Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2022.

PROYECTO 2: CHAPIN WARRIORS, S. A.

202001570– Nataly Saraí Guzmán Duarte

Resumen

En este ensayo se presenta la creación y desarrollo técnico del proyecto No. 2 del curso de Introducción a la Computación 2, del primer semestre del año 2022. Este proyecto se realizó con una interfaz gráfica, haciendo uso de tipos de datos abstractos, memoria dinámica, objetos y módulos para leer archivos XML. El proyecto se llevó a cabo en el lenguaje de programación Python.

El programa desarrollado proporciona al usuario una gestión de datos de forma dinámica, para lo cual los datos se almacenan en seis tipos de listas, las cuales son listas simples, lo cual hacen un uso adecuado de TDA, permitiendo así la posibilidad de realizar un conjunto de funciones indicadas en el documento de requerimiento.

El conjunto de datos obtenido en las listas se puede visualizar a través de los reportes que se crean en el momento de ejecución como son los PDF de las gráficas de los pisos según la ciudad que se requiere, y el pdf con los datos y grafica de la ciudad donde se quiere completar la misión.

Palabras clave

Programación orientada a objetos, listas, TDA, memoria dinámica, clase.

Abstract

This essay presents the creation and technical development of project No. 1 of the Introduction to Computing 2 course, from the first semester of the year 20222. This project was carried out with an interface in console mode, using abstract data types, dynamic memory, objects and modules to read XML files. The project was carried out in the Python programming language.

The developed program provides the user with dynamic data management, for which the data is stored in six types of lists, which are simple lists, which make proper use of TDA, thus allowing the possibility of making a set functions indicated in the requirement document..

The set of data obtained in the lists can be viewed through the reports that are created at the time of execution, such as the PDF of the graphs of the flats according to the city that is required, and the pdf with the data and graph of the city where you want to complete the mission..

Keywords

Object Oriented programming, lists, TDA, dynamic memory, class.

Introducción

La construcción del proyecto se fundamenta en el uso de la programación orientado a objetos, este paradigma nos permitió generar tipos de datos abstractos fundamentados en la información que el documento de requerimiento nos mostraba, así como todos los atributos que correspondían en los archivos a procesar. Para el procesamiento de los archivos se utilizó una librería cuyo propósito se basa en el procesamiento de archivo con extensión XML, esta librería permitió la lectura de datos así también como la construcción de archivos de salida con la extensión anteriormente mencionada.

El desarrollo del proyecto consiste en aplicar funcionalidades en los datos proporcionados en los archivos, la función consiste en encontrar caminos para realizar dos tipos de misiones, la misión de rescate que consiste en rescatar unidades civiles sin pasar por celdas que contengan unidades militares; y la misión de extracción de recursos consistiendo en encontrar el camino para algún recurso y verificando la capacidad de combate para que esta no disminuya según la celda en la que pase. El proyecto proporciona al estudiante poner en práctica sus conocimientos y su lógica para encontrar la solución del problema propuesto en el documento de requerimiento.

Desarrollo del tema

El problema propuesto en el documento de especificaciones y requerimientos, especifica que se requiere un programa para la empresa Chapín Warriors, S. A. el cual consiste que por medio de un archivo XML se pueden ingresar ciudades y robots para los cuales se puede realizar dos tipos de misiones, la misión de rescate y la misión de extracción de recursos, ambas tienen sus restricciones, como contar con puntos de entrada, unidades civiles, tipos de robots según su misión y capacidad de combate de estos, buscando así de manera el camino más conveniente para llegar al recurso o unidad civil según se requiera.

Para encontrar la solución al problema anteriormente mencionado, debemos saber que el programa a desarrollar debía contar con las siguientes especificaciones: los datos necesarios para las ciudades seria proporcionados en archivos XML, para procesar la información de las ciudades y robots se tenía que poner en práctica la memoria dinámica haciendo uso de listas construidas por medio de tipos abstractos, y los reportes generados por el programa seria por medio de grafos, haciendo uso de Graphviz. A continuación, explicaremos a detalle el desarrollo del proyecto. Para proponer la solución del problema se aplicó un término muy conocido en el desarrollo de software y es el Ciclo de Vida del Software el cual implementa las siguientes etapas o fases:

a. Análisis
b. Diseño
c. Codificación
d. Prueba y corrección

A continuación, se explicará cada una de las etapas.

