UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES
DEPARTAMENTO DE COMPUTACIÓN Y SIMULACIÓN DE SISTEMAS
ESTRUCTURAS DISCRETAS PARA LA COMPUTACIÓN
LABORATORIO 3

Facilitador: Tomás J. Concepción Miranda

Indicaciones

Se debe realizar un informe de laboratorio, en el que se detalle, para cada problema, el desarrollo de la solución (no solo la respuesta). **Envíe su informe en Moodle**, **en formato PDF**, **así como el fuente del programa** en el bloque correspondiente **antes de las 11:55 del 11 de junio del 2023**.

Enunciados

Los siguientes problemas hacen uso del lenguaje Python. Los archivos `lab3.py` y `lab3_funciones.py` se encuentra junto a estas instrucciones en el eCampus. El archivo `lab3_funciones.py` contiene funciones se deben completar según el enunciado de los problemas. `lab3.py` sirve como un bosquejo para el desarrollo del laboratorio. Este script importa las funciones de `lab3_funciones.py`, es decir, que `lab3.py` puede usar las funciones escritas en `lab3_funciones.py`.

Las tuplas son un tipo de dato integrado en Python[1]. Estas se pueden crear usando parentesis (`()`), y colocando valores dentro de los parentesis, separados por coma. Una 2-tupla (una tupla de dos elementos), entonces, puede ser creada al poner el primer valor, seguido de una coma, y luego un segundo valor, todo dentro de parentesis.

Problema 1: Asigne las siguientes tuplas a diferentes variables e imprimalas en pantalla: (*5 puntos*)

a) $(1, 2)$ b) $(metal, hierro)$ c) $(arbol, hojas, 7)$
d) $({}_5P_2, 5^2)$ e) $(\%, \$)$

Problema 2: Complete la función `verf_tup`. Esta acepta como argumento dos tuplas, y retorna `True` si ambas tuplas son iguales, de lo contrario retorna `False`. (*5 puntos*)

[1] `https://docs.python.org/es/3/library/stdtypes.html#typesseq`

Problema 3: Asigne los siguientes conjuntos de tuplas a variables e imprimalas en pantalla: (*5 puntos*)

a) $\{(67, 20), (4, 4, 5), (24, 84, 75)\}$

b) $\{(papel, roca), (tijeras, papel), (roca, tijeras)\}$

c) $\{(98, 89), (a, b), (@, \#)\}$

d) $\{(35, agua), (suelo, 942, ***), (ropa)\}$

e) $\{(1, 2), ((20, 50), \{1, 4, 8\}), (8, 8)\}$

Para los siguientes problemas, $A = \{1, 2, 3, 4, 5, 6, 7\}$. Asigne estos conjuntos como variables dentro de su programa.

La función `product`[2] de la librería `itertools` permite obtener el producto cartesiano de varios iterables como un generador. Por ejemplo, `itertools.prodcut([1,2], ["g","u"])` retorna un generador para $(1, g), (1, u), (2, g), (2, u)$. Para obtener una lista del generador, basta con convertir el generado en lista usando la función `list`. Si se quiere el producto cartesiano de un solo iterable, basta con colocar asignar el argumento `repeat=num` dentro de la función `product`, donde `num` es un entero positivo.

Problema 4: Complete la función `prod_cart`. Esta función acepta como argumentos dos conjuntos, y retorna el conjunto resultado del producto cartesiano de estos dos conjuntos. Sugerencia: utilice la función `product` de la librería `itertools`. (*5 puntos*)

Problema 5: Complete la función `verf_rel`. Esta función acepta como argumentos tres conjuntos X, Y, W, y retorna `True` si W es una relación de X a Y, sino retorna `False`. (*5 puntos*)

La compresion de listas (resp. de conjuntos) permite crear una lista (conjunto) con la instrucción `[<var> for <var> in <iterable> if <condición>]` (`{<var> for <var> in <iterable> if <condición>}`), donde `<var>` es una variable, `<iterable>` es un iterable (lista, tupla o conjunto), y `<condición>` es una declaración que, si es cierta, entonces `<var>` forma parte de la nueva lista. Si se quieren elementos de dos iterables distintos, basta con colocar dos `for <iterable>` uno seguido de otro. Por ejemplo, `{<var1>, <var2> for <iterable1> for <iterable2> if <condición>}`.

