



Grafos

Matemática Discreta

23 de mayo de 2021

Ronald Ernesto Tejada Ríos

retejada@alu.ucam.edu.sv

A50274908

Ingeniería Informática – UCAM

Curso 2020/2021

Contenido

Grafos	1
Matemática Discreta.....	1
Contenido	2
Enunciado de la tarea:	3
Parte I: Aprendiendo a dibujar grafos con Python 3 con Jupyter en Windows	4
Instalación	4
Guía rápida para usar Jupyter.....	6
Dibujando grafos con Jupyter	6
Grafo nulo	6
Grafo con caminos (aristas)	7
Grafos dirigidos.....	7
Usando matriz de adyacencia.....	8
Parte II: Producto tensorial de grafos y coloración de grafos.....	9
Un matemático ruso desmiente una conjetura con más de medio siglo de vida.....	9
Ejemplo 1:	11
Ejemplo 2:	11
Colorear un grafo producto tensorial	12
Ejemplo 1 de coloreo de un producto tensorial.....	12
Anexos.....	19
Programas, ejecutables y código usados	19

Enunciado de la tarea:

La tarea consiste en dos partes.

- 1º Instalarnos Jupyter(*), para dibujar grafos y que utilizaremos en la solución del siguiente problema:
- 2º El pasado 19 de julio de 2019 se publicó en elpais.com: Un matemático ruso desmiente una conjetura con más de medio siglo de vida([Yaroslav Shitov: Un matemático ruso desmiente una conjetura con más de medio siglo de vida | Ciencia | EL PAÍS \(elpais.com\)](#)).

Se pide:

- Explicar el artículo.
- Explicar que es un producto tensorial de grafos.
- Poner un ejemplo de la hipótesis de colorear un grafo producto tensorial.

La respuesta se presentará en PDF de manera similar al trabajo que se adjunta.

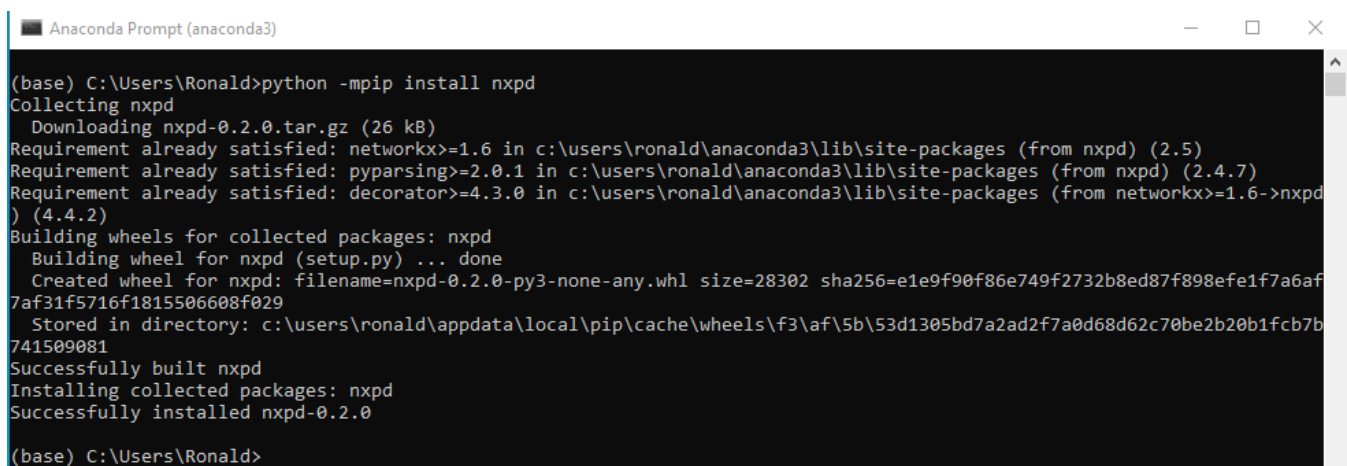
(*)Se puede utilizar otra aplicación para crear gráficos. En ese caso, se explicará la herramienta utilizada y su proceso para su utilización.

Parte I: Aprendiendo a dibujar grafos con Python 3 con Jupyter en Windows

Instalación

Para realizar esta práctica nos vamos a apoyar de Jupyter para poder dibujar los grafos a traves de librerias de Python 3.

