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.

ViralAPP

201407049 – Henry Alexander García Montúfar

Resumen

A lo largo del tiempo se ha observado, el desarrollo constante de las tecnologías de la información y la comunicación, con un uso frecuente de dispositivos móviles en la población, brindando de esta manera acceso a la información en cualquier instante, en ello se puede incluir los temas en salud. De estas herramientas tecnologías, las aplicaciones digitales establecidas para el área de salud han visto un crecimiento exponencial en el mundo y en Guatemala, siendo estas utilizadas por pacientes y por personal prestador de salud, por esta razón se pretende colaborar con la elaboración de una aplicación que permita poder detectar la evolución en la cual las enfermedades infectan las células del cuerpo humano y como se va propagando o retrocediendo en cada una de las células, con el fin de poder detectar, si cierta patología significa un riesgo o no para la vida, permitiendo dar prioridad a aquellos casos que representen un riesgo elevado.

Palabras clave
Tecnología, patología, salud, infecciones, aplicaciones.

Abstract

Over time, the constant development of information and communication technologies has been observed, with frequent use of mobile devices in the population, thus providing access to information at any time, this can include health issues. Of these technological tools, the digital applications established for the health area have seen an exponential growth in the world and in Guatemala, being used by patients and health care providers, for this reason it is intended to collaborate with the development of an application that allows to detect the evolution in which the diseases infect the cells of the human body and how it spreads or regresses in each of the cells, in order to be able to detect, if a certain pathology means a risk or not for life, allowing priority to be given to those cases that represent a high risk

Keywords
Technology, pathology, health, infections, applications..

Introducción

Aplicación diseñada para poder buscar e identificar patrones, para poder detectar el comportamiento, que tienen los virus al infectar las células del cuerpo, la aplicación, viralAPP, fue diseñada en el lenguaje de programación Python, la cual tiene como columna vertebral una lista enlazada que permite realizar un orden de las celdas ingresadas cargadas con la información genética del virus.

ViralAPP

ViralAPP es una aplicación, destinada al área de salud, la cual consta de 7 pantallas en las cuales, se puede acceder a diferentes módulos que permiten procesar la información ingresada por medio de un archivo XML.

a. Cargar Archivo
b. Procesar Archivo
c. Generar Archivo de Salida
d. Créditos
e. Generar gráfica
f. Salir

De esta manera se distribuye la pantalla de bienvenida de la aplicación la cual representa el menú principal de la misma.

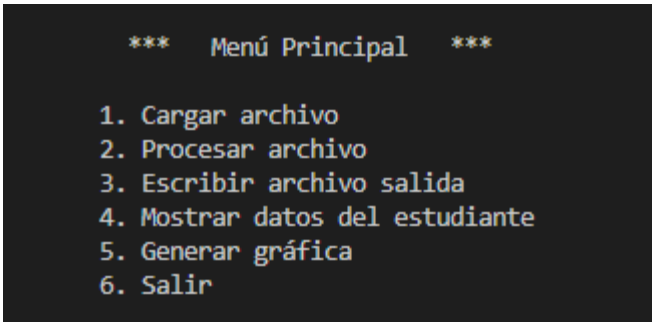

Figura 1. Menú Principal.

Fuente: elaboración propia

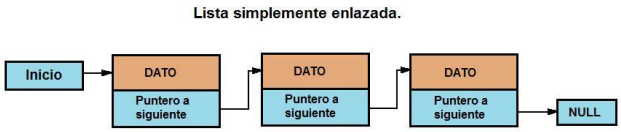

Figura 2. Lista Simple Enlazada.

Fuente: José Pascual 2022

Menú Principal se conforma de una función denominada menú, la cual imprime en pantalla cada uno de los ítems que muestra el menú principal, y permite el acceso a cada uno de las funciones que presentan los ítems iniciales.

```
def menu():
    cls()
    while True:
        print("""\n          ***   Menú Principal   ***\n
        1. Cargar archivo
        2. Procesar archivo
        3. Generar archivo salida
        4. Creditos
        5. Generar gráfica
        6. Salir""")

        opc = input("\nOpción a realizar: ")

        if opc == "1":
            cls()
            print("    --- Cargar Archivo ---    \n")
            try:
                archivo = input("Ingrese el nombre del archivo\n")
                Filename = './' + archivo
                cargar(Filename)
                input("\n--> Archivo cargado exitosamente...")
                cls()
                menu()
            except:
                input("El nombre del archivo no se ha encontrado...")
                menu()
            break
        elif opc == "2":
            cls()
            input("--> Listado de Pacientes agregados: \n")
            ListaPaciente.mostrarPacientes()
            input()
            menu()
```

Figura 3. Menú Principal.

