Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la Programación y Computación 2, 2do. Semestre 2022.

PROYECTO 1

202113553 – Kevin Ernesto García Hernández

Resumen

El laboratorio de investigación epidemiológica de Guatemala ha estado investigando la forma en que las enfermedades infectan las células del cuerpo humano y se expanden produciendo enfermedades graves e incluso la muerte.

Los científicos, luego de hacer experimentos han logrado identificar patrones que pueden determinar si una enfermedad producirá una enfermedad leve, una enfermedad grave, o si el paciente morirá a causa de la enfermedad. La forma en que identifican los patrones que pueden llegar a determinar las características de la gravedad de la enfermedad es a través del uso de rejillas cuadradas con tejido del paciente, estas rejillas contienen una célula en cada celda y cada célula puede estar saludable o contagiada.

Por lo cual, los científicos han identificado si una célula sana tiene exactamente tres vecinas infectadas se infectará, de lo contrario seguirá sana. Y si una célula infectada tiene dos o tres vecinas infectadas se infectará, de lo contrario sanará.

Abstract

The Guatemalan epidemiological research laboratory has been investigating how diseases infect the cells of the human body and spread, causing serious illness and even death.

Scientists, after doing experiments, have been able to identify patterns that can determine if a disease will produce a mild illness, a severe illness, or if the patient will die from the disease. The way in which they identify the patterns that can determine the characteristics of the severity of the disease is through the use of square grids with the patient's tissue, these grids contain a cell in each cell and each cell can be healthy or infected.

Therefore, scientists have identified if a healthy cell has exactly three infected neighbors it will become infected, otherwise it will remain healthy. And if an infected cell has two or three infected neighbors it will get infected, otherwise it will heal.

Palabras clave

Nodo: Es un punto de intersección o unión de varios elementos que confluyen en el mismo lugar.

Lista enlazada simple: Es una estructura de datos en la que cada elemento apunta al siguiente. De este modo, teniendo la referencia del principio de la lista podemos acceder a todos los elementos de esta.

Lista doblemente enlazada: Una lista doblemente enlazada es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior.

Row-Major: Es un método para almacenar matrices multidimensionales en almacenamiento lineal, como la memoria de acceso aleatorio.

POO: La Programación Orientada a Objetos es un paradigma de programación que parte del concepto de "objetos" como base, los cuales contienen información en forma de campos y código en forma de métodos.

GUI: Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Graphviz: Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT. Fue desarrollado por AT&T Labs y liberado como software libre con licencie tipo Eclipse.

XML: Traducido como 'Lenguaje de Marcado Extensible' o 'Lenguaje de Marcas Extensible', es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium utilizado para almacenar datos en forma legible.

Keywords

Node: *It is a point of intersection or union of several elements that come together in the same place.*

Simple linked list: *It is a data structure in which each element points to the next. In this way, having the reference of the beginning of the list we can access all the elements of it.*

Doubly linked list: *A doubly linked list is a linear list in which each node has two links, one to the next node, and one to the previous node.*

Row-Major: *It is a method of storing multidimensional arrays in linear storage, such as random access memory.*

OPP: *Object Oriented Programming is a programming paradigm based on the concept of "objects" as a base, which contain information in the form of fields and code in the form of methods.*

GUI: *It is a computer program that acts as a user interface, using a set of images and graphic objects to represent the information and actions available in the interface.*

Graphviz: *It is a computer program that acts as a user interface, using a set of images and graphic objects to represent the information and actions available in the interface.*

XML: *'Extensible Markup Language' or 'Extensible Markup Language', is a metalanguage that allows defining markup languages developed by the World Wide Web Consortium used to store data in readable form.*

Introducción

El desarrollo del proyecto se realizó en base al paradigma de la Programación Orientada a Objetos (POO), esto tanto para la realización de la GUI (Interfaz Gráfica) como para los distintos funcionamientos principales o contenedores de los datos principales a usar en el sistema.

La carga de datos se realizó en base al formato XML (Lenguaje de Marcado Extensible), donde se encontraban cada uno de los pacientes con sus atributos y datos a evaluar. Igualmente se utilizó el formato XML para la salida de los resultados del paciente que se evaluó.

