Algoritmos y Estructuras de Datos

Guía Práctica 4
Especificación de TADs

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

Ultima actualización: 28/9/2023.
Los cambios respecto de la guía original están marcados en rojo.

Ejercicio 1. ★ Especificar TADs para las siguientes figuras geométricas. Tiene que contener las operaciones rotar, trasladar y escalar y una más propuestas por usted.

a) `Rectangulo` (2D)

b) `Esfera` (3D)

Ejercicio 2. Especificar TADs para las siguientes estructuras:

a) `Multiconjunto<T>`
También conocido como `multiset` o `bag`. Es igual a un conjunto pero cada elemento puede estar contenido múltiples veces, y la estructura lleva el registro de la cantidad de veces que se encuentra cada elemento.

b) `Multidict<K, V>`
Misma idea pero para diccionarios: Cada clave puede estar asociada con múltiples valores.

Ejercicio 3. Un `cache` es una estructura de acceso rápido (tradicionalmente almacenada en memoria) que sirve para guardar temporariamente resultados que son consultados con frecuencia. Tiene la misma interface que un diccionario: guarda valores asociados a claves. Los datos almacenados en un cache pueden *desaparecer* en cualquier momento, en función de diferentes criterios.

Especificar caches genéricos (con claves de tipo K y valores de tipo V) que respeten las siguientes políticas de borrado automático. En todos los casos puede asumir que existe una función `now()` que devuelve la hora actual (asuma que es de tipo real)

a) `TTL` o `time-to-live`:
El cache tiene asociado un máximo tiempo de vida para sus elementos. Los elementos se borran automáticamente cuando se alcanza el tiempo de vida (contando desde que fueron agregados por última vez).

b) `FIFO` o `first-in-first-out`:
El cache tiene una capacidad máxima (máximo número de claves). Si se alcanza esa capacidad máxima se borra automáticamente la clave que fue seteada por última vez hace más tiempo.

c) `LRU` o `last-recently-used`:
El cache tiene una capacidad máxima (máximo número de claves). Si se alcanza esa capacidad máxima se borra automáticamente la clave que fue accedida por última vez hace más tiempo. Si no fue accedida nunca, se considera el momento en que fue agregada.

Ejercicio 4. Cambiar la especificación de los siguientes TADs pero utilizando los observadores propuestos:

a) `Pila<T>` observado con diccionarios

b) `Diccionario<K, V>` observado con conjunto (de tuplas)

c) `Punto` observado con coordenadas polares

Ejercicio 5. Especifique tipos para un robot que realiza un camino a través de un plano de coordenadas cartesianas (enteras), es decir, tiene operaciones para ubicarse en un coordenada, avanzar hacia arriba, hacia abajo, hacia la derecha y hacia la izquierda, preguntar por la posición actual, saber cuántas veces pasó por una coordenada dada y saber cuál es la coordenada más a la derecha por dónde pasó. Indique observadores y precondición/postcondición para cada operación:

```
Coord es struct<x: int, y: int>

TAD Robot {
    proc arriba(inout r: Robot)
    proc abajo(inout r: Robot)
    proc izquierda(inout r: Robot)
    proc derecha(inout r: Robot)
    proc másDerecha(in r: Robot): int
    proc cuantasVecesPaso(in r: Robot, in c: Coord): int
}
```

Ejercicio 6. En este ejercicio vamos a analizar una solución para modelar el comportamiento del programa de televsión *Insoportables*. Este es un programa televisivo muy exitoso que sale al aire todas las noches; en él se debate acerca de las relaciones entre los personajes de la farándula (los "famosos"). Con el tiempo, distintos famosos se van incorporando al programa (y nunca dejan de pertenecer al mismo). A lo largo del programa, los famosos se pelean y se reconcilian entre si. Los productores nos encargaron el desarrollo de un sistema que permita saber en todo momento quiénes están peleados y quiénes no.

```
Famoso es string

TAD Insoportables {
    obs famosos: conj<Famoso>
    obs enemigos: dict<Famoso, conj<Famoso>>

    proc peleados(in p: Insoportables, in f1: Famoso, in f2: Famoso): bool

    proc nuevoFamoso(inout p: Insoportables, in f: Famoso)
    proc pelear(inout p: Insoportables, in f1: Famoso, in f2: Famoso)
    proc reconciliar(input p: Insoportables, in f1: Famoso, in f2: Famoso)
}
```

Parte 2

Los productores descubrieron que quienes miran el programa recuerdan a los famosos por la cantidad de veces que se pelearon a lo largo del mismo, independientemente de si se reconciliaron. Por ello, nos piden conocer cuál es la persona famosa que más peleas tiene acumuladas.

```
proc masPeleadorHistorico(in p: Insoportables): conj<Famoso>
```

Ejercicio 7. Especifique un tipo abstracto de datos `Stock` que permita las siguientes funcionalidades:

- Registrar un nuevo producto con su stock mínimo (un natural).
- Permitir que sea informado un producto sustituto para otro (ambos deben ya estar registrados). El sustituto se utilizará para completar ventas si el stock del producto original no fuera suficiente (eventualmente sustituyendolo por completo). Cada producto tiene a lo sumo un sustituto y a su vez es a lo sumo substituto de otro producto. No hay transitividad en la sustitución (si c es substituto de b y b lo es de a, nunca c cubrirá un faltante de a en una venta).
- Registrar las ventas realizadas de un producto con su respectiva cantidad por venta, sabiendo que la cantidad jamás puede exceder a la suma del stock disponible del producto y su sustituto.
- Registrar las compras de un producto, indicando la cantidad de elementos comprados.
- Devolver el conjunto de todos los productos con stock debajo del mínimo que no tengan sustituto, o bien tales que el total del stock del producto más el de su sustituto esté por debajo del mínimo.

Ejercicio 8. Especifique el juego de la oca.

Dos jugadores, *alternadamente*, se mueven por un camino de baldosas, numeradas éstas $0, 1, 2, \ldots$ avanzando en cada turno tantas baldosas como indique el dado tirado y realizando la acción que indica el tablero para la baldosa alcanzada. Van jugando de a uno por vez.

El tablero indica que ambos jugadores comienzan en la baldosa cero, el número de la baldosa en donde está la llegada (que debe ser distinto del de partida) y cómo debe moverse un jugador cuando llega a una determinada baldosa: cuántas baldosas avanzar o retroceder, o no hacer nada (en los demás casos). La acción correspondiente a una baldosa sólo se toma

si se llega a esa avanzando con el dado, no si se llega mediante la acción de otra baldosa. Gana el primer jugador que alcanza la llegada. Proveer también funciones para saber qué indicó el dado en cada jugada para cada uno de los jugadores, y para saber si un jugador pasó por una baldosa determinada.

Pista 1: definir al menos 2 tipos; uno para el tablero y otro para el juego de la oca propiamente dicho, que use el tablero.

Pista 2: asuma que el valor del dado está dado como parámetro.

Ejercicio 9. Un banco posee dos cajas y dos colas. La caja A, atiende sólo gente *bien*, mientras que la caja B atiende a los *proles*, siempre y cuando no haya gente en la cola *bien*, que tiene prioridad en esa caja. Por razones de seguridad las personas deben identificarse dando su número de DNI al entrar al banco, momento en el cuál se ponen en la cola que le corresponde. Pueden cansarse de esperar y retirarse antes de ser atendidas. Cuando los cajeros están libres gritan "pase el que sigue", momento en el que atienden al próximo de las colas según el criterio de cada caja, que se retira sin dilaciones al terminar su transacción. Se pide modelar el TAD `Banco`.

Ejercicio 10. Con el fin de garantizar que dos alumnos no se encuentren en el baño mientras están realizando un parcial, se decidió implementar la siguiente política. Cuando un alumno solicita ir al baño, este puede concurrir al mismo si ningún otro alumno está en el baño. En caso contrario, éste esperará cerca de la puerta. Un alumno que está esperando cerca de la puerta puede decidir volver a sentarse en cualquier momento. Cuando un alumno regresa del baño, automáticamente va el primero de los alumnos que está esperando cerca de la puerta. Con el fin de no hacer perder mucho tiempo a los alumnos esperando en la cola.

Por otra parte, no se permite a un alumno ir mas de 3 veces al baño durante la realización del parcial.

Modelar mediante TADs el problema descripto anteriormente.

Ejercicio 11. *Técnicos a Domicilio* (o simplemente "TaD"), es una empresa que provee servicio técnico para problemas de electricidad en hogares y empresas. TaD cuenta con un grupo de técnicos altamente capacitados para atender la demanda de sus clientes y tiene una estrategia de trabajo algo particular. Cuando alguien solicita un técnico, la central de TaD verifica si alguno de sus técnicos se encuentra en la empresa y de ser así envía inmediatamente un técnico al domicilio de la persona. En caso de no haber técnicos disponibles en ese momento (i.e., todos se encuentran atendiendo algún pedido), el pedido queda *pendiente de asignación* a la espera de que algún técnico se desocupe.

Por otro lado, cuando un técnico termina de resolver un problema, y antes de retirarse de ese domicilio, el técnico avisa por radio a la central que quedó disponible para otro trabajo. Si existiesen en ese momento pedidos *pendientes de asignación*, la central le asigna al técnico el domicilio que está esperando hace más tiempo y el técnico se dirige automáticamente hacia allí. Por el contrario, de no haber trabajos pendientes, el técnico regresa a la central y queda disponible para futuros trabajos.

Modelar con un TAD la empresa *Técnicos a Domicilio*.