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

202109567 – Alvaro Norberto García Meza

Resumen

Dicho proyecto fue propuesto con la finalidad de crear una aplicación a petición del laboratorio de investigación epidemiologia de Guatemala, para analizar la forma en que las enfermedades infectan las células del cuerpo humano y se expanden produciendo enfermedades graves e incluso la muerte.

Los científicos proveen una rejilla con un numero el cual representa las dimensiones de una matriz cuadrada y que esta compuesta por filas y columnas, donde cada celda puede o no estar infectada. Mediante un patrón estudiado se observa en un determinado número de periodos el desarrollo de esta rejilla para posteriormente devolver un diagnostico que determine la gravedad de la enfermedad y otras señales tales como: en qué periodo este patrón de células infectadas vuelve a repetirse y cada cuanto se en cicla dicho periodo si es que contiene.

Palabras clave

Rejilla, periodo, infectada, patrón y diagnóstico.

Abstract

This project was proposed with the aim of creating an application, at the request of the Guatemalan Epidemiological Research Laboratory, to analyze how diseases infect the cells of the human body and spread, causing serious illness and even death.

The scientists provide a grid with a number that represents the size of a square matrix and is made up of rows and columns, in which each cell may or may not be infected. Through a studied pattern, the development of this grid is observed over a series of periods to later give a diagnosis that determines the severity of the disease and other signals such as: in which period this pattern of infected cells is repeated and how often this period occurs in cycle if it contains it.

Keywords

Grid, period, infected, pattern and diagnostic

Introducción

El desarrollo de la aplicación se realizó con la ayuda de diferentes paradigmas de programación y el uso de estructuras de datos. Los paradigmas de programación implementados fueron Programación Orientada a Objetos, programación funcional y modular para desarrollar de manera ordenada las diferentes funcionalidades las cuales en algunos casos eran dependientes de otras.

La carga de los datos fue establecida al formato XML (Lenguaje de Marcado Extensible), donde contenía los pacientes a analizar con sus respectivos atributos tales como: nombre, edad, periodo, tamaño de la rejilla y apartado con las células especificas infectadas donde al leerlo y evaluarlo retornaba un mismo formato XML con algunos atributos diferentes tales como resultado, n y n1.

Para simular el comportamiento de las células en una matriz utilizando una lista simple enlazada, se implemento el concepto de row mayor, el cual con un simple cálculo matemático se podía obtener la posición de dicha célula en una matriz cuadrada.

Por último, se implementó la herramienta grahpviz la cual generaba una imagen del ultimo periodo ejecutado.

Desarrollo del tema

La aplicación fue desarrollada con el lenguaje de programación “Python” debido a su gran facilidad de sintaxis y manejo de variables. Además, implementando librerías integradas del propio lenguaje.

A. Directorios y Módulos

Cada proyecto y aplicación debe contener cierta jerarquía y organización para un manejo ordenado del a información, en este caso un buen manejo de la funcionalidad y datos globales.

- **App:** El primero directorio de la jerarquía el cual contiene toda la aplicación
  - **Classes:** Es el primer directorio el cual con tiene los módulos los cuales son todas las clases utilizadas para el desarrollo del programa.
    - **Cell.py**
    - **List.py**
    - **Node.py**
    - **Patient.py**
  - **Components:** El segundo directorio contiene los componentes los cuales le dan la funcionalidad y lógica a la aplicación.
    - **Select_and_execute_patient.py**
  - **App_Consola.py:** Es el modulo principal donde todas las demás clases y módulos se conectan para crear la aplicación.

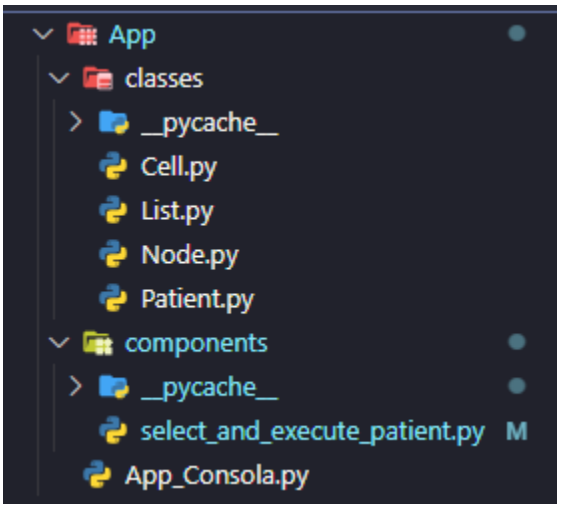

Figura 1. Directorios y Módulos.