- Se instala anaconda para windows guiandote en la documentación¹ o bien puedes descargarlo directamente desde aquí².
- Se instalan librerias en la consola de Anaconda (Anaconda Prompt) usando el comando:
 - `python -mpip install nxpd`



```

Anaconda Prompt (anaconda3)

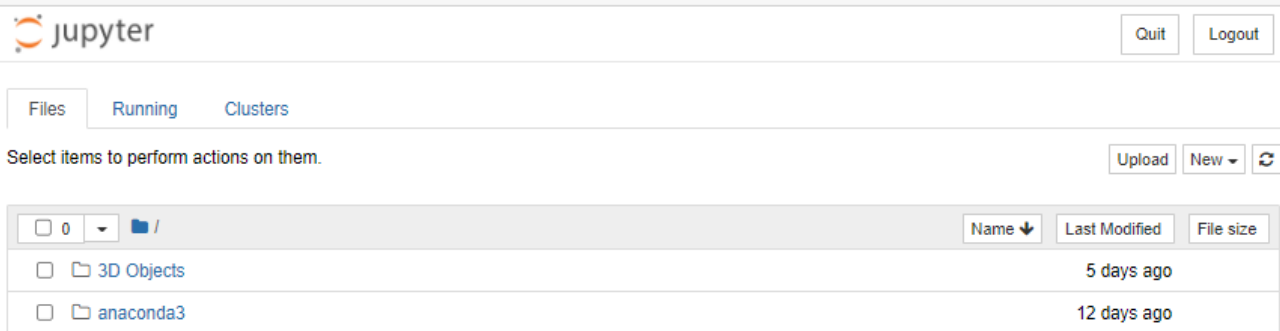
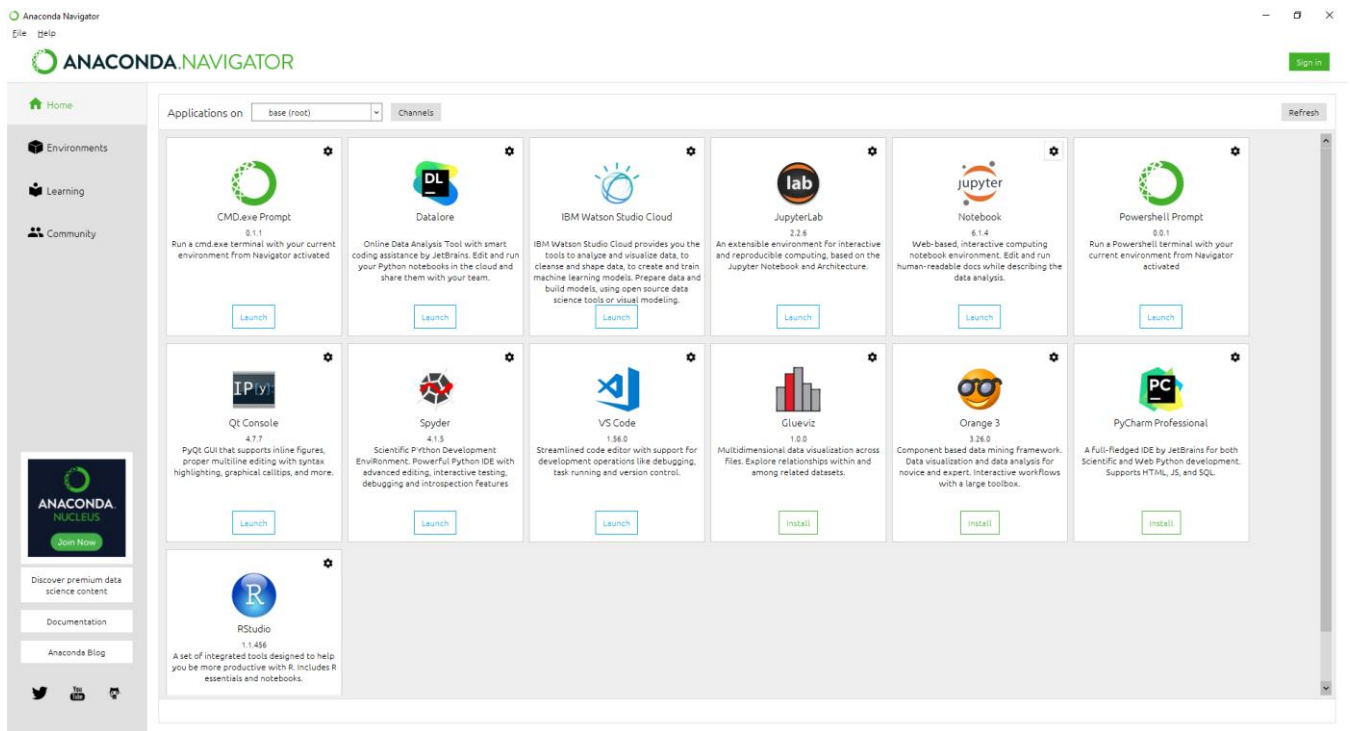
(base) C:\Users\Ronald>python -mpip install nxpd
Collecting nxpd
  Downloading nxpd-0.2.0.tar.gz (26 kB)
Requirement already satisfied: networkx>=1.6 in c:\users\ronald\anaconda3\lib\site-packages (from nxpd) (2.5)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\ronald\anaconda3\lib\site-packages (from nxpd) (2.4.7)
Requirement already satisfied: decorator>=4.3.0 in c:\users\ronald\anaconda3\lib\site-packages (from networkx>=1.6->nxpd) (4.4.2)
Building wheels for collected packages: nxpd
  Building wheel for nxpd (setup.py) ... done
  Created wheel for nxpd: filename=nxpd-0.2.0-py3-none-any.whl size=28302 sha256=e1e9f90f86e749f2732b8ed87f898efe1f7a6af7af31f5716f1815506608f029
  Stored in directory: c:\users\ronald\appdata\local\pip\cache\wheels\f3\af\5b\53d1305bd7a2ad2f7a0d68d62c70be2b20b1fcb7b741509081
Successfully built nxpd
Installing collected packages: nxpd
Successfully installed nxpd-0.2.0

(base) C:\Users\Ronald>
```

