores básicos

posiciones : mapa $\longrightarrow$ conj(posicion)

conexionesDirectas : posicion p $\times$ mapa m $\longrightarrow$ conj(posicion) $\{\text{p} \in \text{posiciones(m)}\}$

otras operaciones

conexion? : posicion $p1$ $\times$ posicion $p2$ $\times$ mapa m $\longrightarrow$ bool $\{\text{p1} \in \text{posiciones(m)} \wedge \text{p2} \in \text{posiciones(m)}\}$

hayAlgunaConexion? : posicion p $\times$ conj(posiciones) c $\times$ mapa m $\longrightarrow$ bool $\{\text{p} \in \text{posiciones(m)} \wedge \text{c} \subseteq \text{posiciones(m)}\}$

distancia : posicion $\times$ posicion $\longrightarrow$ nat

axiomas $\forall\ p, p', p1, p2$: posicion, $\forall\ ps$: conj(posicion), $\forall\ m$: mapa

posiciones(vacio) $\equiv \emptyset$

posiciones(AgregarPos(p,ps,m)) $\equiv$ Ag(p, posiciones(m))

conexionesDirectas(p, AgregarPos(p', ps, m)) $\equiv$ **if** p = p' **then**
ps
else
if p $\in$ ps **then**
Ag(p', conexionesDirectas(p,m))
else
conexionesDirectas(p,m)
fi
fi

conexion?(p1, p2, AgregarPos(p', ps, m)) $\equiv$ **if** p1 = p' **then**
if vacio?(ps) **then**
false
else
p2 $\in$ ps $\vee$ hayAlgunaConexion?(p2, ps, m)
fi
else
if p2 = p' **then**
if vacio?(ps) **then**
false
else
p1 $\in$ ps $\vee$ hayAlgunaConexion?(p1, ps, m)
fi
else
conexion?(p1,p2,m) $\vee$ (hayAlgunaConexion?(p1, ps, m) $\wedge$
hayAlgunaConexion?(p2, ps, m))
fi
fi

hayAlgunaConexion?(p, ps, m) $\equiv$ **if** vacio?(ps) **then**
false
else
conexion?(p, dameUno(ps), m) $\vee$ hayAlgunaConexion?(p, sinUno(ps), m)
fi

distancia(p,p') $\equiv$ **if** π_1(p) $<$ π_1(p') **then**
distancia($\langle \pi_1$(p'), π_2(p)$\rangle$, $\langle \pi_1$(p), π_2(p')$\rangle$)
else
if π_2(p) $<$ π_2(p') **then**
distancia($\langle \pi_1$(p), π_2(p')$\rangle$, $\langle \pi_1$(p'), π_2(p)$\rangle$)
else
$\sqrt{((\pi_1(p) - \pi_1(p')) \times (\pi_1(p) - \pi_1(p'))) + ((\pi_2(p) - \pi_2(p')) \times (\pi_2(p) - \pi_2(p')))}$
fi
fi

Fin TAD

3. TAD Sistema

TAD SISTEMA

géneros sistema

exporta sistema, generadores, observadores, eliminado?, rareza, jugadoresConectados

usa NATEXTENDIDO,POKEMON, JUGADOR, MAPA, POSICION, BOOL, CONJUNTO(α), MULTICONJUNTOEXTENDIDO(α)

