UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2012

30 de Agosto de 2012

TPE De Hoteles y Pasajeros v1.0

Aclaraciones

- Para aprobar la totalidad del TP es necesario tener aprobado cada uno de sus módulos.
- **No** está permitido el uso de **acum** para la resolución.

La industria hotelera se ha caracterizado a los largo de los últimos años por no estar presente en ningún trabajo práctico de AED1.

Tenemos una posibilidad única de remediar la historia, especificando todo lo que se nos pide a continuación.

1. Tipos

`tipo` Fecha = $\mathbb{Z}$;
`tipo` DNI = $\mathbb{Z}$;
`tipo` Dinero = $\mathbb{Z}$;
`tipo` Cadena = String;
`tipo` CheckIn = (DNI, Fecha);
`tipo` CheckOut = (DNI, Fecha);
`tipo` TipoHabitacion = Simple, Doble, Triple, Cuadruple;
`tipo` Accesorio = Jacuzzi, LCD, PS3, DVD, Pelotero, Inflable;

2. Reserva

`tipo Reserva` {
  `observador documento` (r: Reserva) : DNI;
  `observador fechaDesde` (r: Reserva) : Fecha;
  `observador fechaHasta` (r: Reserva) : Fecha;
  `observador tipo` (r: Reserva) : TipoHabitacion;
  `observador confirmada` (r: Reserva) : Bool;

  `invariante NoAntesDeDespues` : $fechaHasta(r) > fechaDesde(r)$;
}

3. Habitación

`tipo Habitacion` {
  `observador numero` (h: Habitacion) : $\mathbb{Z}$;
  `observador tipo` (h: Habitacion) : TipoHabitacion;
  `observador accesorios` (h: Habitacion) : [Accesorio];

  `invariante sinAccesoriosRepetidos` : $sinRepetidos(accesorios(h))$;
  `invariante accesoriosOrdenada` : $ordenada(accesorios(h))$;
}

1. `problema pretensionesDePopStar` (as: [Accesorio], hs: [Habitacion]) = `result` : [Habitacion] A partir de una lista de accesorios as y una lista de habitaciones hs, se debe devolver una lista con aquellas habitaciones de hs que posean la mayor cantidad de accesorios de as. }

4. HotelBoutique

```
tipo HotelBoutique {
  observador nombre (h: Hotel) : Nombre ;
  observador cadena (h: Hotel) : Cadena ;
  observador huespedes (h: Hotel) : [DNI] ;
  observador habitaciones (h: Hotel) : [Habitacion] ;
  observador ingresos (h: Hotel) : [(CheckIn, Habitacion)] ;
  observador salidas (h: Hotel) : [CheckOut] ;
  observador reservas (h: Hotel) : [Reserva] ;
  observador tarifaHabitacionXDia (h: Hotel) : [(TipoHabitacion, Dinero)] ;
  observador precioAccesorio (h: Hotel) : [(Accesorio, Dinero)] ;
  invariante habitacionesValidas : (∀c ← ingresos(h))snd(c) ∈ habitaciones(h) ;
  invariante siEstanNoSeFueron : ∀d ← huespedes(h)|ingresosDe(h, d)| == |salidasDe(h, d)| + 1 ;
  invariante siSeVaEntro : ... ;
  invariante estanAlMenosUnDia : ... ;
  invariante noEntranDosVeces : ... ;
  invariante reservasValidas : (∀r ← reservas(h))existeUnaHabitacionDelTipo(h, tipo(r) ;
  invariante sinTarifasRepetidas : sinRepetidos([prm(t)|t ← tarifaHabitacionXDia(h)]) ;
  invariante sinPreciosRepetidos : sinRepetidos([prm(p)|p ← precioAccesorio(h)]) ;
  invariante tarifasPositivas : (∀t ← tarifaHabitacionXDia(h))snd(t) > 0 ;
  invariante preciosPositivos : (∀p ← precioAccesorio(h))snd(p) > 0 ;
  invariante noValeAcaparar : (∀r1, r2 ← reservas(h), tipo(r1) == tipo(r2) ∧ documento(r1) == documento(r2) ∧
    fechaDesde(r1) == fechaDesde(r2))r1 == r2 ;
}
```

2. `invariante` [siSeVaEntro]Que garantiza que para cada checkOut haya exactamente un checkin asociado (según el DNI en cuestión), con fecha anterior, sin otro checkout en el medio de ambos.
3. `invariante` [estanAlMenosUnDia]Que garantiza que entre el checkIn y el checkOut de cualquier huesped, debe pasar al menos un día.
4. `invariante` [noEntranDosVeces]Que garantiza que ningún huesped tenga dos checkIn el mismo día.
5. `problema sobreReservado` (h: Hotel, f: Fecha) = `result` : Bool Este problema, dados un hotel h y una fecha f, devolverá verdadero sí y sólo si la cantidad de plazas reservadas (calculada a partir de las capacidades de cada tipo de habitación reservada) es estrictamente superior que la capacidad total del hotel.
   La capacidad del hotel puede calcularse considerando cada una de sus habitaciones y su correspondiente tipo. }
6. `problema registrarHuesped` (h: Hotel, d: DNI, f:Fecha, a: Habitacion) Este problema modifica el hotel h registrando el checkin del huesped con dni d en la fecha f para la habitación a.
   Debe existir una reserva sin confirmar, asociada al DNI d cuya fecha sea f y el tipo de habitación coincida con el tipo de la habitación a.
   Una vez finalizado el registro, además de haber registrado a d como huesped (con su correspondiente checkin), se deberá marcar la reserva como confirmada. }
7. `problema desRegistrarHuesped` (h: Hotel, d: DNI, f:Fecha) Este problema modifica el hotel h registrando la salida del huesped d en la fecha f.
   Sólo se puede registrar la salida si en la fecha f el huesped se encuentra efectivamente registrado en el hotel. }
8. `problema huespedesConPalabra` (h: Hotel) = `result` : [DNI] Este problema devuelve una lista con aquellos DNI que hayan respetado sus reservas, cumpliendo tanto con la fecha de checkin (fecha desde) como con la fecha de checkout (fecha hasta). }
9. `problema calcularCuenta` (h: Hotel, i: CheckIn, o: CheckOut, hb: Habitacion) = `result` : Dinero Este problema devuelve el importe asociado a una estadia en el hotel h, en la habitación hb, determinada por el checkin i y el checkout o.
   Sólo se podrá calcular el importe si el checkout o se corresponde con el checkin i. Y si existe en el hotel h, tanto un ingreso en la habitación hb (con los datos de i) como una salida o.
   El importe a calcular dependerá del tipo de habitación, de los accesorios de la misma y de las correspondientes listas de tarifas y precios que posee el hotel. }
10. `problema reservasSolapadas` (h: Hotel, d: DNI) = `result` : Bool Devuelve verdadedo si la persona de documento d tiene reservas sin confirmar cuyas fechas se solapen de alguna manera. }