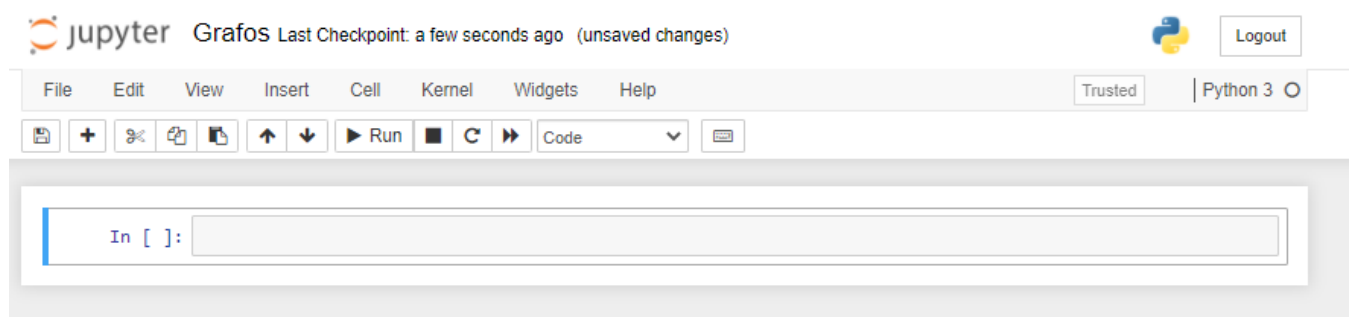
- Procedemos a abrir Anaconda Navigator y luego inicializar (launch) Jupyter Notebook:

¹ [Installing on Windows — Anaconda documentation](#) pueden apoyarse de este tutorial de la documentación

² [Anaconda | Individual Edition](#)