igualdad observacional

$$(\forall s, s' : \text{sistema}) \quad \left(s =_{\text{obs}} s' \iff \left(\begin{array}{l} \text{pokeCotizacion(s)} =_{\text{obs}} \text{pokeCotizacion(s')} \wedge \\ \text{mapa(s)} =_{\text{obs}} \text{mapa(s')} \wedge_{\text{L}} \\ (\forall\ \text{p} : \text{posiciones(mapa(s))})(\text{hayPokemon?(p,s)} =_{\text{obs}} \text{hayPokemon?(p,s')} \wedge (\text{hayPokemon?(p,s)} \Rightarrow_{\text{L}} \text{pokemonEnPos(p,s)} =_{\text{obs}} \text{pokemonEnPos(p,s')})) \wedge_{\text{L}} \\ (\text{jugadores(s)} =_{\text{obs}} \text{jugadores(s')} \wedge_{\text{L}} (\forall\ \text{j} : \text{jugadores(s)}) \\ (\text{sancionesJugador(j,s)} =_{\text{obs}} \text{sancionesJugador(j,s')} \wedge_{\text{L}} \\ \text{sancionesJugador(j,s)} < 5 \Rightarrow_{\text{L}} \\ (\text{conectado?(j,s)} =_{\text{obs}} \text{conectado?(j,s')} \wedge_{\text{L}} \\ \text{conectado?(j,s)} \Rightarrow_{\text{L}} \\ (\text{posicionJugador(j,s)} =_{\text{obs}} \text{posicionJugador(j,s')} \wedge_{\text{L}} \\ (\forall\ \text{p: posiciones(mapa(s))}) \\ (\text{movsLejosDePos(j,p,s)} =_{\text{obs}} \text{movsLejosDePos(j,p,s')}))))) \end{array} \right) \right)$$

generadores

crearSistema	: mapa m × nat n	⟶ sistema	{¬0?(n)}
registrarJugador	: sistema s × jugador j	⟶ sistema	{j ∉ jugadores(s)}
conectarJugador	: sistema s × jugador j × posicion p	⟶ sistema	{(j ∈ jugadores(s) ∧L ¬eliminado?(j,s) ∧L ¬conectado?(j,s)) ∧ p ∈ posiciones(mapa(s))}
desconectarJugador	: sistema s × jugador j	⟶ sistema	{j ∈ jugadoresConectados(jugadores(s),s)}
moverJugador	: sistema s × jugador j × posicion p	⟶ sistema	{j ∈ jugadoresConectados(jugadores(s),s) ∧ p ∈ posiciones(mapa(s))}
agregarPokemon	: sistema s × pokemon × posicion p	⟶ sistema	{p ∈ posiciones(mapa(s)) ∧L ¬pokemonsCerca?(p, posiciones(mapa(s)), s)}
cambiarCotizacion	: sistema s × nat n	⟶ sistema	{¬0?(n)}
pagarBoleta	: sistema s × jugador j × nat n	⟶ sistema	{j ∈ jugadores(s) ∧L ¬eliminado?(j,s) ∧L n ≤ valorPokemones(pokemonesCapturados(j), s)}

observadores básicos

mapa	: sistema	⟶ mapa	
jugadores	: sistema	⟶ conj(jugador)	
conectado?	: jugador j × sistema s	⟶ bool	{j ∈ jugadores(s) ∧L ¬eliminado?(j,s)}
posicionJugador	: jugador j × sistema s	⟶ posicion	{jugadoresConectados(jugadores(s),s)}
sancionesJugador	: jugador j × sistema s	⟶ nat	{j ∈ jugadores(s)}
pokemonsCapturados	: jugador j × sistema s	⟶ multiconj(pokemon)	{j ∈ jugadores(s) ∧L ¬eliminado?(j,s)}
hayPokemon?	: posicion p × sistema s	⟶ bool	{p ∈ posiciones(mapa(s))}
pokemonEnPos	: posicion p × sistema s	⟶ pokemon	{p ∈ posiciones(mapa(s)) ∧L hayPokemon?(p,s)}
movsLejosDePos	: jugador j × posicion p × sistema s	⟶ nat	{j ∈ jugadoresConectados(jugadores(s),s) ∧ p ∈ posiciones(mapa(s))}
pokeCotizacion	: sistema s	⟶ nat	

