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.

"SIMULADOR DE CRECIMEINTO DE BACTERIAS EN UNA MUESTRA"

202110180 – Juan Carlos Gonzalez Valdez

Resumen

Dicho programa es un simulador de colonias bacterianas, en el que el usuario puede agregar datos de manera manual o con carga de archivos de usuario.

La idea de este mismo es poder saber el funcionamiento general de una bacteria, a la hora de sobrevivir y multiplicarse en otras.

En el mismo podemos mostrar datos en una pequeña simulación y como estos podrían actuar en la vida real, esto es una demostración ficticia para poder implementar la idea de estos, espero logren disfrutarla y entender su funcionamiento.

Palabras clave
Tabla, Organismos, Adyacente, XML, Celda.

Abstract

This program is a bacterial colony simulator, in which the user can add data manually or by uploading user files.

The idea of this program is to know the general functioning of a bacterium, at the time of surviving and multiplying in others.

In it we can show data in a small simulation and how they could act in real life, this is a fictitious demonstration to implement the idea of these, I hope you enjoy it and understand its operation.

Keywords
Table, Organism, Adyacent, XML, Cells.

Introducción

Este programa es un simulador de colonias de bacterias. El objetivo es modelar el crecimiento y evolución de las bacterias en una placa de petri. El usuario puede cargar diferentes tipos de organismos y muestras de bacterias desde un archivo XML o agregar organismos manualmente a una muestra seleccionada. La simulación se ejecuta por generaciones, en cada una de las cuales se aplican reglas para determinar el crecimiento y la muerte de las bacterias en la muestra. El estado de la muestra se presenta visualmente en una tabla que se actualiza después de cada generación. Además, se genera un archivo PDF que muestra la evolución de la muestra a lo largo de las generaciones.

Desarrollo del tema

Este programa es un simulador de crecimiento de organismos en una muestra. Se trata de un sistema que permite simular cómo crece una población de organismos en una muestra de laboratorio.

El programa permite cargar una muestra desde un archivo XML y visualizarla en una tabla. Además, el usuario puede agregar organismos manualmente a la muestra y simular su crecimiento durante varias generaciones. Cada organismo se representa por un código y un estado (vivo o muerto).

El programa utiliza técnicas de programación orientada a objetos y estructuras de datos como diccionarios y listas para almacenar y manipular los datos de las muestras y organismos. También utiliza la biblioteca xml.etree.ElementTree para cargar los datos desde archivos XML y la biblioteca os para generar y guardar archivos PDF con la visualización de la muestra.

a. Generación de una tabla en formato DOT y su posterior conversión a un archivo PDF que se utiliza para visualizar la muestra.
b. Carga de datos de un archivo XML y creación de una lista de organismos y una lista de muestras.
c. Agregar un organismo manualmente a la muestra en una posición específica dentro de la tabla.
d. Función para obtener las celdas adyacentes a una celda específica.

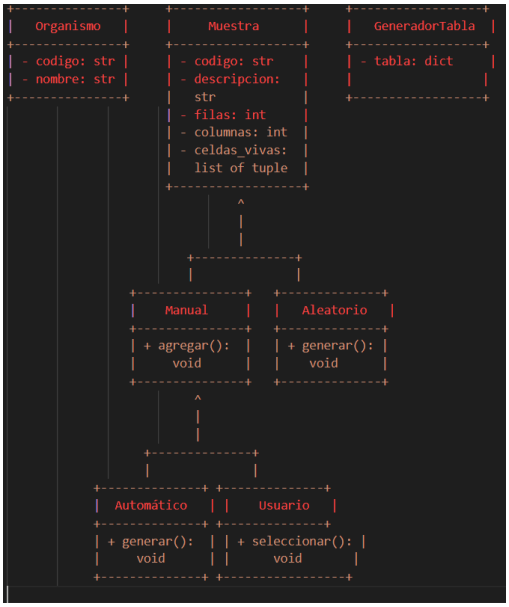


Figura 1. Diagrama de clases.

Fuente: elaboración propia/2023/Juan Gonzalez.

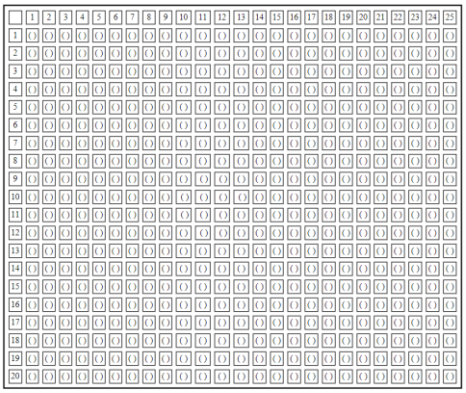


Figura 2. Tabla exportada.

Fuente: elaboración propia/2023/Juan Gonzalez.

