Actividad 05 - Avalancha

UNSAM Programa

Verano 2020

El objetivo de esta actividad será realizar un programa en `Python` que *simule* un proceso de avalancha. **¡¡Recuerden enviar por correo la tarea!!**

El mundo donde modelaremos el fenómeno de una avalancha será el de un cuadrado (dos dimensiones) con un pequeño borde. Este cuadrado va a representar un valle donde cada sector será identificado con una coordenada. Esto lo representaremos en un tablero en cuyos bordes pondremos el valor `-1` para que sea fácil identificarlos. Entonces, si queremos que nuestro valle tenga **tamaño utilizable** de $n \times n$, con n filas y n columnas, agregaremos dos filas y dos columnas en la construcción de la matriz para incluir los bordes.

Para esta actividad, vamos a seguir utilizando el módulo `NumPy` para trabajar con matrices, y con el módulo `PyPlot` para hacer gráficos. Recordar que hay que importalos para poder usarlos:

```
import numpy as np
import matplotlib.pyplot as plt
```

Una vez importados, se puede hacer referencia a ellos directamente con sus **nombres cortos** `np` y `plt` respectivamente.

1. Vamos a definir nuestro *tablero* conteniendo ceros por medio de un `array`. Por ejemplo, para un valle de tamaño $n = 6$, armamos un tablero de 8×8 posiciones ocupadas con ceros. Luego, hacemos que adquiera forma de matriz (todo en el mismo comando, es muy práctico):

   ```
   t = np.repeat(0,8*8).reshape(8,8)
   ```

   Una vez creado, podemos referirnos a una posición del tablero haciendo `t[(i,j)]`, donde i es la fila y j es la columna (comenzando de 0 como en las listas). Entonces, podemos probar los siguientes comandos:

- `print(t[(1,1)])` que deberá imprimir un 0
- `t[(2,3)] = -1`, que pondrá un -1 en $(2, 3)$ de `t` (fila 2, columna 3), ¿Cómo lo verificamos?

> **Importante:** las coordenadas se escriben entre paréntesis, no confundir con las listas.

- También es posible preguntar las dimensiones de `t`: El valor `t.shape[0]` es su cantidad de filas y `t.shape[1]` es su cantidad de columnas.

2. Vamos a identificar los bordes del tablero asignándoles el valor `-1`. Escriba las instrucciones para que marque todos los bordes de `t` con un `-1`.

3. Implemente una función `crear_tablero(n)` que reciba como entrada la cantidad `n` de filas y columnas del valle, y devuelva un tablero vacío (0 en todas posiciones utilizables) y el valor -1 en los *bordes*.

4. Crear un tablero que contenga un valle de 7×7 posiciones utilizables y guardarlo en una variable llamada `t1` (usar la función definida en el punto anterior).

5. Implementar la función `es_borde(tablero, coord)` que reciba un tablero y una posición, y devuelva `True` si dicha posición es un borde y `False` si no.

6. Verificar que:

   - `es_borde(t1, (0, 1))`, nos debe dar como respuesta: `True`.
   - `es_borde(t1, (8, 6))`, nos debe dar como respuesta: `True`.
   - `es_borde(t1, (5, 6))`, nos debe dar como respuesta: `False`.

7. A la hora de divertirnos, ¿Qué mejor que poder poder comenzar una avalancha? Agregaremos funciones que nos permitirán recrear esta dinámica. Empezaremos con una función que permirtirá soltar un copo de nieve en una posición del valle.

   Implementar una función `tirar_copo(tablero, coord)` que tome como entrada un tablero, la coordenada `(i,j)` de la posición donde cae, y agregue un copo en ese lugar (es decir, incremente la cantidad de copos en esa posición).

8. Cuando una posición tenga demasiados copos de nieve, vamos a querer desbordarlo hacia sus *vecinos*. Para ello, necesitamos identificar quiénes son. Los vecinos de una posición son las cuatro posiciones que están arriba, abajo, a la derecha y a la izquierda de la misma, siempre y cuando sean posiciones válidas dentro del tablero.

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	-1
-1	4	0	0	0	0	0	0	-1
-1	0	0	0	3	0	0	0	-1
-1	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Tabla 1: Ejemplo de tablero: los bordes los marcamos con `-1`. Dos posiciones tienen *nieve*, el resto está vacío.

Mirar el tablero 1 y responder las siguientes preguntas:

a) La mayoría del tablero está vacío, salvo 2 coordenadas que están en 3 y 4, ¿Cuáles son esas coordenadas?

b) ¿Cuántos vecinos tienen dichas coordenada? Listar las coordenadas de todos ellos.