Fuente: elaboración propia

Función cargar

Procesa la carga de un archivo xml para poder mostrar en pantalla su contenido, esta función se compone de una ruta de lectura y varios ítems que permiten ingresar a cada parte del texto del xml.

```
def cargar(ruta):
    tree = ET.parse(ruta)
    root = tree.getroot()
    for elemento in root:
        print("Paciente: ", elemento.attrib["nombre"], "ha sido insertado")
        #Almaceno el nombre del nodo
        nombre = elemento.attrib["nombre"]

        #Almaceno la datos personales
        for datoss in elemento.iter("datospersonales"):
            print("Datos del Paciente")
            for columna in datoss.iter("nombre"):
                tamM = columna.text
                print("Nombre del paciente: ", tamM)
            for fila in datoss.iter("edad"):
                tamN = fila.text
                print("Edad del paciente: ", tamN)

        #Almaceno la posicion de inicio del nodo
        for subelemento in elemento.iter("posicioninicio"):

            for inicioX in subelemento.iter("x"):
                xInicio = inicioX.text

            for inicioY in subelemento.iter("y"):
                yInicio = inicioY.text
```

Figura 3. Menú Principal.

Fuente: elaboración propia

Lista paciente

Ordena cada uno de los nodos que entran al programa por medio de un xml, permitiendo generar de esta manera un nuevo documento.

```
from nodoPaciente import Nodo

class ListaPaciente():

    def __init__(self):
        self.inicio = None
        self.fin = None
        self.size = 0

    def vacia(self):
        return self.inicio == None

    def agregarPaciente(self, nombre, columna, fila, inicioX, inicioY, finX, finY):
        Paciente = Nodo(nombre, columna, fila, inicioX, inicioY, finX, finY)
        self.size += 1

        if self.vacia():
            self.inicio = self.fin = Paciente
        else:
            aux = self.fin
            self.fin = aux.siguiente = Paciente

    def mostrarPacientes(self):
        aux = self.inicio

        while aux != None:
            print("Nombre: ", aux.nombre)
            print("Dimensiones del Paciente: ", aux.columna, "x", aux.fila)
            print("Coordenada de Inicio --> Fila/Columna: ", "(", aux.inicioX, ",", aux.inicioY , ")")
            print("Coordenada de Fin --> Fila/Columna: ", "(", aux.finX, ",", aux.finY , ")")
            print("\n")
            aux = aux.siguiente

    def buscarPaciente(self, nombre):
```

Figura 4. Lista enlazada.

Fuente: elaboración propia

La aplicación se compone de diferentes nodos los cuales se encuentran en una lista enlazada simple, permitiendo el ordenamiento de los diferentes datos ingresados al sistema por medio del archivo fuente.

Cada elemento de la lista contiene uno o varios campos con los datos y un puntero apuntando a la dirección donde está ubicado el siguiente nodo. Cada objeto de la lista se llama nodo y contiene como se ha dicho los datos y un puntero al siguiente elemento.
La lista comienza desde la pila con un puntero apuntando al primer nodo, después el primer nodo tiene un puntero apuntando al segundo nodo y así sucesivamente hasta que el puntero del último nodo contiene null.

Lista enlazada

Una lista simplemente enlazada pertenece a las estructuras de datos fundamentales. Suele utilizarse para implementar otras estructuras de datos.
Está estructurada en una secuencia de nodos, en los que se guardan los datos y un puntero que apunta (contiene la dirección de la ubicación) al siguiente nodo.
La principal utilidad de la lista enlazada es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento físico en memoria. De este modo se permite que el orden de lectura de la lista sea diferente al orden de almacenamiento físico. Al contrario de un array, el acceso a cada elemento no se hace a través de un índice sino mediante un puntero. Otra diferencia con los arrays es que estos pueden ser accedidos de forma aleatoria mientras que la lista se recorre de forma ordenada.
La lista también permite añadir o eliminar nodos en cualquier lugar, aunque no permite un acceso aleatorio. Cuando es necesario hacer varias operaciones de inserción y eliminación de elementos en un conjunto resulta conveniente utilizar listas

enlazadas. El puntero del último nodo contiene un valor vacío Null.