otras operaciones

eliminado?	: jugador j × sistema s	⟶ bool	{j ∈ jugadores(s)}

rareza	:	pokemon pk × sistema s	⟶ nat {¬0?(#todosLosPokemons(s))}
todosLosPokemons	:	sistema	⟶ multiconj(pokemon)
pokemonsEnArea	:	conj(posicion) ps × sistema s	⟶ multiconj(pokemon) {ps ⊆ posiciones(mapa(s))}
pokemonsCapturadosPorGrupo	:	conj(jugador) js × sistema s	⟶ multiconj(pokemon) {js ⊆ jugadores(s)}
capturaPokemon?	:	jugador j × jugador j' × posicion p × sistema s	⟶ bool {j ∈ jugadoresConectados(jugadores(s), s) ∧ j' ∈ jugadoresConectados(jugadores(s), s) ∧ p ∈ posiciones(mapa(s))}
pokemonACapturar	:	jugador j × sistema s	⟶ bool {j ∈ jugadoresConectados(jugadores(s), s) ∧ₗ enRangoDeAlgunPk?(j, posiciones(mapa(s)), s)}
jugadoresConectados	:	conj(jugador) js × sistema s	⟶ conj(jugador) {js ⊆ jugadores(s)}
enRangoDeCaptura?	:	jugador j × posicion p × sistema s	⟶ bool {j ∈ jugadoresConectados(jugadores(s), s) ∧ p ∈ posiciones(mapa(s)))}
enRangoDeAlgunPk?	:	jugador j × conj(posicion) ps × sistema s	⟶ bool {j ∈ jugadoresConectados(jugadores(s), s) ∧ ps ⊆ posiciones(mapa(s)))}
posCaptura	:	jugador j × sistema s	⟶ posicion {j ∈ jugadoresConectados(jugadores(s), s) ∧ₗ enRangoDeAlgunPk?(j, posiciones(mapa(s)), s)}
posCapturaAux	:	jugador j × conj(posicion) ps × sistema s	⟶ posicion {j ∈ jugadoresConectados(jugadores(s), s) ∧ ps ⊆ posiciones(mapa(s))) ∧ₗ enRangoDeAlgunPk?(j, ps, s)}
jugadoresCapturando	:	posicion p × sistema s	⟶ conj(jugador) {p ∈ posiciones(mapa(s)))}
jugadoresCapturandoAux	:	conj(jugador) js × posicion p × sistema s	⟶ conj(jugador) {js ⊆ jugadoresConectados(jugadores(s), s) ∧ p ∈ posiciones(mapa(s)))}
movsLejosDePosDespuesDeMov	:	jugador j × posicion p × jugador j' × posicion p' × sistema s	⟶ nat {j ∈ jugadoresConectados(jugadores(s), s) ∧ j' ∈ jugadoresConectados(jugadores(s), s) ∧ p ∈ posiciones(mapa(s)) ∧ p' ∈ posiciones(mapa(s))}
pokemonsCerca?	:	posicion p × conj(posicion) ps × sistema s	⟶ bool {p ∈ posiciones(mapa(s)) ∧ ps ⊆ posiciones(mapa(s))}
alguienCapturaEnPos?	:	conj(jugador) js × posicion p × jugador j × posicion p' × sistema s	⟶ bool {js ⊆ jugadoresConectados(jugadores(s), s) ∧ j ∈ jugadoresConectados(jugadores(s), s) ∧ p ∈ posiciones(mapa(s)) ∧ p' ∈ posiciones(mapa(s))}
valorPokemones	:	multiconj(pokemon) pks × sistema s	⟶ nat {¬0?(#todosLosPokemons(s))}
pokemonesADebitar	:	jugador j × nat n × sistema s	⟶ multiconj(pokemon) {j ∈ jugadores(s)}
filtrar	:	multiconj(conj(pokemon)) $cpks$ × nat n × sistema s	⟶ multiconj(conj(pokemon)) {¬0?(#todosLosPokemons(s))}
menores	:	multiconj(conj(pokemon)) $cpks$ × sistema s	⟶ multiconj(conj(pokemon)) {¬0?(#todosLosPokemons(s))}

axiomas ∀ m: mapa, ∀ s: sistema, ∀ j, j': jugador, ∀ p, p': posicion, ∀ pk: pokemon, ∀ n: nat, ∀ js: conj(jugador), ∀ ps: conj(posicion), ∀ pks: multiconj(pokemon), ∀ $cpks$: conj(multiconj(pokemon))

mapa(crearSistema(m, n)) ≡ m
mapa(registrarJugador(s,j)) ≡ mapa(s)
mapa(conectarJugador(s,j,p)) ≡ mapa(s)
mapa(desconectarJugador(s,j)) ≡ mapa(s)
mapa(moverJugador(s,j,p)) ≡ mapa(s)