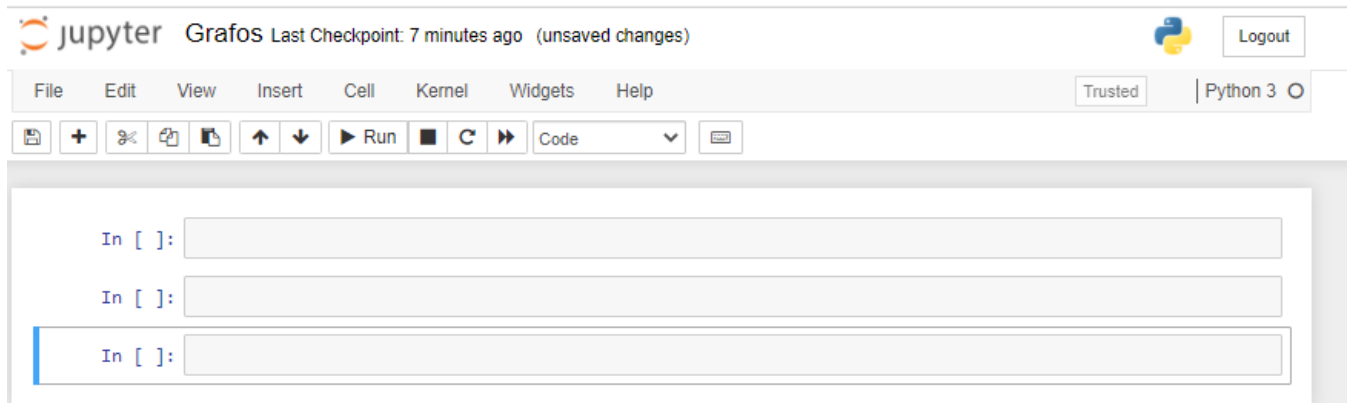
- Cuando Jupyter se abra en nuestro navegador predeterminado tendremos una interfaz como la de abajo. Ahí daremos clic en “New/python 3” para crear un nuevo archivo.



Ahora ya podemos empezar a programar.

Guía rápida para usar Jupyter

Una vez acá podemos empezar a escribir código en Python y ejecutarlo.



Cada línea que dice “In []” es un segmento en el que podemos escribir código para ejecutarlo.

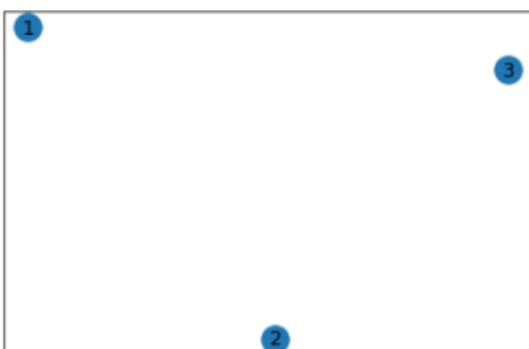
- Para **ejecutar** tenemos que estar en el cuadro de texto que se quiere ejecutar y presionar **run** o bien **ctrl + enter**.
- Si quiere **borrar** un cuadro de texto tienes que presionar la tijera.
- Si quieres **agregar** otro cuadro de texto para código, presional el boton del más.
- Para **cambiar el nombre del archivo** solo da clic en donde dice “Grafos” al lado de Jupyter en la esquina superior izquierda, ese es el nombre del archivo.

Dibujando grafos con Jupyter

Grafo nulo

```
import networkx as nx

G=nx.Graph()
G.add_node(1)
G.add_node(2)
G.add_node(3)
nx.draw_networkx(G)
```



Es un grafo cuyos vertices no tienen aristas que los conecten.

Acá hemos aprendido a agregar vertices a la hora de pintar un grafo.

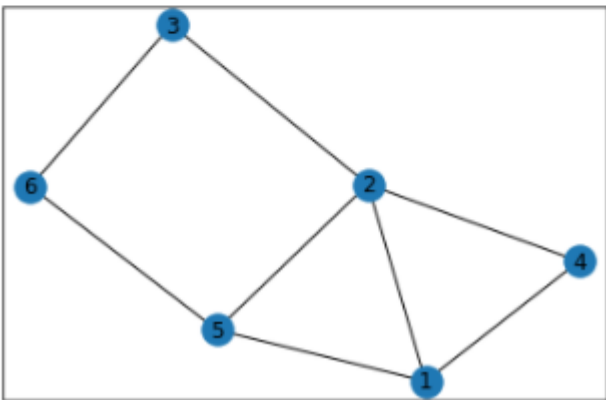
Grafo con caminos (aristas)

```
import networkx as nx

G=nx.Graph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)

G.add_edge(1,2)
G.add_edge(3,2)
G.add_edge(2,4)
G.add_edge(1,4)
G.add_edge(2,5)
G.add_edge(6,5)
G.add_edge(3,6)
G.add_edge(1,5)
nx.draw_networkx(G)
```



Ahora procedemos a complicar un poco más el grafo y a agregar aristas que unan los vertices entre sí.

Esto será muy útil para la práctica puesto que vamos a tener que crear aristas que unan los vertices del producto tensorial.