Para el manejo de las rejillas de las células se utilizó el Row Major, siendo este muy parecido a una lista doblemente enlazada con la única diferencia de que se simulan dos dimensiones en una sola.

Por último, se implementó Graphviz para mostrar el comportamiento de la rejilla durante cada uno de los periodos de simulación.

Desarrollo del tema

El desarrollo del proyecto se llevó a cabo en el lenguaje de Python, siendo utilizado un entorno virtual para la realización de este; esto para descargar librerías externas a utilizar.

Módulos

Este proyecto fue realizado en varios módulos para tener un mayor orden de todo, linkeando cada una de las funciones necesarias con sus respectivos módulos.

- **main:** Siendo el módulo que se ejecutará primero al iniciar el programa, conteniendo toda la interfaz gráfica.
- **load:** Siendo el módulo donde se realiza la lectura del archivo XML que se seleccione.
- **object:** Siendo el módulo que contiene los objetos principales donde se guardarán los datos leídos en el XML, al igual que contiene gran mayoría de funciones de estos objetos.
- **lists:** Siendo el módulo que contiene la lista enlazada utilizando Row Major para el desarrollo de las rejillas.
- **simulation:** Siendo el módulo que se ejecutará al iniciar la simulación de un paciente en la GUI.
- **algorithm:** Siendo el módulo que contiene la mayoría de lógica para poder realizar la simulación de las rejillas, compararlas, graficarlas y retornar un resultado de estas mismas para cambiarlo en su atributo.
- **diagnostic:** Siendo el módulo que devuelve un archivo XML con los resultados de la simulación que se realizó del paciente.

Clases Objetos

Para guardar los datos a leer en el archivo XML se utilizó una clase de objeto Pacient, en este se contienen cada uno de los atributos a leer y a generar en su posterior resultado de salida.

También se realizó una clase objeto de Grid, esta conteniendo cada una de las rejillas de una simulación.

Para la realización y manejo de datos de esta clase se siguió el siguiente diagrama de clases.

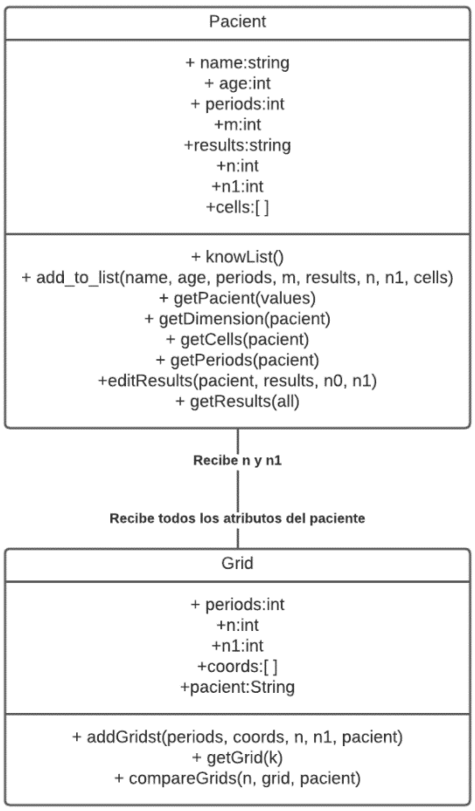

Figura 1. Diagrama de Clases

Fuente: Elaboración propia, 2022.

Lista Enlazada

Para poder manejar la rejilla que contendrá cada una de las células enfermas y sanas que nos proporciona el archivo de entrada se utilizará una lista doblemente enlazada, con la única diferencia que en esta se manejará el método Row Major para simular las dos dimensiones con mucha mayor facilidad.