Problema 6: Cree las siguientes relaciones, con R una relación sobre A y $a, b \in A$: (*5 puntos*)

a) $a \ R \ b \Leftrightarrow a = b$

b) $a \ R \ b \Leftrightarrow a < b$

c) $a \ R \ b \Leftrightarrow a \leq b$

d) $a \ R \ b \Leftrightarrow a|b$

e) $a \ R \ b \Leftrightarrow a \equiv b \pmod{227}$

La librería `networkx`[3] es "un paquete de Python para la creacón, manipulación y estudio de la estructura, dinámicas y funciones de redes complejas". Para tener esta librería, se tiene que instalar el paquete `networkx` usando el comando `pip install`

[2] `https://docs.python.org/es/3/library/itertools.html#itertools.product`

[3] `https://networkx.org/documentation/stable/`

`networkx` en la línea de comando (símbolos del sistema o consola CMD en Windows). Para usar esta librería en un programa, se debe importarla. Podemos importarla con la declaración `import networkx as nx` al inicio del programa, que nos permite usar `nx` como un sobrenombre más corto para la librería. Entre sus funcionalidades, está la de crear digrafos mediante la función `DiGraph`[4].

Problema 7: Cree un grafo vacío usando la función `nx.DiGraph` y asignenlo a una variable llamada `GD`. (*5 puntos*)

La función `add_node` permité añadir un vértice a un digrafo. Por ejemplo, si `AD` es una variable de un digrafo, entonces `AD.add_node(1)` añade 1 como un vértice de `AD`. La función `add_nodes_from` permite añádir vértices de cualquier iterable (*e.g.* `AD.add_nodes_from([1,2,3])`).

Problema 8: Añada los elementos del conjunto A como vertices de `GD`. Sugerencia: utilice la función `add_nodes_from` para agregar los vértices al digrafo `GD`. (*5 puntos*)

La función `add_edge` permité añadir un arco a un digrafo. Por ejemplo, si `AD` es un digrafo que tiene 1 y 2 como vértices, entonces `AD.add_edge(1, 2)` añade un arco de 1 a 2 a `AD`. Similarmente, la función `add_edges_from` permite añádir arcos de cualquier iterable de arcos (*e.g.* `AD.add_nodes_from([(1,2), (2,1)])`).

Problema 9: Añada los pares ordenados de la relación del problema 6.d como arcos de `GD`. Sugerencia: utilice la función `add_edges_from` para agregar los vértices al digrafo `GD`. (*5 puntos*)

La función `nodes`, de la variable `GD`, permite obtener los vértices del digrafo `GD`. La función `edges`, también de la variable `GD`, permite obtener los arcos del digrafo `GD`. La función `number_of_nodes` permite obtener el número de vértices que tiene un digrafo. La función `number_of_edges` permite obtener el número de arcos que tiene un digrafo.

Problema 10: Imprima los vértices y el número vértices, los arcos y el número arcos de `GD`. Sugerencia: utilice la función `nodes` y `edges`. (*5 puntos*)

La librería `matplotlib`[5] es "una librería comprensiva para crear visualizaciones estáticas, dinámicas e interactivas en Python". Para tener esta librería, se tiene que instalar el paquete `matplotlib` usando el comando `pip install matplolib` en la línea de comando (símbolos del sistema o consola CMD en Windows). Para usar esta librería en un programa, se debe importarla. Podemos importarla con la declaración `import matplotlib.pyplot as plt`[6] al inicio del programa, que nos permite usar

[4] `https://networkx.org/documentation/stable/reference/classes/digraph.html#networkx.DiGraph`

[5] `https://matplotlib.org/`

[6] En este caso, `pyplot` es la interface para utilizar `matplolib`, asi que la mayoría de las funciones son por medio de `pyplot`

`plt` como un sobrenombre más corto para la librería.

Para dibujar un digrafo, se utiliza la función `nx.draw`, que acepta como argumento un digrafo `G`, entre otros argumentos opcionales seguidos del primer argumento.

