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 2024

PROYECTO 2: SOLUCIÓN DE LABERINTOS.

201213421 – Jorge Estuardo Pumay Soy

Resumen

El proyecto se desarrolló en lenguaje Python, utilizando el editor de texto Visual Studio Code en su versión para Windows de 64 bits. Para el almacenamiento de datos se utilizó listas enlazadas simples el cual con una clase Nodo, que contenía un objeto y un apuntador, se insertaron en las listas enlazadas. Lo anterior se modeló a través del paradigma de programación orientada a objetos (POO) el cual sirvió para modelar las clases como: “Maqueta”, “Entrada”, “Objetivo”, “Estructura” etc. Clases que posteriormente se instanciaron a través de la información proporcionada por un archivo XML, el programa debía tener la capacidad de ser incremental es decir ingresar y almacenar cuantos archivos se quisieran. Para la resolución del problema principal, el cual era, encontrar una lista de objetivos en un laberinto de “n” filas por “m” columnas. Se utilizó un algoritmo recursivo el cual exploraba todas las casillas del laberinto en busca de los objetivos, y en caso de no controlarlo, enseñaba un mensaje en pantalla.

Palabras clave

Listas enlazadas, programación orientada a objetos, recursividad.

Abstract

The project was developed in Python language, using the Visual Studio Code text editor in its 64-bit Windows version. Simple linked lists were used for data storage, in which a Node class, containing an object and a pointer, was inserted into the linked lists. This was modeled through the object-oriented programming (OOP) paradigm, which served to model classes such as "Maqueta", "Entrada", "Objetivo", "Estructura", etc. Classes that were later instantiated through information provided by an XML file; the program was required to have the ability to be incremental, meaning it could input and store as many files as desired. To solve the main problem, which was to find a list of objectives in a maze of "n" rows by "m" columns, a recursive algorithm was used to explore all the squares of the maze in search of the objectives, and in case of failure, it displayed a message on the screen.

Keywords

Linked lists, Object-oriented programming, recursion.

Introducción

Python es un lenguaje de alto nivel con una sintaxis accesible, ideal para implementar soluciones de todo tipo. Las listas nativas de Python son simples de implementar, pero para los objetivos del proyecto se busca aprender sobre estructuras de datos específicamente listas enlazadas, por lo cual en el proyecto no se implementaron dichas listas nativas de Python y en su lugar se programaron Listas enlazadas simples.

Desarrollo del tema

El programa solicita al usuario, por medio de una interfaz grafica utilizando la biblioteca Tkinter de Python, un archivo con extensión XML que contiene la información para crear los objetos “Maqueta” que posteriormente serán graficadas con la Herramienta Graphviz.

A. Biblioteca Tkinter

Tkinter es una biblioteca de Python que proporciona una interfaz gráfica de usuario (GUI) para aplicaciones. Es parte de la biblioteca estándar de Python y está ampliamente utilizada debido a su simplicidad y facilidad de uso. Tkinter permite crear ventanas, botones, etiquetas, cuadros de texto y otros elementos de interfaz de usuario de manera sencilla y rápida.

Una de las principales ventajas de Tkinter es su compatibilidad multiplataforma. Funciona en sistemas operativos como Windows, macOS y Linux, lo que permite crear aplicaciones GUI que sean portables entre diferentes plataformas sin necesidad de modificar el código. Esto hace que Tkinter sea una opción popular para desarrolladores que buscan crear aplicaciones que funcionen en múltiples sistemas operativos.

Tkinter se basa en el kit de herramientas gráficas Tk, que proporciona una amplia gama de widgets y herramientas para la creación de interfaces gráficas. Se puede personalizar la apariencia y el comportamiento de los elementos de la interfaz de usuario utilizando métodos y propiedades proporcionados por Tkinter. Además, Tkinter es adecuado tanto para proyectos pequeños como para aplicaciones más grandes y complejas, lo que lo convierte en una opción versátil para el desarrollo de aplicaciones GUI en Python.

B. Listas Enlazadas

Las listas enlazadas son una estructura de datos fundamental en informática. Consisten en una secuencia de elementos llamados nodos, donde cada nodo contiene un valor y una referencia (o enlace) al siguiente nodo en la secuencia. Estos nodos están dispuestos de manera consecutiva en la memoria, pero no necesariamente contiguos, lo que permite una flexibilidad en la inserción y eliminación de elementos en la lista.

Una característica clave de las listas enlazadas es su capacidad para crecer o disminuir dinámicamente según sea necesario. Los nuevos nodos pueden ser agregados o eliminados fácilmente sin requerir una

reorganización de toda la estructura de datos. Esto las hace ideales para aplicaciones donde el tamaño de la lista puede variar de manera impredecible.

Además, las listas enlazadas ofrecen un acceso eficiente a los elementos mediante el recorrido secuencial desde el principio o desde cualquier otro nodo de la lista. Cada nodo almacena información sobre el siguiente nodo (apuntador), lo que permite navegar por la lista de forma rápida y eficiente.

C. Recursividad

La recursividad es un concepto fundamental en programación que se refiere a la capacidad de una función para llamarse a sí misma en su definición. Esta técnica permite abordar problemas de manera más elegante y eficiente al dividirlos en casos más pequeños y manejables. Un aspecto crucial de la recursividad es la definición de un caso base que detenga la recursión, evitando así que la función se llame infinitamente. Este enfoque modular y autoreferencial facilita la resolución de problemas complejos al dividirlos en subproblemas más simples.