mapa(agregarPokemon(s,pk,p)) ≡ mapa(s)
mapa(cambiarCotizacion(s,n)) ≡ mapa(s)
mapa(pagarBoleta(s,j,n)) ≡ mapa(s)
jugadores(crearSistema(m,n)) ≡ ∅
jugadores(registrarJugador(s,j)) ≡ Ag(j,jugadores(s))
jugadores(conectarJugador(s,j,p)) ≡ jugadores(s)
jugadores(desconectarJugador(s,j)) ≡ jugadores(s)
jugadores(moverJugador(s,j,p)) ≡ jugadores(s)
jugadores(agregarPokemon(s,pk,p)) ≡ jugadores(s)
jugadores(cambiarCotizacion(s,n)) ≡ jugadores(s)
jugadores(pagarBoleta(s,j,n)) ≡ jugadores(s)
jugadoresConectados(js, s) ≡ **if** vacio?(js) **then**
∅
else
if ¬eliminado?(dameUno(js), s) ∧$_L$ conectado?(dameUno(js), s) **then**
Ag(dameUno(js), ∅)
else
∅
fi jugadoresConectados(sinUno(js), s)
fi
conectado?(j, registrarJugador(s,j')) ≡ **if** j = j' **then** false **else** conectado?(j,s) **fi**
conectado?(j, conectarJugador(s,j',p)) ≡ **if** j = j' **then** true **else** conectado?(j,s) **fi**
conectado?(j, desconectarJugador(s,j')) ≡ **if** j = j' **then** false **else** conectado?(j,s) **fi**
conectado?(j, moverJugador(s,j',p)) ≡ conectado?(j,s)
conectado?(j, agregarPokemon(s,pk,p)) ≡ conectado?(j,s)
conectado?(j, cambiarCotizacion(s,n)) ≡ conectado?(j, s)
conectado?(j, pagarBoleta(s,j',n)) ≡ conectado?(j, s)
posicionJugador(j, registrarJugador(s,j')) ≡ posicionJugador(j,s)
posicionJugador(j, conectarJugador(s,j',p)) ≡ **if** j = j' **then** p **else** posicionJugador(j,s) **fi**
posicionJugador(j, desconectarJugador(s,j')) ≡ posicionJugador(j,s)
posicionJugador(j, moverJugador(s,j',p)) ≡ **if** j = j' **then** p **else** posicionJugador(j,s) **fi**
posicionJugador(j, agregarPokemon(s,pk,p)) ≡ posicionJugador(j,s)
posicionJugador(j, cambiarCotizacion(s,n)) ≡ posicionJugador(j, s)
posicionJugador(j, pagarBoleta(s,j',n)) ≡ posicionJugador(j, s)
pokemonsCapturados(j, registrarJugador(s,j')) ≡ **if** j = j' **then** ∅ **else** pokemonsCapturados(j,s) **fi**
pokemonsCapturados(j, conectarJugador(s,j',p)) ≡ pokemonsCapturados(j,s)
pokemonsCapturados(j, desconectarJugador(s,j')) ≡ pokemonsCapturados(j,s)
pokemonsCapturados(j, agregarPokemon(s,pk,p)) ≡ pokemonsCapturados(j,s)
pokemonsCapturados(j, cambiarCotizacion(s,n)) ≡ pokemonsCapturados(j, s)
pokemonsCapturados(j, pagarBoleta(s,j',n)) ≡ pokemonsCapturados(j, s) - **if** j = j' **then** pokemonesADebitar(j,n,s) **else** ∅ **fi**
pokemonsCapturados(j, moverJugador(s,j',p)) ≡ **if** conectado?(j, s) ∧$_L$ capturaPokemon?(j, j', p, s) **then**
Ag(pokemonACapturar(j, s)), ∅)
else
∅
fi ∪ pokemonsCapturados(j, s)
enRangoDeCaptura?(j, p ,s) ≡ hayPokemon?(p, s) ∧ distancia(posicionJugador(j, s), p) ≤ 2
enRangoDeAlgunPk?(j, ps, s) ≡ **if** vacio?(ps) **then**
false
else
enRangoDeCaptura?(j, dameUno(ps), s) ∨ enRangoDeAlgunPk?(j, sinUno(ps), s)
fi
posCaptura(j, s) ≡ posCapturaAux(j, posiciones(mapa(s)), s)
posCapturaAux(j, ps, s) ≡ **if** enRangoDeCaptura?(j, dameUno(ps), s) **then**
dameUno(ps)
else
posCapturaAux(j, sinUno(ps), s)
fi
pokemonACapturar(j, s) ≡ pokemonEnPos(posCaptura(j, s), s)