Esta conteniendo una clase Node para manejar la posición de la lista y la clase List, donde se contendrán las mayorías de funciones de esta (como insertar, obtener diferentes valores, aplicar el Row Major, etc).

```
class Node:

    def __init__(self, pos=0, value=None, status=None) -> None:
        self.pos = pos
        self.value = value
        self.status = status
        self.next = None

class List:

    def __init__(self) -> None:
        self.size = 0
        self.root = None

    def insert(self, value, status):
        if(self.root==None):
            self.root = Node(self.size, value, status)

        else:
            aux = self.root
            while aux.next != None:
                aux = aux.next
            aux.next = Node(self.size, value, status)

        self.size +=1

    def getMatrix(self, size):
        for i in range(size):
            self.insert("▒", 0)

    def editCell(self, value, status, i, j, max):
        pos = self.rowMajor(j,i,max)
        aux = self.root

        while aux != None:
            if aux.pos == pos:
                aux.value = value
                aux.status = status
                break
            aux = aux.next

    def getCell(self, i, j, max):
        pos = self.rowMajor(j,i,max)
        aux = self.root

        while aux != None:
            if aux.pos == pos:
                return aux.value
            aux = aux.next
        return "Error"

    def getStatus(self, i, j, max):
        pos = self.rowMajor(j,i,max)
        aux = self.root

        while aux != None:
            if aux.pos == pos:
                return aux.status
            aux = aux.next
        return "Error"

    def rowMajor(self, x, y, maxY):
        return x + y * maxY
```

Figura 2. Clases de la lista enlazada.

Fuente: Elaboración propia, 2022.

Interfaz Gráfica

Para la realización de la interfaz gráfica se utilizó la librería de Tkinter, utilizando POO para cada una de las clases que contienen las distintas ventanas que conforman la GUI, esto para hacer herencia de la ventana principal para permitir ocultarla y mostrarla cuando se necesite.

Como se puede ver en el siguiente diagrama, la clase de Menu (siendo esta la pantalla principal) heredando la clase Tk de Tkinter, tiene las funciones de openFile, simulate y exit.

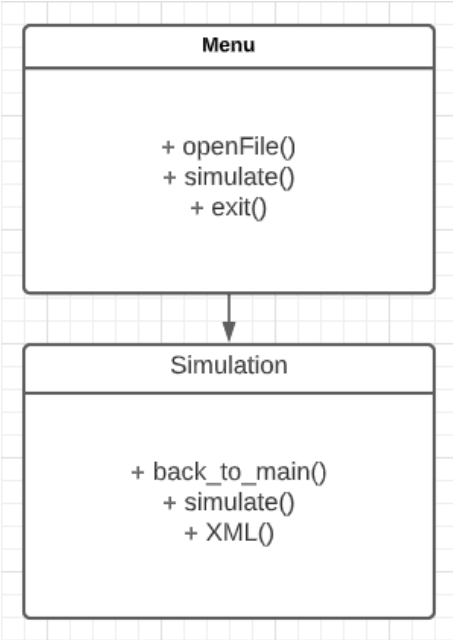

Figura 3. Diagrama de la Interfaz

Fuente: Elaboración propia, 2022.

En caso de que la función simulate de la clase Menu se ejecute esta llamará a la clase Simulation, heredando TopLevel de Tkinter, esta contará con las funciones de back_to_main, simulate y XML.

Funciones de la GUI

openFile: Esta función contenida en un botón de la pantalla principal hará la función de cargar un archivo con la función de filedialog, donde posteriormente leerá el archivo XML con la función read del módulo load. También tiene una validación para saber si se cargó algún archivo para mostrarle un mensaje al usuario con ayuda de messagebox.

simulate: Esta función llamará a la clase Simulation, la cual generará una nueva ventana para la realización de la simulación; esta también esconderá la ventana principal

exit: Esta función dejará de ejecutar el programa, cerrando la GUI consigo.

back_to_main: Esta función destruirá la ventana de Simulation y volverá a mostrar la ventana principal.

simulate: Esta función obtendrá el dato del ComboBox y con ello llamará a la función simulate(paciente) del módulo simulation. También mostrará un mensaje al usuario cuando se finalice la simulación.

XML: Esta función obtendrá el dato del ComboBox y con ello llamará a la función generateXML(paciente) del módulo diagnostic. También mostrará un mensaje al usuario cuando se genere el archivo XML de salida.