Tabla I.

Variables y funciones utilizadas para la simulación de una colonia bacteriana en Python.

CATEGORÍA	CATEGORÍA
tabla	Diccionario
muestra_seleccionada	Diccionario
import os	módulo
xml.etree.ElementTree	módulo
generar_tabla	función
cargar_XML	función
agregar_organismo_manual	función
adyacentes	función
menú	función

Fuente: elaboración propia/2023/Juan Gonzalez.

tabla: diccionario que almacena la información de los organismos presentes en la muestra.
muestra_seleccionada: diccionario que contiene información sobre el tamaño de la muestra seleccionada para mostrar en la tabla.
import os: módulo para interactuar con el sistema operativo.
xml.etree.ElementTree: módulo para trabajar con XML...
generar_tabla: función que genera una tabla con los organismos presentes en la muestra seleccionada y la guarda en un archivo PDF...
cargar_XML: función que carga un archivo XML y devuelve una lista de organismos y una lista de muestras.
agregar_organismo_manual: función que permite agregar un organismo de manera manual a la muestra seleccionada.
adyacentes: función que devuelve una lista con las celdas adyacentes a una celda dada.
menú: función principal del programa.

Conclusiones

En este programa se presenta un menú que permite al usuario cargar organismos desde un archivo XML, agregar organismos manualmente, ejecutar una simulación de crecimiento de bacterias y guardar el resultado en un archivo PDF.

Para la simulación, se utiliza una tabla que representa una muestra de cultivo de bacterias. Cada celda de la tabla puede estar vacía o contener un organismo. Cada organismo tiene un código que lo identifica y un estado que indica si está vivo o muerto.

El programa utiliza la librería xmEn este programa se presenta un menú que permite al usuario cargar organismos desde un archivo XML, agregar organismos manualmente, ejecutar una simulación de crecimiento de bacterias y guardar el resultado en un archivo PDF.

Referencias bibliográficas

Máximo 5 referencias en orden alfabético.

Beazley, D. (2009). Python Essential Reference (4th ed.). Addison-Wesley Professional.

Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow.

McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (2nd ed.). O'Reilly Media.

Rossum, G. V. (1995). Python tutorial (Versión 1.5). Python Software Foundation.

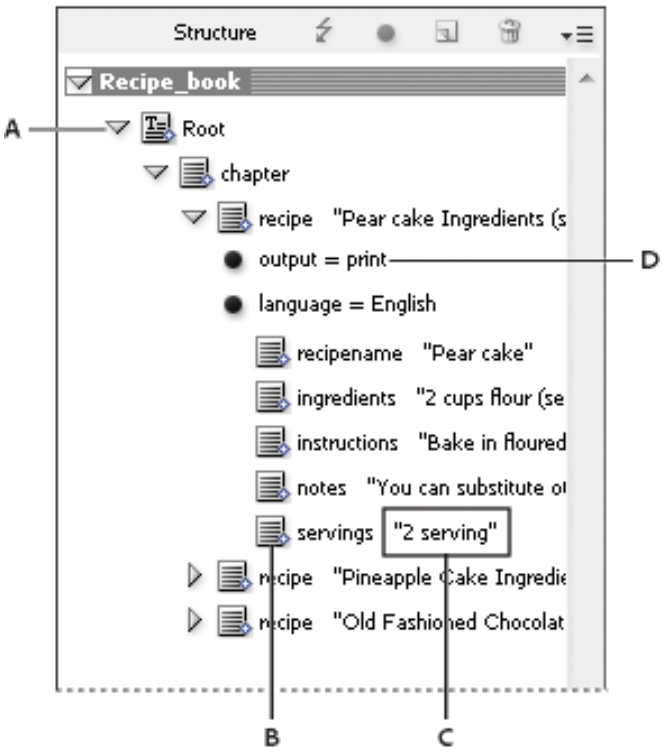
Sweigart, A. (2019). Automate the Boring Stuff with Python: Practical Programming for Total Beginners (2nd ed.). No Starch Press.

Ejemplo sobre el xml:

En la imagen de la derecha aplicamos un ejemplo en el que el elemento raíz es <experimento> y contiene dos elementos principales: <organismos> y <muestras>. El elemento <organismos> contiene una lista de organismos, cada uno de los cuales tiene un nombre, una descripción y una especie. El elemento <muestras> contiene una lista de muestras, cada una de las cuales tiene un nombre, una descripción, un organismo asociado, un tipo de muestra, una fecha y una ubicación.

