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 2023.

PRIMER PROYECTO IPC 2, "EXPERIMENTO MARCIANO"

202100119 – Samuel Isaí Muñoz Pereira

Resumen

Para este proyecto de IPC 2 se pidió crear un programa el cual pueda detectar casillas cercanas a un mismo tipo de variable, de modo que es necesario utilizar un algoritmo que pueda identificar variables del mismo tipo que estén sobre la horizontal, vertical o diagonal de esta.

En este proyecto es necesario implementar grafos, para eso deberemos hacer uso de nodos implementando el paradigma de programación orientada a objetos, (POO)

Para la implementación de grafos se estará haciendo uso de la herramienta Graphviz, la cual permite generar graficas de nodos, lo cual es muy necesario en esta práctica.

El fin de este programa es poner en práctica nuestras habilidades como aprendices de programación, y mejorar en la implementación de POO, a su vez conocer nuevas herramientas y librerías que serán de ayuda en próximos proyectos

Palabras clave
POO, grafos, algoritmo, paradigma

Abstract

For this IPC 2 project, it was requested to create a program that can detect boxes near the same type of variable, so it is necessary to use an algorithm that can identify variables of the same type that are on the horizontal, vertical, or diagonal of this.

In this project, it is necessary to implement graphs, for that, we will have to make use of nodes implementing the object-oriented programming paradigm (OOP).

For the implementation of graphs, we will be using the Graphviz tool, which allows generating node graphs, which is very necessary in this practice.

The purpose of this program is to put our skills as programming learners into practice, and improve in the implementation of OOP, at the same time, to know new tools and libraries that will be helpful in future projects.

Keywords
OOP, graphs, algorithm, paradigm.

Introducción

El presente proyecto tiene como objetivo poner a prueba las habilidades básicas que todo programador debe de tener, como la creación de algoritmos de detección y acción para diversos tipos de situaciones, algoritmos que se puedan adaptar dependiendo el campo de acción.

Fue necesario implementar POO para esta practica ya que se solicita la implementación de nodos.

Para los nodos fue necesario la creación de un esquema el cual se pueda usar como una matriz, esto hizo que la practica fuera un reto complejo ya que de esta manera es necesario la implementación de más lógica.

También se hizo uso de algunas librerías y herramientas extras como lo es Graphviz, una herramienta de creación de grafos la cual se combina perfectamente con los nodos de esta práctica.

Desarrollo del tema

Carga del Archivo XML:

Para poder manipular de manera apropiada el archivo XML fue necesario hacer uso de la librería `xml.etree.ElementTree`, con esta librería puedes leer y escribir archivos XML, analizar y modificar elementos y atributos del documento XML, y navegar por la estructura del documento utilizando una sintaxis sencilla y familiar similar a la del acceso a los elementos de un diccionario o lista.

Además, esta librería proporciona métodos para buscar y seleccionar elementos específicos dentro del documento XML, y para generar documentos XML a partir de datos estructurados en Python.

```python
def cargaArchivo():
    ruta = input("Ingrese la ruta ABASOLUTA del arhivo: ")
    tree = ET.parse(ruta)
    root = tree.getroot()


    for organismo in root.findall(".//organismo"):
        codigo = organismo.find("codigo").text
        nombre = organismo.find("nombre").text

        #print(f"Código: {codigo}, Nombre: {nombre}")
```

Figura 1. Implementación de lectura XML

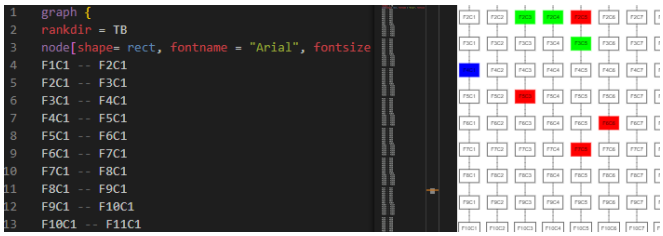
Graphviz

Graphviz es una herramienta de visualización de gráficos que permite crear diagramas y visualizaciones de manera rápida y sencilla. Es muy útil para representar relaciones y conexiones entre distintos elementos o entidades, como por ejemplo en diagramas de flujo, árboles genealógicos, mapas mentales, diagramas de redes, etc.

Con Graphviz, puedes crear diagramas a partir de archivos de texto que describen la estructura y relaciones entre los elementos. La herramienta se encarga de generar automáticamente el diagrama en una variedad de formatos (PDF, PNG, SVG, etc.) que puedes usar para presentar, compartir o imprimir.

La sintaxis de Graphviz es bastante sencilla y se basa en lenguaje de marcado DOT. En este lenguaje, puedes definir los nodos, bordes, colores, estilos y atributos que quieres utilizar para construir el diagrama. También puedes agregar etiquetas y texto para hacer más claro y legible el diagrama.

para este proyecto fue indispensable el uso de Graphviz ya que la representación de la matriz fue hecha en un tablero con un archivo .DOT, en la cual se encuentran todas las conexiones necesarias para la representación precisa de un tablero con filas y columnas



```
graph {
rankdir = TB
node[shape= rect, fontname = "Arial", fontsize
F1C1 -- F2C1
F2C1 -- F3C1
F3C1 -- F4C1
F4C1 -- F5C1
F5C1 -- F6C1
F6C1 -- F7C1
F7C1 -- F8C1
F8C1 -- F9C1
F9C1 -- F10C1
F10C1 -- F11C1
```

Figura 2. Archivo .DOT para la creación de conexiones y resultado de grafo

Listas Enlazadas:

En este proyecto explícitamente estaba prohibido el uso de listas nativas para el manejo de los datos, lo cual fue un reto muy importante, debido a esta restricción fue necesario el uso de listas enlazadas, en las cuales tendríamos que ordenar de manera que pudiéramos recorrer de manera simple buscando cada uno de las células vivas.