Funciones Importantes

simulate(pacient): Esta recibirá como parámetro el nombre de un paciente, obteniendo cada uno de sus atributos con las funciones de la clase Pacient. Con ello obtendrá cada una de las coordenadas con células infectadas e imprimirá la matriz inicial en consola. Por último, llamará a la función eva evaluateMatrix(m, matrix, periods, pacient) del módulo algorithm.

```
def simulate(pacient):

    folder = 'Gráficas'

    for f in os.listdir(folder):
        filep_path = os.path.join(folder, f)
        os.unlink(filep_path)

    folder = 'Resultados'

    for f in os.listdir(folder):
        filep_path = os.path.join(folder, f)
        os.unlink(filep_path)


    m = Pacient.getDimension(pacient)

    matrix = List()

    matrix.getMatrix(m * m)

    print_matrix(matrix, m)

    #---------------------------------

    getInfected(matrix, m, pacient)

    periods = Pacient.getPeriods(pacient)

    algorithm.graphicInitial(matrix, m)

    algorithm.evaluateMatrix(m, matrix, periods, pacient)
```

Figura 4. Función simulate.

Fuente: Elaboración propia, 2022.

evaluateMatrix(m, matrix, periods, pacient): Esta recibirá como parámetro la dimensión de la matriz, la matriz con las celdas infectadas, los periodos y el nombre el paciente. Con ello iterará hasta el periodo obtenido e iterará la matriz para obtener la celda actual junto con todos sus vecinos, luego evaluará si estas están enfermas o no para cambiarle su estado. Fuera de la iteración de todas las celdas, se cambiará definitivamente el valor de cada una de las celdas, esto para imprimir la rejilla en la consola al igual que generar una imagen png con Graphviz. Por último, se hará una comparación del patrón actual con cada uno de los anteriores, esto para encontrar si hay uno repetido, para así obtener los resultados correspondientes y parar la iteración de periodos.

```
def evaluateMatrix( m, matrix, periods, pacient):

    for n in range(periods):

        grid = []

        for i in range(m):
            for j in range(m):

                actualCell = matrix.getCell(i,j,m)

                neighbor1 = matrix.getCell(i-1,j-1,m)
                neighbor2 = matrix.getCell(i-1,j,m)
                neighbor3 = matrix.getCell(i-1,j+1,m)
                neighbor4 = matrix.getCell(i,j-1,m)
                neighbor5 = matrix.getCell(i,j+1,m)
                neighbor6 = matrix.getCell(i+1,j-1,m)
                neighbor7 = matrix.getCell(i+1,j,m)
                neighbor8 = matrix.getCell(i+1,j+1,m)

                sickCells =
        evaluateCells(neighbor1,neighbor2,neighbor3,neighbor4,neighbor5,neighbor6,neighbor7,neighbor8)

                evaluateSicks(actualCell, sickCells, matrix, m , i, j)

        infecteCells(matrix, m, grid)

        simulation.printPeriods(matrix, m, n)

        results = Grids.compareGrids(n+1, grid, pacient)

        graphicMatrix(matrix, m, str(n+1))


        if results[0] == "Enfermedad MORTAL":

            break

        elif results[0] == "Enfermedad GRAVE":

            break

        Grids.addGrids(n+1, grid, pacient)

    print("Periodos: ", results[1], results[2])

    Pacient.editResults(pacient, results[0], results[1], results[2])
```

Figura 5. Función evaluateMatrix.

Fuente: Elaboración propia, 2022.

graphicMatrix(matrix, m, n): Esta recibirá como parámetro la dimensión de la matriz, la matriz y el periodo actual de la simulación. Con ello, generará un texto con estructura de un archivo .dot, iterando una tabla las veces que sea necesario hasta que quede la matriz ingresada. Luego, se generará un archivo Graphviz el cual contendrá el texto generado anteriormente, esto para generar una imagen .png con el archivo de Graphviz, guardándolo en una carpeta con el número de periodo que corresponde.

```python
def graphicMatrix(matrix, m, n):

    graphviz = 'digraph EJEMPLO{\n    node [shape=plaintext];'
    graphviz += '\n    struct1 [label=<'
    graphviz += '\n        <TABLE>'

    for i in range(m):
        graphviz += '\n         <TR>'

        for j in range(m):

            if matrix.getStatus(i, j, m) == 1:

                graphviz += '\n             <td bgcolor="red"></td>'

            elif matrix.getStatus(i, j, m) == 0:
                graphviz += '\n             <td bgcolor="green"></td>'

        graphviz += '\n         </TR>'

    graphviz += '\n        </TABLE>'

    graphviz += '\n    >];'
    graphviz += '\n}'


    with open('Gráficas\graphviz.txt', 'w') as file:

            file.write(graphviz)

    os.system('dot.exe -Tpng Gráficas\graphviz.txt -o Gráficas\Período_'+n+'.png')
```

Figura 6. Función graphicMatrix.