Nodo

Un nodo es una variable de un tipo dado, entero, cadena, etc almacenada en la memoria. En este caso es una variable.

Puntero

Un puntero es un dato que contiene la dirección del siguiente nodo. En este caso es un objeto de tipo nodo que apunta al siguiente nodo llamado siguiente.

Insertar un elemento en la lista

Para insertar un nuevo elemento en la lista se sigue la siguiente secuencia:
-Declaración del elemento a insertar.
-Asignación de la memoria para el nuevo elemento
- Insertar el contenido en los datos.
-Actualizar los punteros que apuntan al primer y último elemento.
-Actualizar el tamaño de la lista.

Además, se presentan varios casos a la hora de insertar un elemento en una lista:
- 1. Inserción en una lista vacía.
- 2. Inserción al comienzo de la lista.
- 3. Inserción al final de la lista.
- 4. Inserción en otra parte de la lista.

Inserción en una lista vacía

Las etapas para insertar un elemento en una lista vacía son las siguientes:
-Reserva de memoria para el nuevo elemento.
-Introducir el nuevo elemento en memoria.
-Como es una lista vacía el puntero hacia el siguiente elemento apuntará a Null.
-Los punteros inicio y fin apuntaran hacia el nuevo elemento.

- Se actualiza la variable que contiene el tamaño de la lista.

Conclusiones

La lista enlazada permite tener un manejo optimo de los datos por medio del procesamiento de los mismos.

Las listas enlazadas contienen nodos los cuales se complementan con un apuntador que apunta al siguiente, al eliminar el apuntador se pierde la comunicación con el nodo y este deja de ser ubicado en la lista.

Se pueden insertar nuevos nodos a listas ya existentes.

Anexos

Nodo

```python
#Clase TDA del nodo donde se tendra el valor
class NodoPaciente():
    def __init__(self, fila, columna, valor):
        self.fila = fila
        self.columna = columna
        self.valor = valor
        self.derecha = None
        self.izquierda = None
        self.arriba = None
        self.abajo = None


#Clase TDA del nodo donde se tendran los encabezados
class NodoEncabezado():
    def __init__(self, id):
        self.id = id
        self.siguiente = None
        self.anterior = None
        self.acceso = None
```

Figura 5. Nodo.

Fuente: elaboración propia

Nodo paciente

```python
from matriz import Matriz
class Nodo():

    def __init__(self, nombre, columna, fila, inicioX, inicioY, finX, finY):
        self.nombre = nombre
        self.columna = columna
        self.fila = fila
        self.inicioX = inicioX
        self.inicioY = inicioY
        self.finX = finX
        self.finY = finY
        self.matriz = Matriz()
        #Aqui instanciamos la matriz donde va el Paciente
        self.siguiente = None

    def getPaciente(self):
        return self.matriz
```

Figura 6. Nodo Paciente.

Fuente: elaboración propia

XML entrada

```xml
<pacientes>
    <paciente nombre="1">
        <datospersonales>
            <nombre>Nombre </nombre>
            <edad>3</edad>
        </datospersonales>
        <posicioninicio>
            <x>1</x>
            <y>1</y>
        </posicioninicio>
        <posicionfin>
            <x>3</x>
            <y>3</y>
        </posicionfin>
        <celda f="1" c="1"></celda>
        <celda f="1" c="2"></celda>
        <celda f="1" c="3"></celda>
        <celda f="2" c="1"></celda>
        <celda f="2" c="2"></celda>
        <celda f="2" c="3"></celda>
        <celda f="3" c="1"></celda>
        <celda f="3" c="2"></celda>
        <celda f="3" c="7"></celda>
    </paciente>
</pacientes>
```

Figura 7. XML.

Fuente: elaboración propia

Referencias bibliográficas

1. J. Pascual, (2022). *Analysis programacion*
2. Anonimo.Wikipedia https://en.wikipedia.org/wiki/Linked_list