jugadoresCapturando(p, s) ≡ jugadoresCapturandoAux(jugadoresConectados(jugadores(s), s), p, s)
jugadoresCapturandoAux(js, p, s) ≡ **if** vacio?(js) **then**
∅
else
if enRangoDeCaptura?(dameUno(js), p, s) **then**
Ag(dameUno(js), ∅)
else
∅
fi ∪ jugadoresCapturando(sinUno(js), p, s)
fi
capturaPokemon?(j, j', p, s) ≡ enRangoDeAlgunPk?(j, posiciones(mapa(s)), s) $\wedge_{\text{L}}$
movsLejosDePosDespuesDeMov(j, posCaptura(j, s), j', p, s) = 10 ∧
¬∅?(jugadoresCapturando(posCaptura(j, s), s)) $\wedge_{\text{L}}$
j = dameUno(jugadoresCapturando(posCaptura(j, s), s))
movsLejosDePos(j, p, registrarJugador(s,j')) ≡ movsLejosDePos(j, p, s)
movsLejosDePos(j, p, conectarJugador(s,j',p')) ≡ **if** j = j' **then** 0 **else** movsLejosDePos(j, p, s) **fi**
movsLejosDePos(j, p, desconectarJugador(s,j')) ≡ movsLejosDePos(j, p, s)
movsLejosDePos(j, p, agregarPokemon(s,pk,p')) ≡ **if** p = p' **then** 0 **else** movsLejosDePos(j, p, s) **fi**
movsLejosDePos(j, p, moverJugador(s,j',p')) ≡ movsLejosDePosDespuesDeMov(j, p, j', p', s)
movsLejosDePos(j, p, cambiarCotizacion(s,n)) ≡ movsLejosDePos(j, p, s)
movsLejosDePos(j, p, pagarBoleta(s,j',n)) ≡ movsLejosDePos(j, p, s)
movsLejosDePosDespuesDeMov(j, p, j', p', s) ≡ **if** distancia(p, posicionJugador(j, s)) ≤ 2 **then**
if j = j' **then**
if distancia(p, p') ≤ 2 **then**
movsLejosDePos(j, p ,s)
else
0
fi
else
if distancia(p, p') ≤ 2 **then**
if distancia(p, posicionJugador(j', s)) ≤ 2 **then**
movsLejosDePos(j, p ,s)
else
0
fi
else
1 + movsLejosDePos(j, p ,s)
fi
fi
else
0
fi
sancionesJugador(j, registrarJugador(s,j')) ≡ **if** j = j' **then** 0 **else** sancionesJugador(j,s) **fi**
sancionesJugador(j, conectarJugador(s,j',p)) ≡ sancionesJugador(j,s)
sancionesJugador(j, desconectarJugador(s,j')) ≡ sancionesJugador(j,s)
sancionesJugador(j, moverJugador(s,j',p)) ≡ **if** j = j' $\wedge_{\text{L}}$ (10 ≤ distancia(p, posicionJugador(j,s)) ∨
(¬conexion?(p, posicionJugador(j,s), mapa(s)))) **then**
1
else
0
fi + sancionesJugador(j,s)
sancionesJugador(j, agregarPokemon(s,pk,p)) ≡ sancionesJugador(j,s)
sancionesJugador(j, cambiarCotizacion(s,n)) ≡ sancionesJugador(j, s)
sancionesJugador(j, pagarBoleta(s,j',n)) ≡ sancionesJugador(j, s)
hayPokemon?(p, crearSistema(m,n)) ≡ false
hayPokemon?(p, registrarJugador(s,j)) ≡ hayPokemon?(p,s)
hayPokemon?(p, conectarJugador(s,j,p')) ≡ hayPokemon?(p,s)
hayPokemon?(p, desconectarJugador(s,j)) ≡ hayPokemon?(p,s)
hayPokemon?(p, moverJugador(s,j,p')) ≡ hayPokemon?(p,s) ∧
¬alguienCapturaEnPos?(jugadoresConectados(jugadores(s), s), p,
j', p', s)