Fuente: elaboración propia, 2022.

B. Clases

Al implementar distintos paradigmas de programación como Programación Orientada a Objetos, se implementó distintas clases para almacenar pacientes, células y el uso de una clase nodo la cual se complementaba con la clase la cual era una lista simple enlazada.

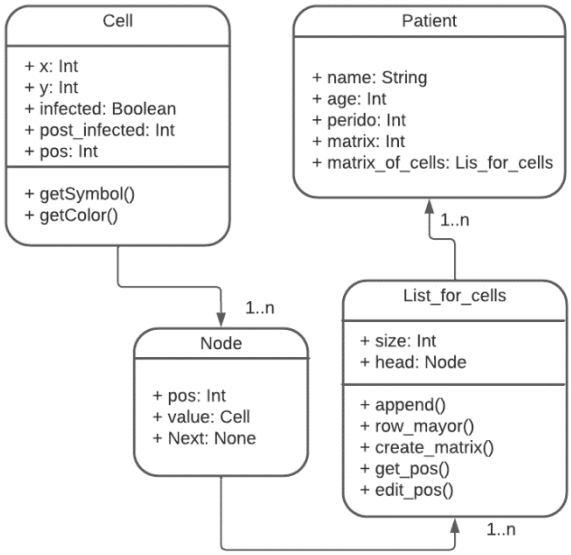

Figura 2. Diagrama de clases.

Fuente: elaboración propia, 2022.

Figura 1. Título o descripción breve de la figura.

Fuente: elaboración propia, o citar al autor, año y página.

C. Lista simple enlazada

Para la simulación de una matriz utilizando listas simples la manera más sencilla de realizarlo es utilizando lo que se conoce como row_mayor el cual con un simple cálculo matemático el retorna lo que es la suma de la posición de la fila y la multiplicación de la columna por el tamaño de la matriz. Con esta simple función se sabe en que posición de la matriz esta la célula. Además, de al ser una lista simple se debe manejar la posición de cada objeto con el uso de nodos el cual apunta al siguiente.

```python
class List_for_cells:
    #constructor
    def __init__(self,) -> None:
        self.size = 0
        self.head: Node  = None


    def append(self,value):
        if self.head == None:
            self.head = Node(self.size,value)
        else:
            aux = self.head
            while aux.next != None:
                aux = aux.next
            aux.next = Node(self.size,value)
        self.size += 1

    def row_mayor(self,x,y,m):
        return x + y*m

    def create_matrix(self,size):
        for i in range(size):
            self.append(None)

    def get_pos(self,x,y,m):
        pos = self.row_mayor(y,x,m)
        aux = self.head

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

    def edit_pos(self,value,x,y,m):
        pos = self.row_mayor(y,x,m)
        aux = self.head

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

Figura 3. Lista simple enlazada.

Fuente: elaboración propia, 2022.

D. Módulos

La aplicación esta divida en dos módulos, el primero contiene el desarrollo de todo el menú principal, donde se puede cagar archivos, ver los pacientes y salir y el segundo módulo contiene toda la funcionalidad lógica que el programa demanda tal

como: escoger un paciente, evaluar un paciente, generar un gráfico de los resultados y a su vez devolver un diagnostico del paciente seleccionado. Además, tiene la posibilidad de terminar de evaluar un paciente y regresar al menú para salir.

Modulo: App_Consola.py

Como primer punto en la figura 4. Es la recopilación de funciones las cuales juntas generan un menú sencillo con 3 opciones a elegir: cargar archivo, ver paciente y salir.

Al momento de seleccionar cargar un archivo la función de la figura 5 se ejecuta, lee el XML y convierte a cada paciente en un objeto, luego se almacenan en una lista nativa las células infectadas proporcionadas, posteriormente se crea el objeto de lista simple donde almacena el valor de las célula contagiadas y no contagiadas con su respectiva posición en memoria y simulacro de rejilla.