9. Implementar la función `vecinos_de(tablero, coord)` que reciba una coordenada y devuelva una lista de sus posiciones vecinas en el tablero que se pasa como parámetro.

   Sugerencia: Para cada posible vecino, agregarlo o no a la lista de vecinos según si es un borde o no.

10. Probar la función `vecinos_de(tablero, coord)` con los siguientes casos:

    - `vecinos_de(t1, (1, 4))`, nos debe dar como respuesta: `[(2, 4), (1, 5), (1, 3)]`.
    - `vecinos_de(t1, (2, 5))`, nos debe dar como respuesta: `[(3, 5), (1, 5), (2, 6), (2, 4)]`.
    - `vecinos_de(t1, (7, 7))`, nos debe dar como respuesta: `[(6,7), (7,6)]`.

11. Defina `desbordar_posicion(tablero, coord)`, que dado un tablero y una coordenada se fija cuántos granitos hay en ese lugar. Si hay cuatro copos o más, entonces debe desbordar, es decir, se deben descontar cuatro copos de dicha posición y repartir uno a cada uno de sus vecinos. Si la posición tiene menos de cuatro vecinos, los granitos sobrantes se caen por el borde y se pierden.

12. `desbordar_nieve(tablero)`: Esta función deberá recorrer todo el tablero, desbordando aquellas posiciones que lo requieran. Vamos a hacer este recorrido yendo fila por fila y deteniéndonos en cada posición, recordando que no hay que procesar los bordes. Si queremos recorrer el tablero `t1`, el esquema sería:

```
cantidad_filas = t1.shape[0]
cantidad_columnas = t1.shape[1]
for i in range(1, cantidad_filas - 1):
    for j in range(1, cantidad_columnas - 1):
        ___COMPLETAR___
```

13. Definir `hay_que_desbordar(tablero)`, que devuelve `True` si hay alguna posición en el tablero que hay que desbordar, y `False` si no.

14. Al agregar un copo, puede pasar que haya que desbordar esa posición. Y eso puede desencadenar otros desbordes en sus posiciones vecinas. Implementar la función `estabilizar(tablero)`, que, mientras haya alguna posición que tenga al menos cuatro granitos, llame a `desbordar_nieve`. Pista:

```
while (hay_que_desbordar(t)):
  desbordar_nieve(t)
```

15. Una vez que se tiene la dinámica de desborde, podemos implementar la función `paso(tablero)`, que suelta un copos de nieve en el centro del tablero y realiza todos los desbordes necesarios.

16. Vamos a guardar los sucesivos estados del tablero en una lista de imágenes, para poder realizar una animación. Para esto necesitamos un paquete de `MatPlotLib` llamado `animation`. Podemos construir la animación haciendo:

```
#agregar animation a los imports
import matplotlib.animation as animation

#algunos parametros de ejemplo
n=5
cant_iteraciones = 20

t = crear_tablero(n)
ims = []
fig = plt.figure()
for i in range(cant_iteraciones):
  paso(t)
  im = plt.imshow(t, animated=True)
  ims.append([im])
```

```
ani = animation.ArtistAnimation(fig, ims, interval=50, blit=True,
    repeat_delay=400)
print("Listo para guardar animacion")
ani.save('dynamic_images.mp4')
plt.show()
```

17. **Optativo** El tablero que consideramos empezó estando vacío. ¿Qué habría pasado si ya hubiera tenido nieve desde antes? Implementar la función `generar_tablero_aleatorio(n, cant_granitos)`, que crea un tablero como `crear_tablero` y le agrega `cant_granitos` copos de nieve en posiciones aleatorias.

18. **Optativo+** En nuestro modelo, los copos de nieve siempre caen en la posición central del tablero. ¿Qué pasaría si pudieran caer en otros lugares? Modificar la función `tirar_granito` para que suelte un copos de nieve en una posición al azar del tablero.

19. **Optativo++** En nuestro modelo, hemos fijado que la capacidad de copos que puede almacenar cada posición es fija y vale cuatro, ¿Cómo podríamos hacer para que esa cantidad fuera distinta en cada posición del tablero?

20. **Optativo+++** Hasta ahora, al desbordar lo hacíamos de manera fija hacia los cuatro costados. ¿Y si lo hacemos de manera aleatoria hacia algo de los ocho vecinos (incluyendo los lugares que están en diagonal)? La regla seía fijar una probabilidad p de desbordar hacia alguno de los vecinos y luego determinar también aleatoriamente cuántos copos hay que desbordar. Proponga qué modificaciones tendría que realizar al programa para poder implementar esta mecánica y, si acepta esta misión, impleméntelas.