Fuente: Elaboración propia, 2022.

generateXML(pacient): Esta recibirá como parámetro el nombre el paciente contenido en el ComboBox. Con ello encontrará todos los datos del paciente y con la librería xml.etree.ElementTree se escribirá un archivo XML de salida, conteniendo los resultados del paciente tras la simulación del mismo; este guardándolo en una carpeta. Consigo, abrirá este mismo archivo al usuario para que lo pueda ver.

```python
def generateXML(pacient):

    all = []

    all = Pacient.getResults(all)

    pacients = ET.Element("pacientes")

    for i in range(len(all)):

        if all[i][0] == pacient:

            pacient= ET.Element("paciente")
            pacients.append(pacient)

            dates = ET.Element("datospersonales")
            pacient.append(dates)

            name = ET.SubElement(dates, "nombre")
            name.text = all[i][0]

            age = ET.SubElement(dates, "edad")
            age.text = str(all[i][1])

            period = ET.SubElement(pacient, "periodos")
            period.text = str(all[i][2])

            m = ET.SubElement(pacient, "m")
            m.text = str(all[i][3])

            result = ET.SubElement(pacient, "resultado")
            result.text = all[i][4]

            if all[i][6] != 0:
                n = ET.SubElement(pacient, "n")
                n.text = str(all[i][5])

                n1 = ET.SubElement(pacient, "n1")
                n1.text = str(all[i][6])

            if all[i][6] == 0:

                n = ET.SubElement(pacient, "n")
                n.text = str(all[i][5])


    tree = ET.ElementTree(pacients)
    ET.indent(pacients)
    tree.write("Resultados\Resultados.xml", xml_declaration=True, encoding='utf-8')
    webbrowser.open('Resultados\Resultados.xml')
```

Figura 7. Función generateXML.

Fuente: Elaboración propia, 2022.

Conclusiones

Con todo esto se puede concluir que la utilización de las listas enlazadas es muy importante para las funciones de obtener ciertos datos, al igual si se trata de realizar muchas iteraciones.

Al igual que un buen manejo de POO para la realización de cualquier proyecto, ya que con este paradigma podemos llegar a ahorrarnos bastante trabajo en lo que se refiere a código.

Por último, es importante saber la implementación de poder leer y escribir un archivo XML, siendo este uno de los formatos más utilizados comercialmente actualmente.

Referencias

- Python Software Foundation. (2022). tkinter — Python interface to Tcl/Tk. Python. Recuperado 2 de septiembre de 2022, de https://docs.python.org/3/library/tkinter.html
- Python Software Foundation. (2022). xml.etree.ElementTree — The ElementTree XML. Python. Recuperado 2 de septiembre de 2022, de https://docs.python.org/3/library/xml.etree.elementtree.html
- The Graphviz Authors. (2022). Documentation. Graphviz. Recuperado 2 de septiembre de 2022, de https://graphviz.org/documentation/

Anexos

```
def read (filepath):

    try:

        tree = ET.parse(filepath)

        root = tree.getroot()

        for patient in root:

            if patient.tag == "paciente":

                for data in patient:

                    if data.tag == "datospersonales":

                        for sub_data in data:

                            if sub_data.tag == "nombre":

                                name = sub_data.text

                            elif sub_data.tag == "edad":

                                age = int(sub_data.text)

                    elif data.tag == "periodos":

                        periods = int(data.text)

                    elif data.tag == "m":

                        m = int(data.text)

                    elif data.tag == "rejilla":

                        cells = []

                        for sub_data in data:


                            if sub_data.tag == "celda":

                                f = int(sub_data.get('f'))

                                c = int(sub_data.get('c'))

                                coords = [f, c]

                                cells.append(coords)

                results = ""
                n = 0
                n1 = 0

                Pacient.add_to_list( name, age, periods, m, results, n, n1, cells)

    except:
        messagebox.showerror(message="No se cargó ningún archivo.", title="Sin cargar")
```

Figura 8. Función readXML.