Grafos dirigidos

Notese que usando el mismo código, si lo que necesitamos hacer es un grafo dirigido. Lo único que hay que cambiar es la línea donde se inicializa el grafo:

```
G = nx.Graph()
```

Se cambia por

```
G = nx.DiGraph()
```

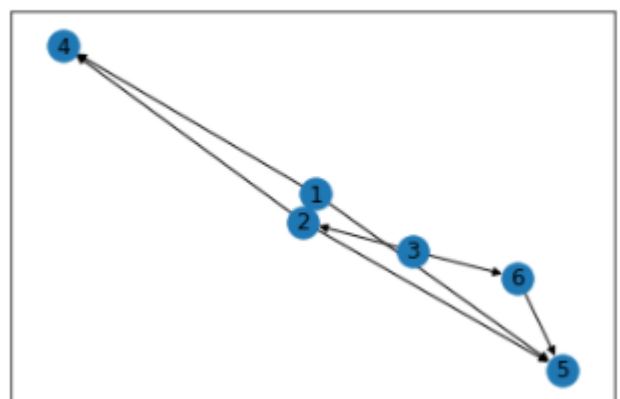
Y ahora tenemos ya un grafo dirigido como de muestra a la derecha.

```
import networkx as nx

G=nx.DiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)

G.add_edge(1,2)
G.add_edge(3,2)
G.add_edge(2,4)
G.add_edge(1,4)
G.add_edge(2,5)
G.add_edge(6,5)
G.add_edge(3,6)
G.add_edge(1,5)
nx.draw_networkx(G)
```



Usando matriz de adyacencia

```
import networkx as nx
import numpy as np #Usamos esta libreria

print("Nodo Aislado")
G1 = np.array([
    [0,0,1,0],
    [1,0,1,0],
    [0,1,0,0],
    [0,0,0,0]
])

G1 = nx.from_numpy_matrix(G1)
nx.draw_networkx(G1)
```

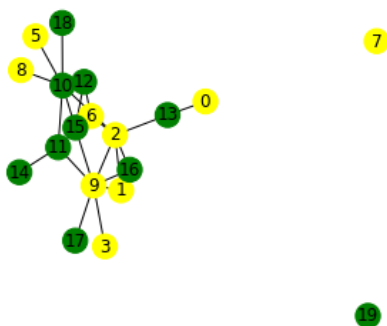
Nodo Aislado



```
import networkx as nx
import numpy as np #Usamos esta libreria
import matplotlib.pyplot as plt

G = nx.erdos_renyi_graph(20, 0.1)
color_map = []
for node in G:
    if node < 10:
        color_map.append('yellow')
    else:
        color_map.append('green')

nx.draw(G, node_color=color_map, with_labels=True)
plt.show()
```



Al hacer la matriz de adyacencia se puede observar que en la cuarta fila no hay ningún 1. Por ende ese nodo no tiene ningún camino hacia otro nodo.

Por último tenemos este código que vamos a adaptar posteriormente para colorear los grafos de los productos tensoriales.

Con esto tenemos todo lo que necesitamos para resolver los problemas planteados en este trabajo.

Parte II: Producto tensorial de grafos y coloración de grafos.

Un matemático ruso desmiente una conjetura con más de medio siglo de vida³

Uno de los problemas más estudiados es encontrar el mínimo número de colores que se pueden asignar de tal forma que no existan dos vertices con el mismo color y que estén unidos por una arista. La historia de estos problemas se remontan a mediados del siglo XIX con la conjetura de Frabcus Guthrie y su hermano que dice que “*cualquier grafo que se pueda dibujar en el plano de forma que dos aristas distintas no se crucen, salvo en un vértice común, se puede colorear siempre con cuatro colores o menos*”. Este es llamado el **problema de los 4 colores**. El cual fue resuelto aproximadamente 125 años después, en 1976, por Kenneth Appel y Wolfgang Hanken.

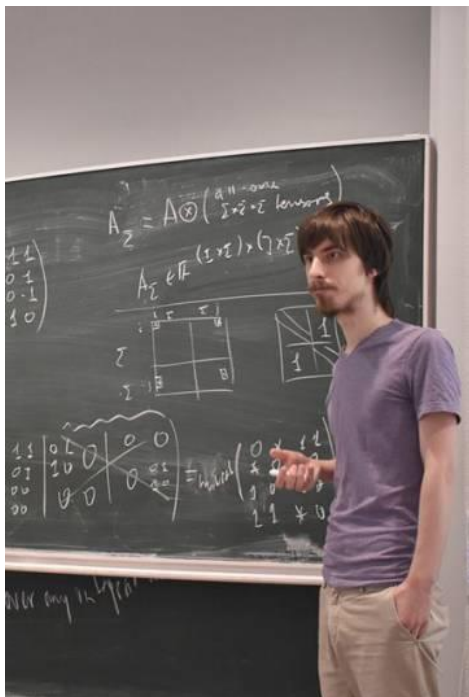
El **problema de los 4 colores** puede parecer sencillo, pero tiene una complejidad bastante grande cuando la cantidad de vertices y aristas aumenta. Tanto que existe un premio de un millón de dólares para quien encuentre un mejor algoritmo de coloración de los actuales.