```
def show_menu(options):
    print(Fore.CYAN + 'Bienvenidos al laboratorio de investigación epidemiológica de Guatemala')
    print(Fore.CYAN + '--------------------------------------------------------------------------------')
    print(Fore.CYAN + 'Seleccione una opcion: ')
    for key in sorted(options):
        print(Fore.CYAN + f'{key}) {options[key][0]}')

def read_options(options):
    while (a:= input(Fore.YELLOW + 'Opción: ')) not in options:
        print(Fore.YELLOW + 'Opción incorrecta, vuelva a intentarlo.')
    return a

def execute_option(option,options):
    options[option][1]()

def generate_menu(options,exit_option):
    option = None
    while option != exit_option:
        show_menu(options)
        option = read_options(options)
        execute_option(option,options)
        print()

def main_menu():
    options = {
        '1':('Cargar un archivo',upload_file),
        '2':('Ver pacientes',lambda: select_patient(list_of_patients)),
        '3':('Salir',exit),
    }
    generate_menu(options,'3')
```

Figura 4. Menú principal.

Fuente: elaboración propia, 2022

```
def upload_file():
    tree = ET.parse('entrada.xml')
    patients = tree.getroot()
    #print each patient
    for patient in patients:
        #print(f'Paciente')
        for personal_data in patient.iter('datospersonales'):
            #Print personal_data
            name = personal_data.find("nombre").text
            age = personal_data.find("edad").text

        #Print periods and m of the matrix
        period = patient.find("periodos").text
        m =  int(patient.find("m").text)

        # Create the patient object
        patient_obj = Patient(name,age,period,m)
        patient_obj.name = name
        patient_obj.age = int(age)
        patient_obj.period = int(period)
        patient_obj.matrix = m

        #find the cells
        infected_cells = []
        for rack in patient.iter('rejilla'):
            for cell in rack:
                cell_infected = Cell(int(cell.attrib["f"]),int(cell.attrib["c"]),True,1)
                infected_cells.append(cell_infected)

        #Create the matrix
        patient_obj.matrix_of_cells = List_for_cells()
        patient_obj.matrix_of_cells.create_matrix(m*m)
        #Add the infected cells and the ones who are not infected
        for i in range(m):
            for j in range(m):
                for n in infected_cells:
                    if i == n.x and j == n.y:
                        patient_obj.matrix_of_cells.edit_pos(n,i,j,m)
                    elif patient_obj.matrix_of_cells.get_pos(i,j,m) == None:
                        cell = Cell(i,j,False,0)
                        patient_obj.matrix_of_cells.edit_pos(cell,i,j,m)

        #append the patient to the global navite list
        global list_of_patients
        list_of_patients.append(patient_obj)
```

Figura 5. Función upload_file.

Fuente: elaboración propia, 2022

Modulo: select_and_execute_patient.py

Este modulo es uno de lo que más funciones contiene ya que es lo que le da la funcionalidad total a la aplicación. Como primer punto en la figura 6 genera un submenú para escoger los usuarios cargados y evaluarlos conforme los periodos.

Desde la figura 7.1 hasta 7.4 ejecuta cada matriz del paciente seleccionado, guiándose por el patrón ya establecido para monitorear el comportamiento de las células con el pasar del tiempo.

En la figura 8 se almacenan 2 funciones sencillas que se complementan con las de la figura 7 las cuales tienen la finalidad de contar la cantidad de celdas vecinas infectadas de una célula y retornar un valor e imprimir en consola cada periodo seleccionado de la matriz respectivamente.

Por consiguiente, la figura 9 genera un diagnostico y este usará la información del paciente seleccionado y

el ultimo periodo ejecutado para generar un formato XML con el resultado.

Por último, en la figura 10 contiene la función que genera una imagen png con listas simples en forma de matriz simulando el ultimo periodo ejecutado.