hayPokemon?(p, agregarPokemon(s,pk,p')) ≡ p = p' ∨ hayPokemon?(p,s)
hayPokemon?(p, cambiarCotizacion(s,n)) ≡ hayPokemon?(p, s)
hayPokemon?(p, pagarBoleta(s,j,n)) ≡ hayPokemon?(p, s)
alguienCapturaEnPos?(js, p, j, p', s) ≡ **if** vacio?(js) **then**
false
else
if capturaPokemon?(dameUno(js), j, p', s) **then**
p = posCaptura(dameUno(js), s)
else
alguienCapturaEnPos?(sinUno(js), p, j, p', s)
fi
fi
pokemonEnPos(p, registrarJugador(s,j)) ≡ pokemonEnPos(p,s)
pokemonEnPos(p, conectarJugador(s,j,p')) ≡ pokemonEnPos(p,s)
pokemonEnPos(p, desconectarJugador(s,j)) ≡ pokemonEnPos(p,s)
pokemonEnPos(p, moverJugador(s,j,p')) ≡ pokemonEnPos(p,s)
pokemonEnPos(p, agregarPokemon(s,pk,p')) ≡ **if** p = p' **then** pk **else** pokemonEnPos(p,s) **fi**
pokemonEnPos(p, cambiarCotizacion(s,n)) ≡ pokemonEnPos(p, s)
pokemonEnPos(p, pagarBoleta(s,j,n)) ≡ pokemonEnPos(p, s)
pokemonsCerca?(p, ps, s) ≡ ¬vacio?(ps) $\wedge_L$ ((distancia(p, dameUno(ps)) < 5 ∧ hayPokemon?(dameUno(ps), s)) ∨ pokemonsCerca?(p, sinUno(ps), s))
eliminado?(j,s) ≡ sancionesJugador(j,s) = 5
rareza(pk, s) ≡ 100 - ((100 × #(pk,todosLosPokemons(s))) ÷ #(todosLosPokemons(s)))
todosLosPokemons(s) ≡ pokemonsEnArea(posiciones(mapa(s)),s) ∪ pokemonsCapturadosPorGrupo(jugadores(s),s)
pokemonsEnArea(ps, s) ≡ **if** vacio?(ps) **then**
∅
else
if hayPokemon?(dameUno(ps), s) **then**
Ag(pokemonEnPos(dameUno(ps), s), ∅)
else
∅
fi ∪ pokemonsEnArea(sinUno(ps),s)
fi
pokemonsCapturadosPorGrupo(js, s) ≡ **if** vacio?(js) **then**
∅
else
if eliminado?(dameUno(js), s) **then**
∅
else
pokemonsCapturados(dameUno(js), s)
fi ∪ pokemonsCapturadosPorGrupo(sinUno(js), s)
fi
pokeCotizacion(crearSistema(m, n)) ≡ n
pokeCotizacion(registrarJugador(s,j)) ≡ pokeCotizacion(s)
pokeCotizacion(conectarJugador(s,j,p)) ≡ pokeCotizacion(s)
pokeCotizacion(desconectarJugador(s,j)) ≡ pokeCotizacion(s)
pokeCotizacion(moverJugador(s,j,p)) ≡ pokeCotizacion(s)
pokeCotizacion(agregarPokemon(s,pk,p)) ≡ pokeCotizacion(s)
pokeCotizacion(cambiarCotizacion(s, n)) ≡ n
pokeCotizacion(pagarBoleta(s, j, n)) ≡ pokeCotizacion(s)
valorPokemones(pks, s) ≡ **if** vacio?(pks) **then**
0
else
(rareza(dameUno(pks), s) × pokeCotizacion(s)) + valorPokemones(sinUno(pks), s)
fi
pokemonesADebitar(j,n,s) ≡ dameUno(menores(filtrar(partes(pokemonsCapturados(j, s)), n, s), s))