Es en este campo donde entra Stephen Hedetniemi, quien en 1966 conjeturó en su tesis doctoral que dados dos grafos G y H , el número de colores necesario para colorear el grafo producto tensorial $G \times H$ es el menor de los colores necesarios para colorear G y H . Este grafo se obtiene combinando de cierta forma los vértices y aristas de G y H .

Conjetura que se cumplía con todos los ejemplos encontrados hasta antes de junio del 2019.

³ https://elpais.com/elpais/2019/07/01/ciencia/1561975026_683783.html



Acá es donde entra Yaroslav Shitov (imagen de la izquierda), quien ha comprobado que una conjetura matemática es falsa.

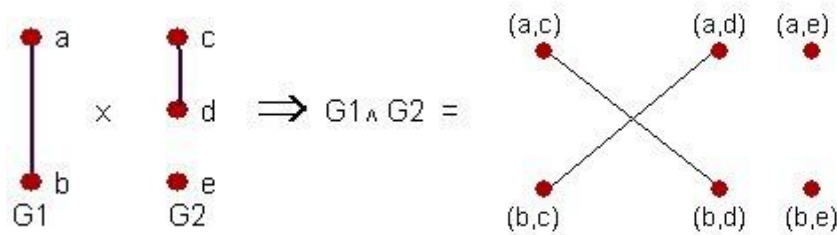
Los ejemplos G y H de Shitov son grafos enormes: es posible que G tenga 2^{200} vértices, lo cual es un número inmenso, de un orden de magnitud cercano al número de partículas elementales que podemos encontrar en todo el universo visible. Pero H es aún mayor, mucho mayor, con más de 2^{20000} vértices. De hecho no los construye explícitamente en su corto artículo⁴ (dos páginas y media). Pero demuestra que existen basándose en propiedades conocidas, con lo que queda resuelto el problema.

⁴ [COUNTEREXAMPLES TO HEDETNIEMI'S CONJECTURE – YAROSLAV SHITOV](#)

Producto Tensorial de grafos

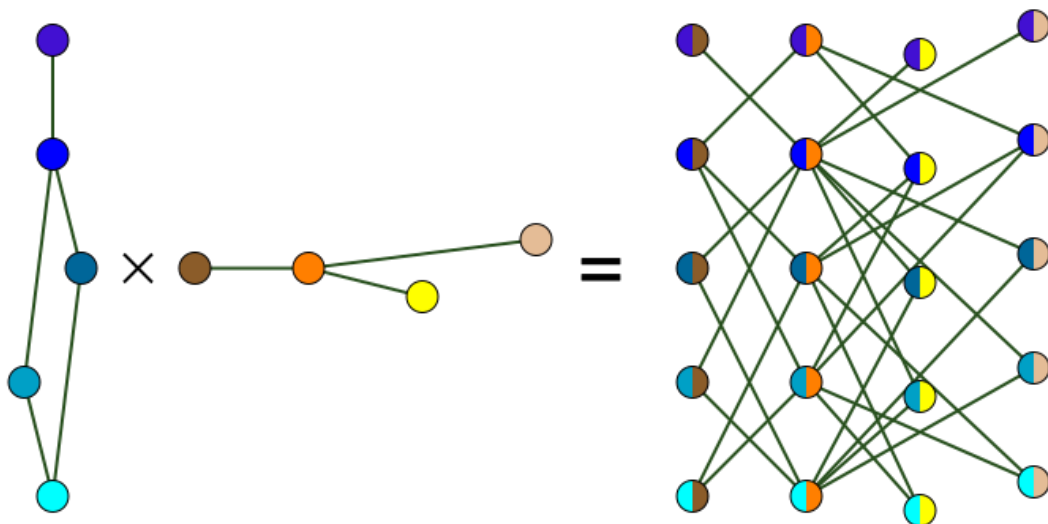
Productos tensoriales: grafos que se obtienen combinando dos grafos diferentes (llámese G y H) de una manera determinada. El producto tensorial de G y H es un nuevo grafo más grande en el que cada nodo representa un par de nodos de los grafos originales -uno de G y otro de H- y dos nodos del producto tensorial están conectados si sus correspondientes nodos en G y sus correspondientes nodos en H están conectados.