```python
def select_patient(list_of_patients):
    print(Fore.CYAN +'------------------Pacientes-------------------')
    #print the patients of the that were uploaded
    dict_of_patients = {}
    for i in range(len(list_of_patients)):
        print(f' {i+1}) Nombre: {list_of_patients[i].name} Edad: {list_of_patients[i].age}')
        dict_of_patients[i+1] = list_of_patients[i]

    print(Fore.CYAN +'-----------Seleccione un paciente que desea analizar-------------')
    while True:
        print('Ingrese el número del paciente a evaluar \n"0" para regresar al menu principal')
        option = int(input())
        #Select and evaluate

        try:
            if option == 0:
                break
        except ValueError:
            print('Ingrese una opción valida')

        for n in range(len(list_of_patients)+1):
            try:
                if option == n:
                    patient = dict_of_patients[option]
                    print(f'Has seleccionado al paciente -> {patient.name}')
                    execute_patient(patient)
            except ValueError:
                print('Ingrese una opción valida')
```

Figura 6. Función: select_patient.

Fuente: elaboración propia, 2022

```python
def execute_patient(patient):
    #Variables
    list_of_each_period = []
    name = patient.name
    age = patient.age
    period = patient.period
    m = patient.matrix
    c_periods = 0
    print(Fore.YELLOW + '-----------------------------------------------------------')
    print(Fore.YELLOW + f'Paciente: {name}')
    print(Fore.YELLOW + f'Edad: {age}')
    print(Fore.YELLOW + '---------Datos medicos--------')
    print(Fore.YELLOW + f'Periodos a analizar: {period}')
    print(Fore.YELLOW + f'Tamaño de la cuadrilla: {m}x{m}')
    print(Fore.YELLOW + '--------------------Cuadrilla Celular----------------------------')
    #This goint to help us later
```

Figura 7.1. Función execute_patient.

Fuente: elaboración propia, 2022

```python
    while True:
        # Variable
        #Generate a new list
        patient_matrix = List_for_cells()
        patient_matrix.create_matrix(m*m)
        for x in range(m):
            for y in range(m):
                patient_matrix.edit_pos(patient.matrix_of_cells.get_pos(x,y,m).getSymbol(),x,y,m)

        list_of_each_period.append(patient_matrix)
        #another things
        print(f'Periodos analizados -> {c_periods}')
        print_matrix(m,patient)
        print(f'1. Para ejecutar un periodo \n2. Terminar evaluación y realizar diagnostico')
        option = int(input('Ingrese una opción: '))
```

Figura 7.2. Función execute_patient.

Fuente: elaboración propia, 2022

```python
        if option == 1:
            c_periods += 1

            for x in range(m):
                for y in range(m):
                    # (x,y)
                    main_cell = patient.matrix_of_cells.get_pos(x,y,m)
                    # (x-1,y-1)
                    cell_1 = patient.matrix_of_cells.get_pos(x-1,y-1,m)
                    # (x-1,y)
                    cell_2 = patient.matrix_of_cells.get_pos(x-1,y,m)
                    # (x-1,y+1)
                    cell_3 = patient.matrix_of_cells.get_pos(x-1,y+1,m)
                    # (x,y-1)
                    cell_4 = patient.matrix_of_cells.get_pos(x,y-1,m)
                    # (x,y+1)
                    cell_5 = patient.matrix_of_cells.get_pos(x,y+1,m)
                    # (x+1,y-1)
                    cell_6 = patient.matrix_of_cells.get_pos(x+1,y-1,m)
                    # (x+1,y)
                    cell_7 = patient.matrix_of_cells.get_pos(x+1,y,m)
                    # (x+1,y+1)
                    cell_8 = patient.matrix_of_cells.get_pos(x+1,y+1,m)

                    cells = [cell_1,cell_2,cell_3,cell_4,cell_5,cell_6,cell_7,cell_8]
                    num_of_infected_cells = verified(cells)