filtrar(cpks, n, s) $\equiv$ **if** vacio?(cpks) **then**
∅
else
if n $\leq$ valorPokemones(dameUno(cpks), s) **then** Ag(dameUno(cpks), ∅) **else** ∅ **fi**
$\cup$ filtrar(sinUno(cpks), n, s)
fi

menores(cpks, s) $\equiv$ **if** vacio?(cpks) **then**
∅
else
if vacio?(sinUno(cpks)) $\vee_{\text{L}}$ valorPokemones(dameUno(cpks), s) $\leq$ valorPokemones(dameUno(menores(sinUno(cpks), s)), s) **then**
Ag(dameUno(cpks), ∅)
else
∅
fi $\cup$ menores(sinUno(cpks), s)
fi

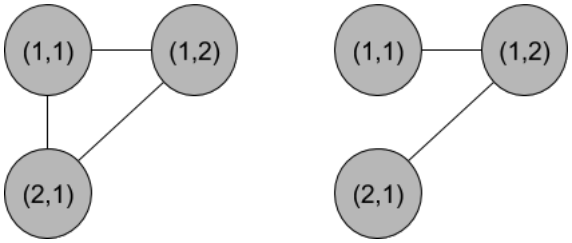
Fin TAD

4. Consideraciones

4.1. Consideraciones de diseño

- El TAD Pokemon es String porque asumimos que lo único que nos interesa saber es el tipo de pokemon. Asimismo, el TAD Jugador es Nat que se refiere a un ID único. El resto de los detalles relacionados (como sus posiciones, conexión del jugador, etc.) los maneja el TAD Sistema.
- Para el TAD Mapa el enunciado nos pide saber las posiciones válidas y como están conectadas entre sí, y para esto último supusimos que, además de saber si dos puntos están conectados, nos interesa saber si existe una conexión directa entre ellos (es decir, si hay un camino entre ambos sin otras posiciones en el medio)

  Por ejemplo, los siguientes casos se consideran distintos mapas (los puntos son posiciones y las líneas son conexiones directas):



  Cabe destacar que esto no afecta la lógica del juego, ya que los requisitos para movimientos válidos siguen siendo 1) una conexión (directa o no) y 2) distancia menor a 10, que se cumplen en ambos mapas. Por lo tanto, el movimiento de (1,1) a (2,1) y su inverso siempre son válidos.
- Cuando un jugador se registra, el mismo está desconectado y no tiene una posición inicial.
- Suponemos que un jugador puede estar en la misma posición que otros jugadores o un pokemon.
- Dado que el TAD Pokemon es String y no podemos diferenciar pokemons del mismo tipo, los `pokemonsCapturados` los representamos con un Multiconjunto de pokemons.
- La función `movsLejosDePos` tiene la logica de cuantos movimientos hubo fuera del rango de captura para avanzar la captura del pokemon que se ubica ahí. Asumimos que si 2 o más jugadores estan en el rango de captura de un pokemon determinado y uno de ellos sale del rango, este movimiento cuenta para los jugadores que permanecen en el rango.
- Extendimos los naturales para agregar la división y la raíz cuadrada. En ambos casos tomamos la parte entera y descartamos el resto.
- Extendimos el multiconjunto para agregarle la operación de partes, que genera un conjunto con todos los posibles subconjuntos del multiconjunto original.

4.2. Consideraciones de reentrega

- Las funciones que corresponden a TADs preexistentes fueron movidas a extensiones de los mismos.
- Para solucionar la dependencia circular infinita y el comportamiento automático con capturaPokemon, modularizamos algunas funciones y agregamos `movsLejosDePosDespuesDelMov`. Así, podemos reducir la instancia sin problemas pero mantener el contador de 10 que teníamos antes.
- El generador `crearSistema` ahora toma la cotización inicial del sistema como parámetro.
- No se especifica en la consigna si pagarBoleta requiere que el jugador en cuestión esté conectado. Esto se puede modificar agregando la restricción `j ∈ jugadoresConectados(jugadores(s), s)`.