Problema 11: Dibuje el digrafo usando la función `nx.draw`[7], con los siguientes argumentos: (*5 puntos*)

a) `arrows=True`

b) `arrowstyle="->"`

c) `connectionstyle='arc3, rad = 0.1'`

d) `with_labels=True`

Problema 12: Muestre en pantalla el digrafo usando la función `plt.show`. (*5 puntos*)

La función `plt.savefig` acepta como argumento un nombre de ruta de archivo, y guardar la figura en el archivo especificado en la ruta. Si el nombre de ruta termina en `.pdf` o `.png`, `plt.savefig` guardará la figura como un archivo PDF o PNG respectivamente.

Problema 13: Complete la función `guardar_digafro`. Esta función acepta como argumento una ruta de archivo, y guarda la figura del digrafo usnado la función `plt.savefig`. (*5 puntos*)

La función `nx.all_simple_paths`[8] acepta como argumentos un digrafo, un vértice de inicio y un vértice final, y permite obtener todas trayectorias simples de un digrafo entre dos vértices.

Problema 14: Complete la función `obtener_trayectorias`. Esta función acepta como argumento un digrafo, y devuelve la lista de todas las trayectorias simples en el digrafo. Sugerencia: utilice la función `nx.all_simple_paths`. Para obtener los vértices de un digrafo, utilice la función `nodes`. (*5 puntos*)

Problema 15: Complete la función `tray_long_n`. Esta función acepta como argumentol `lista_tray` una lista de trayectorias y `n` un entero positivo, y devuelve la lista de todas las trayectorias de longitud n de `lista_tray`. (*5 puntos*)

El argumento `node_color` de `nx.draw` acepta una lista de nombres de colores para colorear los vértices. Esta lista contine tantos elementos como vértices tiene el digrafo. Por ejemplo, si los vértices son $1, 2, 3$ (segun la función `nodes`) y la lista es `[blue, red, blue]`, los vértices 1 y 3 serán de color azul, mientras que el vértice 2

[7] `https://networkx.org/documentation/stable/reference/generated/networkx.drawing.nx_pylab.draw.html#networkx.drawing.nx_pylab.draw`

[8] `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html#networkx.algorithms.simple_paths.all_simple_paths`

será de color rojo. Otro argumento, llamado `edge_color`, acetpa una lista de nombre de colores para colorear los arcos, de manera similar a `node_color`. También, el argumento `width`, acetpa una lista de flotantes para modificar el ancho de los arcos.

Problema 16: Coloree los vértices y los arcos, y modifique el ancho de los arcos de la trayectoria $1, 6, 4$ en el digrafo `GD` usando la función `nx.draw`. Sugerencia: utilice la función `plt.clf` para borrar el dibujo anterior antes de hacer un nuevo dibujo. (*5 puntos*)

Problema 17: Elimine todos los arcos de `GD` usando la función `clear_edges`[9], y agrege los arcos de la siguiente relación: $\{(1,2),(1,6),(2,3),(3,3),(3,4),(4,3),(4,1),(4,5),(6,4),(1,7),(7,2)\}$. (*5 puntos*)

La función `nx.simple_cyles`[10] acepta como argumento un digrafo, y devuelve la lista de trayectorias sin el último vértice. La función `nx.selfloop_edges`[11] acepta como argumento un digrafo, y devuelve la lista de trayectorias de longitud 1.

Problema 18: Complete la función `ciclos_simples`. Esta función acepta como argumento un digrafo, y devuelve la lista de todos los ciclos simples en el digrafo. Sugerencia: utilice al función `nx.simple_cyles`. (*5 puntos*)

Problema 19: Complete la función `ciclos_long_1`. Esta función acepta como argumento un digrafo, y devuelve la lista de todos los ciclos de longitud 1 en el digrafo. Sugerencia: utilice al función `nx.selfloop_edges`. (*5 puntos*)

Problema 20: Coloree los vértices y los arcos, y modifique el ancho de los arcos del ciclo $1, 2, 3, 4, 1$ en el digrafo `GD` usando la función `nx.draw`. Sugerencia: utilice la función `plt.clf` para borrar el dibujo anterior antes de hacer un nuevo dibujo. (*5 puntos*)

[9] `https://networkx.org/documentation/stable/reference/classes/generated/networkx.DiGraph.clear_edges.html#networkx.DiGraph.clear_edges`

[10] `https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cycles.simple_cycles.html#networkx.algorithms.cycles.simple_cycles`

[11] `https://networkx.org/documentation/stable/reference/generated/networkx.classes.function.selfloop_edges.html#selfloop-edges`