                    if main_cell.getSymbol() == '▒':
                        if num_of_infected_cells == 3:
                            main_cell.post_infected = 1
                        else:
                            main_cell.post_infected = 0

                    elif main_cell.getSymbol() == '▉':
                        if num_of_infected_cells == 3 or num_of_infected_cells == 2:
                            main_cell.post_infected = 1
                        else:
                            main_cell.post_infected = 0

            for x in range(m):
                for y in range(m):
                    if patient.matrix_of_cells.get_pos(x,y,m).post_infected == 1:
                        patient.matrix_of_cells.get_pos(x,y,m).infected = True
                    else:
                        patient.matrix_of_cells.get_pos(x,y,m).infected = False
```

Figura 7.3. Función execute_patient.

Fuente: elaboración propia, 2022

```python
        elif option == 2:
            print(Fore.CYAN+'----------------------------')
            print('SE HA CREADO UN DIAGNOSTICO!')
            print(Fore.CYAN+'----------------------------')
            print_graphviz(m,patient)
            diagnostic(patient,list_of_each_period,m)
            break
        else:
            print('Ingrese una opción valida!')
```

Figura 7.4. Función execute_patient.

Fuente: elaboración propia, 2022

```python
def verified(list):
    c = 0
    for cell in list:
        if cell is not None:
            if cell.getSymbol() == '▉':
                c +=1
    return c

#This fun print any matrix
def print_matrix(m,patient):
    for i in range(m):
        for j in range(m):
            print('',f'{patient.matrix_of_cells.get_pos(i,j,m).getSymbol()}',end='')
        print('\n')
```

Figura 8. Función verified y print_matrix.

Fuente: elaboración propia, 2022

```python
def xml_response(patient):
    #decorate
    def indent(elem,level=0):
        i = "\n" + level*"  "
        j = "\n" + (level-1)*"  "
        if len(elem):
            if not elem.text or not elem.text.strip():
                elem.text = i + "  "
            if not elem.tail or not elem.tail.strip():
                elem.tail = i
            for subelem in elem:
                indent(subelem, level+1)
            if not elem.tail or not elem.tail.strip():
                elem.tail = j
        else:
            if level and (not elem.tail or not elem.tail.strip()):
                elem.tail = j
        return elem

    patients = ET.Element('pacientes')
    person = ET.SubElement(patients,'paciente')

    personal_data = ET.SubElement(person,'datospersonales')
    name = ET.SubElement(personal_data,'nombre')
    age = ET.SubElement(personal_data,'edad')
    name.text = patient.name
    age.text = str(patient.age)

    period = ET.SubElement(person,'periodos')
    period.text = str(patient.period)

    matrix = ET.SubElement(person,'m')
    matrix.text = str(patient.matrix)

    result = ET.SubElement(person,'resultado')
    result.text = 'Leve'

    n = ET.SubElement(person,'n')
    n.text = '99'

    n1 = ET.SubElement(person,'n1')
    n1.text = '1'

    # EXPORT
    tree = ET.ElementTree(indent(patients))
    tree.write(f'{patient.name}_diagnostico.xml',xml_declaration=True, encoding='utf-8')
```

Figura 9. Función xml_response.

Fuente: elaboración propia, 2022

```python
def print_graphviz(m,patient):
    graph = '''
digraph Patient{
    node [shape=box, nodesep=1, fillcolor=blue, style=filled]
    compound=true
    edge[fontcolor="black" color="#ff5400"]
    subgraph cluster_periodo {
        bgcolor = "#398D9C"
'''
    graph += f'''
        label = "Paciente: {patient.name} - Periodo N.{patient.period}"
    '''
    for x in range(m):
        for y in range(m):
            cell = patient.matrix_of_cells.get_pos(x,y,m)
            graph += f'''
        nodoF{str(x)}_C{str(y)}[label=" ({x},{y})", fillcolor="{cell.getColor()}"];
    '''
    # generate the ranks
    cc = 0
    while m > cc:
        string = ''
        rank = '{rank = same'
        rank_f = '}'
        for x in range(m):
            rank_value = f''';nodoF{str(cc)}_C{str(x)}'''
            string += rank_value
        graph += f'''
        {rank}{string}{rank_f}
        '''
        cc+=1
    # generate the unions betwewen columns
    cc_g = 0
    while m > cc_g:
        string_2 = ''
        for x in range(m):
            rank_value_2 = f'''nodoF{str(cc_g)}_C{str(x)}'''
            string_2 += f'{rank_value_2} -> '
        new_string = string_2[:-4]