Etapa de Análisis:
En esta etapa se consideró cada una de las especificaciones propuestas en el documento de especificaciones y requerimientos y se logró determinar que lo más eficiente para solución sería utilizar el paradigma de programación orientada a objetos, ya que este nos permite el manejo de clases y objetos. Las clases son una plantilla abstracta de los individuos que interactúan en el problema. Tomando como base los individuos que interactúan en el

programa, se logró determinar que se necesita una clase llamada Ciudad, la cual tendría los atributos como nombre, filas, columnas, listado de fila y listado de celdas; también se necesita una clase llamada Robot, la cual tendría los atributos como tipo, capacidad y nombre, una clase llamada celda que tiene los atributos fila, columna y carácter y finalmente una lista de Unidades Militares la cuales tendrá una clase llamada UnidadMilitar que tendrá como atributos fila, columna y numero. Estas clases se convirtieron en los tipos de datos utilizados para generar las listas dinámicas donde almacenaran los datos.

Las clases lista simples y matriz dispersa eran indispensables ya que estas nos proporcionarían el manejo de datos, la lista simple fue la encargada de almacenar los datos de ciudades y robots y unidades militares, dentro de cada nodo fila se necesitó una lista doble donde se almaceno las celdas.

Tabla I. *Lista de clases.*

CATEGORÍA	CATEGORÍA
CIUDAD	TDA
FILA	TDA
CELDA	TDA
UNIDAD MILITAR	TDA
ROBOT	TDA
CABECERA	TDA
LISTA CIUDAD	LISTA DATOS
LISTA FILA	LISTA DATOS
LISTA CELDAS	LISTA DATOS
LISTA ROBOTS	LISTA DATOS
LISTA UNIDAD MILITAR	LISTA DATOS

Fuente: elaboración propia.

Luego de tener los datos se procedería aplicar los algoritmos para encontrar los caminos según cada misión, el de rescate que consiste en llegar a la unidad civil y el de misión de extracción que consistirá en llegar al recurso.

Etapa de Diseño:

Obteniendo la información de la etapa de análisis, se procedió a crear el diseño de las clases propuestas en la Tabla I y la relación entre ellas. Para esta etapa se utilizó el diagrama de clases que nos proporciona UML. Para el diagrama utilizamos relaciones de asociación para relacionar las clases TDA con las listas de datos, el diagrama también cuenta con la multiplicidad en las relaciones entre las clases.

El diagrama de clases ya finalizado es el siguiente:

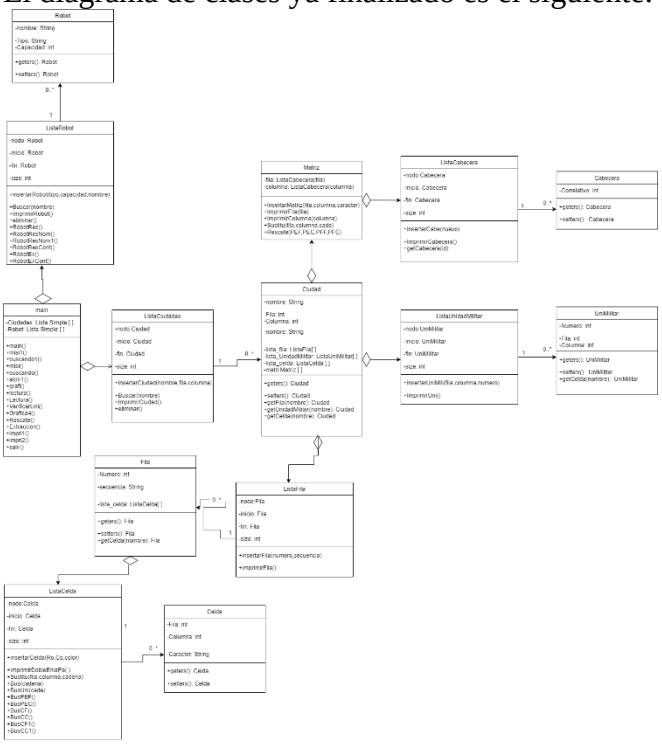


Figura 1. Diagrama de clases UML.
Fuente: elaboración propia.

Etapa de Codificación:

Para esta etapa del proyecto utilizamos los siguientes componentes:

a. Sistema operativo: Windows 10
b. Editor de código: Visual Studio Code
c. Lenguaje de Programación: Python 3.
d. Librería para XML: Element Tree
e. Creación Reportes: Graphviz

f. Librería para interfaz gráfica: PyQt5

Debemos saber que la codificación va de la mano con el diagrama mostrado en la figura 1, se explicara la funcionalidad de cada una de las clases y los métodos utilizados.