5. MinisterioDeTurismo

```
tipo MinisterioDeTurismo {
  observador secretarias (m: MinisterioDeTurismo) : [Provincia] ;
  observador registro (m: MinisterioDeTurismo, p: Provincia ) : [Hotel] ;
       requiere p ∈ secretarias(m) ;
  observador cadenasDeHoteles (m: MinisterioDeTurismo) : [[Hotel]] ;
  invariante sinHotelesRepetidas : (∀xs ← cadenasDeHoteles(m))sinRepetidos(xs) ;
  invariante sinCadenasRepetidas : sinRepetidos(xs) ;
  invariante sinProvinciasRepetidas : sinRepetidos(secretarias(m)) ;
  invariante cadenasBienFormadas : ... ;
  invariante sinNombresRepetidosEnCadenas : ... ;
  invariante hotelesConsistentes : ... ;
}
```

11. `invariante` [cadenasBienFormadas]Este invariante garantiza que todos los hoteles de cada elemento del observador cadenasDeHoteles tienen el mismo valor en el observador cadena.
12. `invariante` [sinNombresRepetidosEnCadenas]Este invariante garantiza que los hoteles de cada elemento del observador cadenasDeHoteles tienen distintos nombres.
13. `invariante` [hotelesConsistentes]Este invariante garantiza que los hoteles correspondientes a todas las cadenas (observador *cadenasDeHoteles*) son los mismos que se encuentran en las distintas provincias (observador *registro*)
14. `problema cadenasAmarretas` (m: MinisterioDeTurismo) = `result` : [Cadena] Este problema devuelve una lista con aquellas cadenas que poseen hoteles en la menor cantidad de provincias. }
15. `problema fusionAutorizada` (m: MinisterioDeTurismo, c1: Cadena, c2: Cadena) = `result` : $\mathbb{Z}$ Este problema modifica el ministerio m realizando una fusión entre las cadenas $c1$ y $c2$.
    Dicha fusión sólo se podrá realizar si ambas cadenas están registradas en el ministerio, si son compatibles (es decir, si la cadena resultante de la fusión continua siendo una cadena como tal) y si no compartían ninguna provincia entre sí (es decir, sus hoteles se encontraban en provincias diferentes).
    Luego de la fusión, todos los hoteles que correspondían a $c2$ deben pasar a corresponder a $c1$. }

6. Auxiliares

`aux dniCheckIn` (c: CheckIn) : DNI $= prm(c)$;

`aux dniCheckOut` (c: CheckOut) : DNI $= prm(c)$;

`aux fechaCheckIn` (c: CheckIn) : Fecha $= snd(c)$;

`aux fechaCheckOut` (c: CheckOut) : Fecha $= snd(c)$;

`aux ingresosDe` (h: Hotel, d: DNI) : [CheckIn] $= [x | x \leftarrow ingresos(h), dniCheckIn(prm(x)) == d]$;

`aux salidasDe` (h: Hotel, d: DNI) : [CheckOut] $= [x | x \leftarrow salidas(h), dniCheckOut(prm(x)) == d]$;

`aux existeUnaHabitacionDelTipo` (h: Hotel, t: TipoHabitacion) : Bool $= (\exists x \leftarrow habitaciones(h))tipo(x) == t$;

`aux capacidad` (h: Hotel) : $\mathbb{Z} = [\, cantidadHuespedes(tipo(h)) \mid h \leftarrow habitaciones(h) \,]$;

`aux cantidadHuespedes` (t: TipoHabitacion) : $\mathbb{Z}$ =
if $t == Simple$ then 1 else (if $t == Doble$ then 2 else (if $t == Triple$ then 3 else 4)) ;

`aux aplanar` (xss: [[T]]) : [T] $= [\, x \mid xs \leftarrow xss, x \leftarrow xs \,]$;

`aux ordenada` (l:[T]) : Bool $= (\forall i \leftarrow [0..|l| - 1))l_i \leq l_{i+1}$;

`aux sinRepetidos` (l: [T]) : Bool $= (\forall i, j \leftarrow [0..|l|), i \neq j)l_i \neq l_j$;

`aux sacarRepetidos` (l: [T]) : Bool $= (\forall i \leftarrow [0..|l|))l_i \notin l[i + 1..longitud(l) - 1)$;