        if not cc_g % 2 == 0:
            graph += f'''
        {new_string} [dir=back];
        '''
        else:
            graph += f'''
        {new_string};
        '''
        cc_g+=1
    # generate the unions betwewen rows
    string_3 = ''
    for x in range(m):
        union = f'''nodoF{str(x)}_C{str(0)} -> '''
        string_3 += union
    new_string_2 = string_3[:-4]
    graph += f'''
    {new_string_2};
    '''
    graph += '''
    }
    '''
    graph += '''
}
    '''
    #generate  graphviz
    # compilated
    miArchivo = open(f'graphviz.dot','w')
    miArchivo.write(graph)
    miArchivo.close()
    system(f'dot -Tpng graphviz.dot -o {patient.name}_gráfica.png')
    system(f'cd ./{patient.name}_gráfica.png')
    startfile(f'{patient.name}_gráfica.png')
    print(graph)
```

Figura 10. Función print_grahpviz.

Fuente: elaboración propia, 2022

Conclusiones

El propósito de este proyecto es leer, generar e interpretar de manera correcta archivos con el formato XML, cumplir la necesidad que el programa solicita mediante el uso de todas las herramientas que la programación ofrece como los paradigmas de programación.

Dicho proyecto busca fortalecer los conocimientos de como las estructuras de datos funcionan ya que estas son la base de como se construyen las diferentes listas nativas de un lenguaje de programación y aplicar conceptos matemáticos para poder adaptar una herramienta a cualquier necesidad.
Por último, hacer uso de la herramienta grahpviz para la generación de grafos o en este caso listas enlazadas y sus derivados.

Referencias bibliográficas

The Graphviz Authors. (2022). Documentation. Graphviz. Recuperado 6 de septiembre de 2022, de https://graphviz.org/documentation/

A linked list program using one class. (2020, 19 septiembre). Stack Overflow. Recuperado 6 de septiembre de 2022, de https://stackoverflow.com/questions/63973503/a-linked-list-program-using-one-class

xml.etree.ElementTree — The ElementTree XML API — Python 3.10.6 documentation. (s. f.). Recuperado 6 de septiembre de 2022, de https://docs.python.org/3/library/xml.etree.elementtree.html

Anexos

```xml
<?xml version="1.0" encoding="UTF-8"?>
<pacientes>
   <paciente>
      <datospersonales>
         <nombre>AriGameplays</nombre>
         <edad>24</edad>
      </datospersonales>
      <periodos>12</periodos>
      <m>10</m>
      <rejilla>
         <celda f="0" c="4" />
         <celda f="1" c="4" />
         <celda f="2" c="4" />
         <celda f="1" c="3" />
         <celda f="1" c="5" />
      </rejilla>
   </paciente>
   <paciente>
      <datospersonales>
         <nombre>Auronplay</nombre>
         <edad>33</edad>
      </datospersonales>
      <periodos>12</periodos>
      <m>10</m>
      <rejilla>
         <celda f="3" c="3" />
         <celda f="3" c="6" />
         <celda f="5" c="1" />
         <celda f="5" c="8" />
         <celda f="6" c="2" />
         <celda f="6" c="7" />
         <celda f="7" c="3" />
         <celda f="7" c="4" />
         <celda f="7" c="5" />
         <celda f="7" c="6" />
      </rejilla>
   </paciente>
</pacientes>
```

Figura 11. Documento de entrada en XML.

Fuente: elaboración propia, 2022

```python
from os import startfile, system
import xml.etree.ElementTree as ET
from colorama import Fore
```

Figura 12. librerías utilizadas.

Fuente: elaboración propia, 2022