Se comenzó codificando los dos TDA que serán la base del proyecto, la clase ciudad cuenta con distintos métodos empezando por un constructor quien se encarga de inicializar los atributos de la clase, también cuenta con los métodos getCiudad, getFila y getCelda ambos métodos retorna una ciudad y un listado de Filas y Celdas respectivamente. La clase fila al igual que la clase anterior cuenta con un constructor y con sus respectivos getters y setteres, así como también la clase celda que cuenta con lo mismo.

Las siguientes clases a codificar fueron las listas simples y dobles ya que estas utilizan como nodos los tipos abstractos. Estas clases fueron implementadas haciendo uso de algoritmos de listas dinámicas cuentan ambas con sus métodos insertar e imprimir y cuenta también con un método que retorne un nodo del tipo TDA del cual la lista se basa, también cuenta con un método de obtención de un nodo en particular.

Ya teniendo la base fundamental del proyecto se procedió a codificar la parte visual, construyendo en la clase main, el cual contiene una clase que visualiza la parte grafica que se creó con PyQt5 las opciones que el usuario podrá realizar como ejemplo está la figura 3 de la sección de Anexos.

La lectura de archivos XML se implementó haciendo uso de la librería Element Tree la cual nos proporciona parsear el contenido del archivo a un objeto el cual podemos recorrer para obtener los datos y estos fueron almacenados en las listas dinámicas creadas, un ejemplo de los archivos XML de entrada esta la figura 4 de la sección de Anexos.

Obteniendo los datos ya almacenados se procedió a codificar y aplicar el algoritmo sobre la información de las ciudad y las posiciones de los elementos que contenía dicha ciudad para luego encontrar los caminos de las misiones según lo deseen.

La salida de datos XML se utilizó al igual que la lectura el implementar la librería Element Tree la cual a base de recorrer nuestra lista se obtuvo los archivos XML como ejemplo esta la figura 5 de la sección de Anexos.

Para generar los reportes haciendo uso de Graphviz se implementó un método que generaba un archivo con extensión dot (ejemplo en la figura 6 de la sección de Anexos) y que al procesarlo produce el reporte en PDF para que sea útil para el usuario.

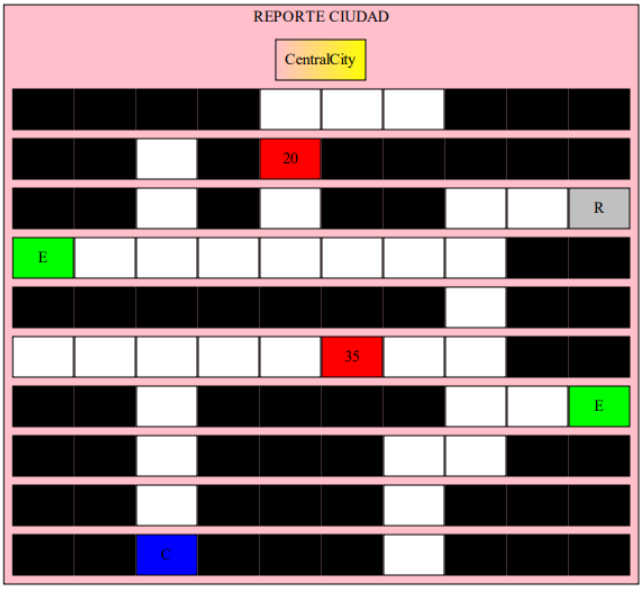


Figura 2. Reporte de Ejemplo.
Fuente: elaboración propia.

La codificación de este proyecto fue muy laboriosa por el manejo de listas dinámicas construidas por TDA, se puso en práctica los conceptos de programación orientada a objetos, TDA y uso de lenguaje Python para aumentar los conocimientos en este lenguaje.

Etapa de Prueba y Corrección:
En esta etapa se realizaron pruebas de cada una de las funciones en especial las del manejo de datos, la creación de reportes con Graphviz, cuya imagen está en la figura 4 de la sección de Anexos, al finalizar las pruebas se realizaron las correcciones necesarias para cumplir con las especificaciones.

Conclusiones

Los TDA tiene una gran facilidad para ser implementados en distintas estructuras de datos, y aun así proveer la misma funcionalidad, esto hace que los tipos abstractos sea muy versátiles.
La memoria dinámica tiene una gran ventaja, la cual beneficia tanto al programador como al equipo de cómputo, ya que gracias a que esta se crea y se destruye en tiempo de ejecución, la manipulación de datos es más sencilla y la computadora no desperdicia fragmentos de memoria.
La implementación de listas enlazadas tiene un gran beneficio respeto a las listas convencionales es que el orden de los elementos puede ser diferente al orden de almacenamiento en la memoria ya que estos en si son fragmentos de memoria que usan apuntadores para saber cuál es el fragmento siguiente.