Fuente: Elaboración propia, 2022

```
def getInfected(matrix, m, pacient):

    cells = Pacient.getCells(pacient)

    initial = []

    for i in range(len(cells)):
        x = cells[i][0] - 1          #Esto para que quede en coordenadas de la matriz
        y = cells[i][1] - 1
        matrix.editCell("■", 1, x, y ,m)

        coords = [x,y]

        initial.append(coords)


        if i == len(cells)-1:
            Grids.addGrids(0, initial, pacient)
```

Figura 9. Función getInfected.

Fuente: Elaboración propia, 2022.

```
def printPeriods(matrix, m, periods):

    print("Perído ",periods + 1,"\n")

    for i in range(m):
        for j in range(m):
            print(" ", matrix.getCell(i,j,m) ,end=" ")
        print("\n")
```

Figura 10. Función printPeriods.

Fuente: Elaboración propia, 2022.

```
def evaluateCells(neighbor1,neighbor2,neighbor3,neighbor4,neighbor5,neighbor6,neighbor7,neighbor8):

    sickCells = 0

    if neighbor1 == "■":
        sickCells += 1

    if neighbor2 == "■":
        sickCells += 1

    if neighbor3 == "■":
        sickCells += 1

    if neighbor4 == "■":
        sickCells += 1

    if neighbor5 == "■":
        sickCells += 1

    if neighbor6 == "■":
        sickCells += 1

    if neighbor7 == "■":
        sickCells += 1

    if neighbor8 == "■":
        sickCells += 1

    return sickCells
```

Figura 11. Función evaluateCells.

Fuente: Elaboración propia, 2022.

```python
def evaluateSicks(actualCell, sickCells, matrix, m, x, y):

    if actualCell == "░":

        if sickCells == 3:
            matrix.editCell(actualCell, 1, x, y ,m)

            return matrix.getStatus(x,y,m)

        else:
            matrix.editCell(actualCell, 0, x, y ,m)

            return matrix.getStatus(x,y,m)

    elif actualCell == "▒":

        if (sickCells == 3) or (sickCells ==2):
            matrix.editCell(actualCell, 1, x, y ,m)

            return matrix.getStatus(x,y,m)

        else:
            matrix.editCell(actualCell, 0, x, y ,m)

            return matrix.getStatus(x,y,m)
```

Figura 12. Función evaluateSicks.

Fuente: Elaboración propia, 2022.

```python
def infecteCells(matrix, m, infected):

    for i in range(m):

        for j in range(m):

            if matrix.getStatus(i, j, m) == 1:
                matrix.editCell("▒", 1, i, j ,m)

                coords = [i,j]
                infected.append(coords)


            elif matrix.getStatus(i, j, m) == 0:

                matrix.editCell("░", 0, i, j ,m)
```

Figura 13. Función infecteCells.

Fuente: Elaboración propia, 2022.

```python
def compareGrids(n, grid, pacient):

        status = ""

        for i in infected:

            if grid == i.coords:

                period_repeated =  i.period

                if period_repeated == 0:

                    n0 = n - period_repeated
                    n1 = 0

                    if n0 == 1:
                        status = "Enfermedad MORTAL"
                        return status, n0, n1

                    else:
                        status = "Enfermedad GRAVE"

                else:
                    n0 = n
                    n1 = n - period_repeated

                    if n1 == 1:
                        status = "Enfermedad MORTAL"
                        return status, n0, n1


                    else:
                        status = "Enfermedad GRAVE"

        if status == "Enfermedad GRAVE":
            return status, n0, n1

        else:
            status = "Enfermedad LEVE"
            n0 = 0
            n1 = 0
            return status, n0, n1
```

Figura 14. Función compareGrids.

Fuente: Elaboración propia, 2022.