Cada muestra está vinculada a un organismo específico mediante el elemento <organismo> que utiliza el nombre del organismo como valor. En este ejemplo, la muestra "Muestra 1" está vinculada al organismo "Humano" y la muestra "Muestra 2" está vinculada al organismo "Ratón"

```xml
<experimento>
  <organismos>
    <organismo>
      <nombre>Humano</nombre>
      <descripcion>Organismo humano</descripcion>
      <especie>Homo sapiens</especie>
    </organismo>
    <organismo>
      <nombre>Ratón</nombre>
      <descripcion>Organismo de ratón</descripcion>
      <especie>Mus musculus</especie>
    </organismo>
  </organismos>
  <muestras>
    <muestra>
      <nombre>Muestra 1</nombre>
      <descripcion>Primera muestra</descripcion>
      <organismo>Humano</organismo>
      <tipo>Tumor</tipo>
      <fecha>2022-01-01</fecha>
      <ubicacion>Frío</ubicacion>
    </muestra>
    <muestra>
      <nombre>Muestra 2</nombre>
      <descripcion>Segunda muestra</descripcion>
      <organismo>Ratón</organismo>
      <tipo>Tejido</tipo>
      <fecha>2022-02-01</fecha>
      <ubicacion>Temperatura ambiente</ubicacion>
    </muestra>
  </muestras>
</experimento>
```



En base al mismo tipo de ejemplo el cómo podemos generar un PDF y DOT en base a la función generar_tabla.

Ejemplo de cómo generar un pdf:

```python
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.lib.units import inch
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle

def generar_tabla(datos):
    # Crear archivo PDF
    doc = SimpleDocTemplate("tabla.pdf", pagesize=letter)

    # Crear tabla
    tabla = Table(datos)

    # Configurar estilo de tabla
    estilo_tabla = TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, 0), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, 0), 14),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('TEXTCOLOR', (0, 1), (-1, -1), colors.black),
        ('ALIGN', (0, 1), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 1), (-1, -1), 'Helvetica'),
        ('FONTSIZE', (0, 1), (-1, -1), 12),
        ('BOTTOMPADDING', (0, 1), (-1, -1), 6),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
    ])

    tabla.setStyle(estilo_tabla)

    # Crear contenido del archivo PDF
    contenido = []
    contenido.append(tabla)

    # Agregar contenido al archivo PDF
    doc.build(contenido)
```

En este ejemplo, se utiliza la biblioteca ReportLab para generar un archivo PDF que contiene una tabla con los datos proporcionados en la variable datos. Primero, se crea un objeto SimpleDocTemplate que representa el archivo PDF y se especifica su tamaño de página (letter). Luego, se crea la tabla a partir de la variable datos y se configura su estilo utilizando un objeto TableStyle. Finalmente, se crea una lista de contenido que contiene la tabla y se agrega al archivo PDF utilizando el método build.

En este lado estaría mostrando un ejemplo con la misma variable, pero sería para generar un archivo DOT.

Ejemplo de cómo generar un DOT:

```python
def generar_tabla(organismos, muestras, archivo_dot):
    with open(archivo_dot, 'w') as archivo:
        archivo.write('digraph {\n')
        archivo.write('node [shape=plaintext]\n')
        archivo.write('tabla [\n')
        archivo.write('label=<\n')
        archivo.write('<table border="0" cellspacing="0" cellborder="1">\n')
        archivo.write('<tr><td><b>Organismo</b></td><td><b>Muestras</b></td></tr>\n')

        for organismo in organismos:
            archivo.write(f'<tr><td rowspan="{len(muestras[organismo]) + 1}">{organismo}</td></tr>\n')

            for muestra in muestras[organismo]:
                archivo.write(f'<tr><td>{muestra}</td></tr>\n')

        archivo.write('</table>\n')
        archivo.write('>];\n')
        archivo.write('}\n')
```

En esta función, primero abrimos un archivo DOT en modo escritura utilizando el archivo pasado como argumento. Luego, escribimos las primeras líneas necesarias para generar un archivo DOT válido: la declaración digraph y el estilo del nodo que usaremos para la tabla.

Luego, escribimos el contenido de la tabla. Primero, escribimos la etiqueta de apertura de la tabla, y luego escribimos una fila de encabezado que contiene los nombres de columna "Organismo" y "Muestras".

Este archivo DOT generado puede ser utilizado con la herramienta Graphviz para generar un gráfico visual de la tabla. Por ejemplo, si la herramienta Graphviz está instalada en el sistema, se puede generar un archivo PDF a partir de este archivo DOT ejecutando el siguiente comando en la línea de comandos:
dot -Tpdf archivo.dot -o archivo.pdf
Esto genera un pdf llamado archivo.pdf.

Agregando a las cargas, hare un listado de excepciones con cada función y descripción:

- Función cargar_organismos_xml():
- Excepción FileNotFoundError: Si el archivo XML de entrada no se encuentra en la ubicación especificada, se lanza esta excepción y se muestra un mensaje de error indicando que el archivo no se encuentra.