Referencias bibliográficas

A. V. Aho, J. E. Hopcroft, (1998). *Estructura de datos y algoritmos.* Addison-Wesley Publishing Company, Inc.

Anexos

El menú del proyecto proporción al usuario siete opciones la primera permite el ingreso de la ruta de archivo que será procesado, la segunda permite mostrar los ciudades cargados, la tercera permite mostrar los robots cargados, la cuarta genera un archivo pdf con la gráfica de una ciudad en específico, la quinta realiza la misión de rescate, la sexta realiza la misión de extracción de recurso y la septima termina la ejecución del programa.

Figura 3. Menú del proyecto.
Fuente: elaboración propia

Los archivos de entrada son proporcionados por el usuario, tienen extensión XML y tienen la siguiente estructura.

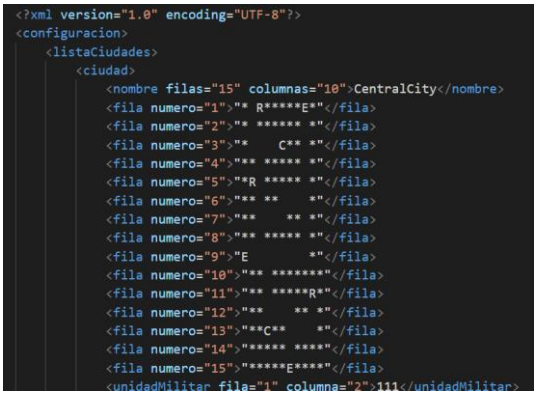


Figura 4. Ejemplo de archivos de entrada.
Fuente: elaboración propia

Los archivos de salida son generados por el programa, tienen extensión pdf y tienen la siguiente estructura.

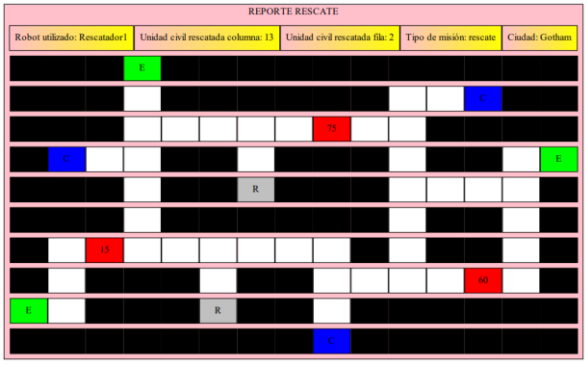
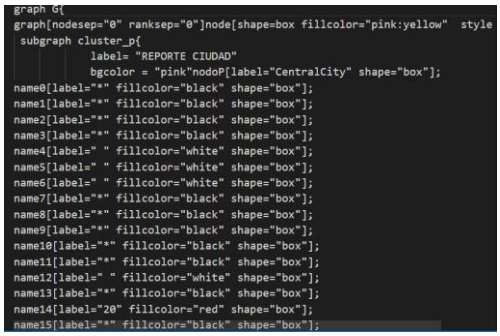


Figura 5. Ejemplo de archivos de salida.
Fuente: elaboración propia

La opción cinco, cuya función es generar el reporte gráfico, nos crea un archivo tipo dot que al procesarlo nos deja archivos PDF.



```
graph G{
graph[nodesep="0" ranksep="0"]node[shape=box fillcolor="pink:yellow"  style
 subgraph cluster_p{
            label= "REPORTE CIUDAD"
            bgcolor = "pink"nodoP[label="CentralCity" shape="box"];
name0[label="*" fillcolor="black" shape="box"];
name1[label="*" fillcolor="black" shape="box"];
name2[label="*" fillcolor="black" shape="box"];
name3[label="*" fillcolor="black" shape="box"];
name4[label=" " fillcolor="white" shape="box"];
name5[label=" " fillcolor="white" shape="box"];
name6[label=" " fillcolor="white" shape="box"];
name7[label="*" fillcolor="black" shape="box"];
name8[label="*" fillcolor="black" shape="box"];
name9[label="*" fillcolor="black" shape="box"];
name10[label="*" fillcolor="black" shape="box"];
name11[label="*" fillcolor="black" shape="box"];
name12[label=" " fillcolor="white" shape="box"];
name13[label="*" fillcolor="black" shape="box"];
name14[label="20" fillcolor="red" shape="box"];
name15[label="*" fillcolor="black" shape="box"];
```

Figura 6. Ejemplo de archivos dot de grafos.
Fuente: elaboración propia.