En mi caso yo hice uso particular de una lista simplemente enlazada, lo que hizo que el trabajo se complicara un poco, debido a que necesitaba recorrer toda la lista en caso que fuera necesario extraer un dato de esta.

```python
class Lista():

    def __init__(self):
        self.primero = None
        self.ultimo = None


    def vacio(self):
        return self.primero == None


    def agregar(self, dato):
        if self.vacio() == True:
            self.primero = self.ultimo = nodo(dato)

        else:
            aux = self.ultimo
            self.ultimo = aux.siguiente = nodo(dato)


    def recorrer(self):
        aux = self.primero
        while aux != None:
            print("Fila:", aux.dato.fila_CelulaViva, "Columna:",aux.dato.columna_CelulaViva,
            aux = aux.siguiente
```

Figura 3. Lista simplemente enlazada y sus métodos

POO:

En mi caso el POO no fue tan bien implementado, ya que solo hacia uso de una clase para guardar los datos de las células vivas presentes en el archivo XML, por lo que no se utilizaron métodos para el manejo de los datos

El POO también fue implementado en las listas enlazadas, tanto para crear cada uno de los nodos como para extraer, recorrer, imprimir, plasmar, entre otros, cada uno de los datos presentes en la lista

```python
class celulasVivas():

    def __init__(self, fila_CelulaViva, columna_CelulaViva, codigo_CelulaViva):
        self.fila_CelulaViva = fila_CelulaViva
        self.columna_CelulaViva = columna_CelulaViva
        self.codigo_CelulaViva = codigo_CelulaViva


    def get_fila_CelulaViva(self):
        return self.fila_CelulaViva

    def get_columna_CelulaViva(self):
        return self.columna_CelulaViva

    def get_codigo_CelulaViva(self):
        return self.codigo_CelulaViva
```

Figura 4. Programación orientada a objetos para guardar datos

Una de las recomendaciones de la programación orientada a objetos es el uso de métodos para poder reciclar los mismos y utilizarlos para diversos motivos, pero debido a que la naturaleza de este proyecto es más logística y no de repetición, en mi caso no vi la necesidad de métodos para acceder a los datos, debido que con las listas enlazadas podía llamar a mis datos haciendo referencia a ellos como objetos de la clase

Uso de bucles o ciclos:

En esta presentación fueron muy utilizados los bucles ya que con estos se recorrían las lista una y otra vez dependiendo del propósito, también se utilizaron bucles for para poder generar el archivo .DOT que seria el que Graphviz tomaría para generar el pdf con todas las conexiones

Archivos presentes en todo el programa:

- Carga.py: en este archivo se aloja la función que hace posible cargar o agregar los datos necesarios del archivo XML en la lista enlazada.
- clasesDatos.py: contiene las clases que se utilizan para guardar los datos de manera organizada.
- CreacionTablero.py: contiene todos los algoritmos para la creación y manejo del archvio.dot, de manera que se pueda crear los reportes en Graphviz de una manera automatizada con poca intervención del usuario
- Main.py: es el archivo principal desde donde se carga todos los demás archivos del programa, desde ahí se selecciona las opciones del menú, y el usuario elige en que parte del programa desea acceder.
- Nodo.py: en este archivo se encuentra la clase nodo y la clase de lista enlazada, en la cual se encuentran todos los métodos para agregar, leer y pintar el tablero con los datos.

Como se Utiliza el programa:

De primero el programa redirige al usuario al menú principal, en el cual encuentra las siguientes opciones:

- Creación del tablero: en esta opción el programa permitirá ingresar el número de filas y columnas que desee el usuario, el programa solo admite diez mil filas y columnas, por lo que pasarse de alguna de estas dos cifras, ara que el programa vuelva a solicitar la información.
  Una vez el usuario ingresa los datos, el programa se encarga de crear cada una de las conexiones necesarias en graphviz, una vez termina el usuario genera un pdf mostrando el tablero.

- Graficar el Tablero: esta opción permite leer el archivo XML, y se encarga de pintar con un color diferente cada uno de los organismos presentes en el archivo. Por último, sobrescribe los datos presentes en el archivo.dot y genera el pdf. Donde se encuentra el tablero con las células pintadas

Conclusiones

Este proyecto fue un nuevo desafío en el cual se pusieron a prueba mis habilidades de programación, lamentablemente no puede terminar en su totalidad el proyecto, pero puedo decir que en lo poco que logré realizar aprendí sobre el manejo de las listas enlazadas y la codificación de algoritmos de reconocimiento aun que desde mi punto de vista es mucho mas simple utilizar listas nativas en Python para el manejo de cierto tipos de datos. Aun que es totalmente comprensible que el uso de listas enlazadas permite crear métodos y funciones específicamente para usos personales y acondicionarlas para propósitos muy específicos

Referencias bibliográficas

Dasgupta, Anirban, et al. Algorithms. McGraw-Hill Education, 2018.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., & Woodhull, G. (2002). Graphviz - Open Source Graph Drawing Tools. In International Symposium on Graph Drawing (pp. 483-484). Springer.

Graphviz documentation, available at: https://graphviz.org/documentation/

North, S. C. (2003). Toward measuring visualization insight. IEEE Computer Graphics and Applications, 23(5), 6-9

Pilgrim, Mark. Dive Into Python 3. Apress, 2012.

Zelle, John M. Python Programming: An Introduction to Computer Science. Franklin, Beedle & Associates Inc., 2017.