La importancia de la recursividad en programación radica en su capacidad para abordar problemas que tienen una estructura repetitiva. Muchos algoritmos fundamentales, como la búsqueda en profundidad en árboles, la ordenación rápida y la generación de secuencias, se implementan de manera elegante y eficiente mediante la recursividad. Además, la recursividad promueve un diseño de código limpio y modular al fomentar la reutilización de funciones y la separación de preocupaciones.

Sin embargo, es importante tener en cuenta que el uso excesivo de la recursividad puede llevar a problemas de rendimiento y consumo excesivo de memoria, especialmente en lenguajes de programación que no optimizan la recursión.

D. Archivos XML

Los archivos XML, Extensible Markup Language, son un formato de datos ampliamente utilizado para almacenar y transportar información de manera legible tanto para humanos como para máquinas. Su diseño se basa en etiquetas que permiten estructurar los datos de manera jerárquica, similar a HTML, pero con una sintaxis más flexible y extensible. Esto lo hace ideal para representar una amplia variedad de datos, desde configuraciones de software hasta documentos complejos.

Una de las características más importantes de XML es su capacidad para definir sus propias etiquetas y estructuras, lo que lo hace altamente personalizable y adaptable a diferentes necesidades. Esta flexibilidad se logra a través de la especificación de Document Type Definitions (DTD) o XML Schemas, que definen las reglas y la estructura válida del documento XML. Estas especificaciones permiten la validación de los documentos XML para garantizar su conformidad con un conjunto predefinido de reglas.

Python ofrece librerías para la lectura de archivos XML, para la realización de este proyecto se utilizó "xml.dom"

E. Graphviz

Graphviz es una herramienta de software libre y de código abierto utilizada para visualizar estructuras de datos y grafos de manera gráfica. Funciona mediante la especificación de los elementos del grafo y sus relaciones en un lenguaje de descripción de gráficos llamado DOT.

En el proyecto, Graphviz desempeñó un papel importante en la representación gráfica de los laberintos generados a partir de archivos XML. Utilizando la librería Graphviz para Python, se tradujeron los datos de los laberintos en un formato compatible con DOT, que describe las estructuras de las maquetas. Luego, esta descripción se procesó con Graphviz para generar imágenes visuales de los laberintos, lo que facilitó la comprensión y la visualización de la estructura de los laberintos.

La integración de Graphviz en el proyecto permite a los usuarios visualizar de manera clara y concisa los laberintos y sus características, así como también la solución de los mismos.

Conclusiones

La utilización de listas enlazadas proporcionó una estructura de datos dinámica y eficiente para almacenar y manipular la información de manera ordenada, lo que facilitó el procesamiento de las maquetas y su estructura.

La programación orientada a objetos (POO) permitió modelar de manera clara y modular los diferentes elementos involucrados en el proyecto, como las maquetas, entradas, objetivos y estructuras. Esta metodología de diseño de software facilitó la reutilización de código, la organización del proyecto y la implementación de funcionalidades adicionales de manera sencilla y mantenible.

La recursividad fue fundamental para la resolución del problema principal, ya que permitió explorar de todas las posibles rutas en el laberinto en busca de los objetivos. Este enfoque algorítmico demostró ser eficiente y escalable, lo que garantizó la solución óptima del problema independientemente del tamaño y la complejidad del laberinto.

Por último, la integración de Graphviz para la visualización de los laberintos proporcionó una herramienta poderosa y efectiva para representar gráficamente las estructuras de las maquetas o laberintos.

Referencias bibliográficas

Lutz, M. (2013). Learning Python, 5th Edition. O'Reilly Media.

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.

Anexos

```python
def buscarCamino(self,fila,columna,nueva_lista_estructuras,objetivo):
    if nueva_lista_estructuras.devolver_caracter(fila,columna)==objetivo:
        return True
    else:
        if (nueva_lista_estructuras.devolver_caracter(fila,columna)!="*" and
            nueva_lista_estructuras.devolver_caracter(fila,columna)!="#"):

            nueva_lista_estructuras.reemplazarCaracter(fila,columna, "#")
            if  (self.buscarCamino(fila-1,columna,nueva_lista_estructuras,objetivo) or #abajo
                 self.buscarCamino(fila+1,columna,nueva_lista_estructuras,objetivo) or #arriba
                 self.buscarCamino(fila,columna-1,nueva_lista_estructuras,objetivo) or #izquierda
                 self.buscarCamino(fila,columna+1,nueva_lista_estructuras,objetivo)):  #derecha
                return True
            else:
                nueva_lista_estructuras.reemplazarCaracter(fila,columna, "-")
    return False
```

Figura 1. Algoritmo de búsqueda recursivo.

Fuente: elaboración propia.

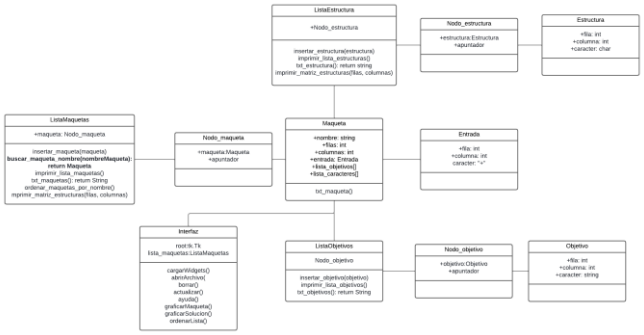

Figura 2. Diagrama de clases del proyecto.

Fuente: elaboración propia.