Ejemplo 1:



Como se aprecia en las imágenes de arriba ese es un ejemplo del producto tensorial. Cabe recalcar que los puntos (a,e) y (b,e) no están conectados a ningún nodo cumpliéndose las características de un producto tensorial, pues e no tiene ningún nodo conectado a él en G_2 .

Ejemplo 2:



Este ejemplo es mucho más fácil de visualizar pues se puede ver que cada nodo del resultado está compuesto de dos colores (uno por cada nodo de sus grafos originales correspondientes al producto tensorial) y que las aristas de este efectivamente conectan únicamente si ambos nodos en sus grafos originales están conectados.

Colorear un grafo producto tensorial

Ejemplo 1 de coloreo de un producto tensorial

Para este primer ejemplo vamos a tomar como referencia el último ejemplo mostrado.

```
%matplotlib notebook

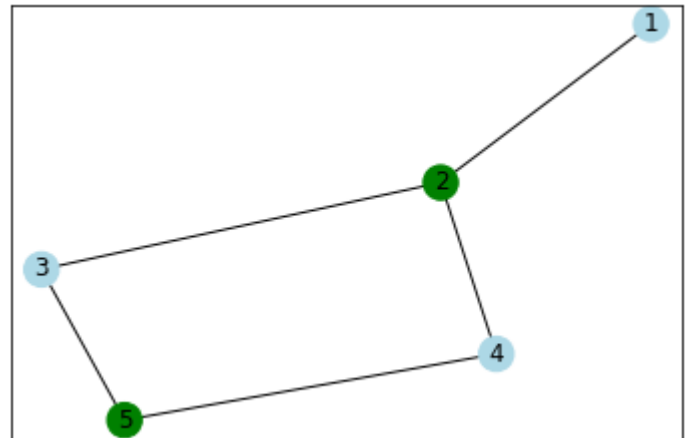
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(2,4)
G.add_edge(4,5)
G.add_edge(3,5)
color_map = []
color_map.append('lightblue')
color_map.append('green')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('green')

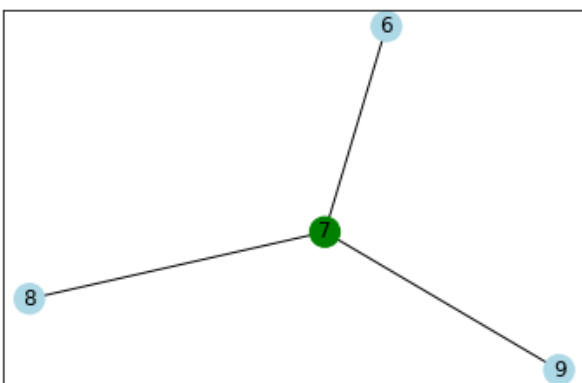
nx.draw_networkx(G, node_color = color_map)
```

Definamos el grafo G1:



Como se observa, es obvio que con 2 colores es la mínima cantidad de colores para colorear G1.

De manera analoga definimos el grafo H1:



Y como se observa se puede colorear con solo 2 colores.

```
%matplotlib notebook

import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
G.add_node(6)
G.add_node(7)
G.add_node(8)
G.add_node(9)

G.add_edge(6,7)
G.add_edge(7,8)
G.add_edge(7,9)

color_map = []
color_map.append('lightblue')
color_map.append('green')
color_map.append('lightblue')
color_map.append('lightblue')

nx.draw_networkx(G, node_color = color_map)
```

- Ahora procedemos a hacer el producto tensorial de $G1 \wedge H1$:

Para facilitar la construcción del producto tensorial se ha creado un programa que haga el producto tensorial (puntos y aristas) para que pueda solo imprimirlo en Jupyter.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    ///DATOS WIKIPEDIA
    int n1 = 5;
    int n2 = 4;
    char G1[5] = "12345";
    char G2[4] = "6789";

    int CantCaminos = 8;
    char edges[8][2] ={
        "12", "23", "24", "45", "35",
        "67", "78", "79"
    };
};
```