- Excepción etree.ElementTree.ParseError: Si hay un error al analizar el archivo XML de entrada, se lanza esta excepción y se muestra un mensaje de error indicando que el archivo XML no se puede analizar.

- Función cargar_muestras_xml():
  Excepción FileNotFoundError: Si el archivo XML de entrada no se encuentra en la ubicación especificada, se lanza esta excepción y se muestra un mensaje de error indicando que el archivo no se encuentra.

- Excepción etree.ElementTree.ParseError: Si hay un error al analizar el archivo XML de entrada, se lanza esta excepción y se muestra un mensaje de error indicando que el archivo XML no se puede analizar.

- Función generar_tabla():
  Excepción FileNotFoundError: Si la plantilla HTML para generar la tabla no se encuentra en la ubicación especificada, se lanza esta excepción y se muestra un mensaje de error indicando que la plantilla no se encuentra.

- Excepción TypeError: Si los datos de entrada no son del tipo esperado, se lanza esta excepción y se muestra un mensaje de error indicando que los datos de entrada no son válidos.

- Excepción ValueError: Si los datos de entrada tienen un valor incorrecto, se lanza esta excepción y se muestra un mensaje de error indicando que los datos de entrada no son válidos.

- Función generar_grafo():
  Excepción FileNotFoundError: Si la plantilla HTML para generar el grafo no se encuentra en la ubicación especificada, se lanza esta excepción y se muestra un mensaje de error indicando que la plantilla no se encuentra.

- Excepción TypeError: Si los datos de entrada no son del tipo esperado, se lanza esta excepción y se muestra un mensaje de error indicando que los datos de entrada no son válidos.

- Excepción ValueError: Si los datos de entrada tienen un valor incorrecto, se lanza esta excepción y se muestra un mensaje de error indicando que los datos de entrada no son válidos.

- Excepción subprocess.CalledProcessError: Si hay un error al llamar al comando dot para generar el grafo, se lanza esta excepción y se muestra un mensaje de error indicando que el grafo no se puede generar.

- En todas las excepciones, se muestra un mensaje de error indicando la causa del problema y se detiene la ejecución del programa.

Para complementar daré una descripción a detalle de como hice la implementación de adyacente en el Código:

La función adyacente se encarga de determinar las celdas adyacentes a una celda dada en una matriz y retorna una lista con las coordenadas de estas celdas. Además, esta función toma en cuenta los organismos agregados manualmente en la matriz.

Para determinar las celdas adyacentes, se utiliza un enfoque de vecindario de Von Neumann, lo que significa que se consideran solo las celdas que están directamente arriba, abajo, a la izquierda y a la derecha de la celda dada. Las coordenadas de estas celdas se calculan sumando o restando 1 a las coordenadas de la celda dada. Por ejemplo, si la celda dada tiene coordenadas (2, 3), las celdas adyacentes serían (1, 3), (3, 3), (2, 2) y (2, 4).

Sin embargo, para asegurarse de que no se accedan a coordenadas fuera del rango de la matriz, se verifican las coordenadas calculadas y se omiten las que estén fuera de rango.

Además, si en una celda adyacente hay un organismo agregado manualmente, también se agrega a la lista de celdas adyacentes. Para esto, se verifica si la celda adyacente está en el diccionario organismos_manuales que contiene las coordenadas de las celdas con organismos agregados manualmente. Si la celda adyacente está en el diccionario, se agrega a la lista de celdas adyacentes.

Finalmente, la función retorna la lista de celdas adyacentes encontradas. Si no se encontraron celdas adyacentes, la función retorna una lista vacía.

En base a la complementación del Código describiré la utilización de generar_tabla y como este esta con el diagrama de graphviz:

Dicha función es responsable de crear una tabla que muestra la distribución de organismos en las diferentes celdas. También utiliza Graphviz para generar un diagrama que representa la misma información en un formato visualmente atractivo.

La función comienza creando una tabla vacía con celdas vacías para cada posición en la cuadrícula. A continuación, recorre la lista de organismos para agregarlos a la tabla, determinando su posición en la cuadrícula. Para cada organismo, se busca la celda más cercana en la tabla y se le asigna ese organismo. Si la celda ya está ocupada, se lanza una excepción.

La función también permite agregar organismos manualmente a celdas específicas. Esto se hace mediante la especificación de las coordenadas de la celda y el organismo que se va a agregar.

Una vez que se ha completado la tabla, se utiliza Graphviz para generar un diagrama que representa la misma información en un formato visualmente atractivo. Se crea un objeto Digraph de Graphviz y se agregan nodos para cada celda de la tabla. A continuación, se recorre la tabla y se agregan bordes entre las celdas adyacentes que contienen organismos.

Finalmente, se utiliza el método render del objeto Digraph para generar el archivo PDF con el diagrama. El archivo resultante se guarda en la ruta especificada en la variable archivo_salida.