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

Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2016

13 de mayo de 2016

TPI - Agricultura con drones

1. Introducción

En el TPE, hemos especificado el comportamiento de una serie de problemas relacionados con el sistema CropFieldRobotics. Nuestro próximo paso es obtener una implementación en un lenguaje imperativo, para lo cual la especificación original ha sido *especialmente* adaptada teniendo en cuenta los mecanismos de abtracción (el uso de clases) y características propias del paradigma imperativo. Por eso, es importante utilizar esta especificación concebida para el TPI que se entrega como parte del enunciado, y no la solución a la que hayan llegado ustedes del TPE.

El objetivo de este trabajo es programar en C++ la especificación dada para obtener una primera versión estable (donde se han removido todas las fallas importantes y su uso esté disponible para los usuarios) del sistema CropFieldRobotics. El diseño de clases les será provisto y deben respetarlo. En la página de la materia estarán disponibles los *headers* (archivos .h) de las clases que deben implementar. Cada problema de cada tipo de la especificación dada se tiene que implementar como un método público de una clase en C++. Pueden agregar los métodos auxiliares que necesiten que deberán estar en la parte privada de la clase correspondiente. No se permite modificar la parte pública de las clases.

Podrán utilizar, además, el tipo abstracto `vector`, presente en la STL (*Standard Template Library*) de C++. Para hacer uso de este contenedor, sólo deben incluir la biblioteca `vector`.

2. Implementación de tipos

Los tipos `Campo`, `Drone` y `Sistema` tienen dentro de su clase métodos públicos que son similares a sus observadores y por lo tanto es sencillo comprender su equivalencia. Dentro de la clase `Campo` tenemos un nuevo tipo llamado `Grilla` que se trata básicamente de un vector de vectores para almacenar una matriz de `Parcelas` en el atributo `_grilla`. La clase `Drone` tiene un nuevo tipo llamado `Secuencia` que es un renombre de vector para almacenar la trayectoria del vuelo realizado en el atributo `_trayectoria`. La clase `Sistema` posee un atributo `_enjambre` que almacena el conjunto de drones en una `Secuencia`. Además, cada clase tiene declarado el `operator==` para definir la igualdad entre instancias de la clase.

3. Entrada/Salida

Todas las clases del proyecto tienen tres métodos relacionados con entrada/salida:

`mostrar` : que se encarga de mostrar todo el contenido de la instancia de la clase en el flujo de salida indicado. El formato es a gusto del consumidor, pero se espera que sea algo más o menos informativo y legible para el usuario.

`guardar` : que se encarga de escribir en un archivo de texto el contenido de una instancia de la clase en un formato predeterminado que debe ser decodificable por el método **cargar** que se detalla a continuación. Esta acción de codificar la información de un objeto en un medio de almacenamiento se como *serialización* y es un mecanismo ampliamente usado para transportar objetos a través de una red o para hacer persistente un objeto en un archivo o base de datos.

`cargar` : que se encarga de leer e interpretar el archivo de texto generado por el método anterior, modificando el valor del parámetro implícito (this) para que su contenido coincida con aquel almacenado en el archivo.

- `void mostrar(std::ostream& ) const;`
- `void guardar(std::ostream& ) const;`
- `void cargar(std::istream& );`

El detalle del formato que se debe usar en cada clase para guardar y cargar su contenido se indica a continuación. Tener en cuenta que los números se deben guardar como texto. Es decir, si escriben a un archivo y después lo miran con un editor de texto, donde escribieron un número 65 deben ver escrito 65 y no una letra `A`.

Campo: Se debe guardar entre llaves primero una C, indicando que es un campo, luego su dimensión entre corchetes separando por una coma el ancho y el largo, y por último la grilla de parcelas se representa como vector de vectores, cada vector agrupado entre corchetes y entre comas cada elemento de un vector. Por ejemplo:

```
{ C [3,3] [[Cultivo,Cultivo,Granero],[Cultivo,Casa,Cultivo],[Cultivo,Cultivo,Cultivo]]}
```

Drone: Se debe guardar entre llaves primero una D, indicando que es un drone, luego su id, la carga de su batería, un vector de posiciones que representa el vuelo realizado, otro vector con los productos químicos disponibles, un booleano que nos dice si se encuentra en vuelo y por último la posición en la que se encuentra el Drone. Por ejemplo:

```
{ D 12 83 [[1,2],[1,1],[1,0],[2,0]] [PlaguicidaBajoConsumo,Herbicida,Fertilizante] true [2,0]}
```

Sistema: Se debe guardar entre llaves primero una S, indicando que es un sistema, luego el campo, el enjambre de drones como un vector separado por comas, y por último la grilla de estados del cultivo como un vector de vectores. Por ejemplo:

```
{ S
{ C [3,3] [[Cultivo,Cultivo,Granero],[Cultivo,Casa,Cultivo],[Cultivo,Cultivo,Cultivo]]}
[{ D 12 83 [[1,2],[1,1],[1,0],[2,0]] [PlaguicidaBajoConsumo,Herbicida,Fertilizante] true [2,0]},
{ D 15 46 [[0,1],[1,1],[2,1],[2,2]] [HerbicidaLargoAlcance,Fertilizante,Plaguicida] true [2,2]}]
[[NoSensado,EnCrecimiento,NoSensado],[ConMaleza,NoSensado,ConPlaga],
[EnCrecimiento,ListoParaCosechar,ConPlaga]]
}
```

Importante: Los saltos de línea han sido dejado por cuestiones de visualización. En el archivo real no deben incorporarse.

4. Demostraciones

Además de realizar la implementación, la empresa GreenSoft que nos convocó para el desarrollo del sistema CropFieldRobotics, quiere estar segura que el software que se les va a entregar va a funcionar correctamente. Por suerte (para ustedes) no piden la demostración formal de toda la implementación, pero sí saber que son capaces de hacer la demostración en caso de ser necesario. Para esto, les pide que escriban Pc, Qc, I, B, $cota$ y fv de cada ciclo que aparece en la implementación de los problemas:

- vueloEscalerado
- listoParaCosechar

No deben realizar la demostración de correctitud entera para estos algoritmos, pero sí deben demostrar los siguientes puntos del Teorema del Invariante para cada uno de los ciclos involucrados:

- $Pc \rightarrow I$
- $(I \wedge \neg B) \rightarrow Qc$
- $(I \wedge fv < cota) \rightarrow \neg B$

Importante: Pueden implementar estos problemas utilizando las funciones auxiliares que deseen, pero cada una debe tener a lo sumo un sólo ciclo. Además, deben especificar las funciones auxiliares implementadas para este punto.

5. Forma de entrega

La entrega de este trabajo consiste en un informe y el código de implementación. Respecto del código, deben entregar un paquete (archivo comprimido) con todos los archivos fuentes del trabajo de manera que el corrector pueda compilar y ejecutar el programa `correrTests`. También deben entregar los casos de test extras que hayan realizado. Respecto del informe, deben presentar todas las decisiones de implementación que hayan tomado para el desarrollo del trabajo, junto con las demostraciones requeridas en el punto anterior. El código como el informe deben enviarlos por correo electrónico a la dirección `tpalgo1@gmail.com` idicando el número de grupo en el asunto y detallando los datos de los integrantes en el cuerpo. Además, deben entregar el informe en formato impreso siguiendo las mismas pautas que para el primer trabajo práctico. Recuerden que no puede diferir la versión digital y la impresa del informe. Tienen tiempo hasta el viernes 10 de junio a las 17:59 hs para realizar la entrega de este trabajo.