```
G.add_node('16') #0
G.add_node('17') #1
G.add_node('18') #2
G.add_node('19') #3
G.add_node('26') #4
G.add_node('27') #5
G.add_node('28') #6
G.add_node('29') #7
G.add_node('36') #8
G.add_node('37') #9
G.add_node('38') #10
G.add_node('39') #11
G.add_node('46') #12
G.add_node('47') #13
G.add_node('48') #14
G.add_node('49') #15
G.add_node('56') #16
G.add_node('57') #17
G.add_node('58') #18
G.add_node('59') #19
G.addEdge('16', '27')
G.addEdge('17', '26')
G.addEdge('17', '28')
G.addEdge('17', '29')
G.addEdge('18', '27')
G.addEdge('19', '27')
G.addEdge('26', '17')
G.addEdge('26', '37')
G.addEdge('26', '47')
G.addEdge('27', '16')
G.addEdge('27', '36')
G.addEdge('27', '38')
G.addEdge('27', '39')
G.addEdge('27', '46')
G.addEdge('27', '48')
G.addEdge('27', '49')
G.addEdge('28', '37')
G.addEdge('28', '47')
G.addEdge('29', '37')
G.addEdge('29', '47')
G.addEdge('36', '57')
G.addEdge('37', '56')
G.addEdge('37', '58')
G.addEdge('37', '59')
G.addEdge('38', '57')
G.addEdge('39', '57')
G.addEdge('46', '57')
G.addEdge('47', '56')
G.addEdge('47', '58')
G.addEdge('47', '59')
G.addEdge('48', '57')
G.addEdge('49', '57')
```

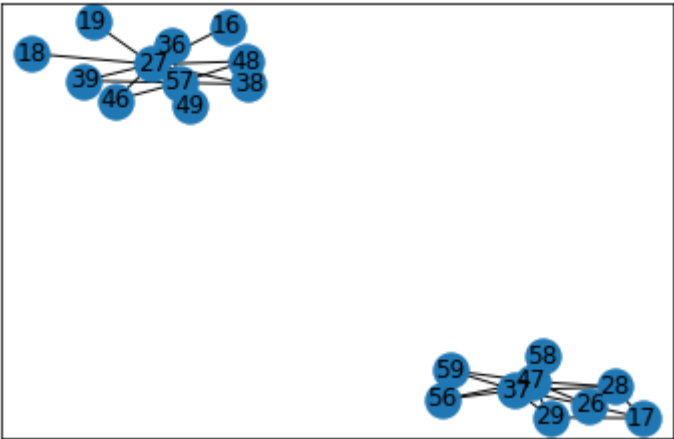
```
import networkx as nx
import matplotlib.pyplot as plt

G=nx.Graph()

G.add_node('16') #0
G.add_node('17') #1
G.add_node('18') #2
G.add_node('19') #3
G.add_node('26') #4
G.add_node('27') #5
G.add_node('28') #6
G.add_node('29') #7
G.add_node('36') #8
G.add_node('37') #9
G.add_node('38') #10
G.add_node('39') #11
G.add_node('46') #12
G.add_node('47') #13
G.add_node('48') #14
G.add_node('49') #15
G.add_node('56') #16
G.add_node('57') #17
G.add_node('58') #18
G.add_node('59') #19

G.add_edge('16', '27')
G.add_edge('17', '26')
G.add_edge('17', '28')
G.add_edge('17', '29')
G.add_edge('18', '27')
G.add_edge('19', '27')
G.add_edge('26', '17')
G.add_edge('26', '47')
G.add_edge('27', '36')
G.add_edge('27', '38')
G.add_edge('27', '39')
G.add_edge('27', '46')
G.add_edge('27', '48')
G.add_edge('27', '49')
G.add_edge('28', '37')
G.add_edge('28', '47')
G.add_edge('29', '37')
G.add_edge('29', '47')
G.add_edge('36', '57')
G.add_edge('37', '56')
G.add_edge('37', '58')
G.add_edge('37', '59')
G.add_edge('38', '57')
G.add_edge('39', '57')
G.add_edge('46', '57')
G.add_edge('47', '56')
G.add_edge('47', '58')
G.add_edge('47', '59')
G.add_edge('48', '57')
G.add_edge('49', '57')

#30 caminos
nx.draw_networkx(G)
```



- Procedemos a buscar la coloración mínima:

Ya que estamos hablando de un grafo bastante grande, vamos a pensar en la mejor manera de colorearlo: usando un algoritmo⁵.

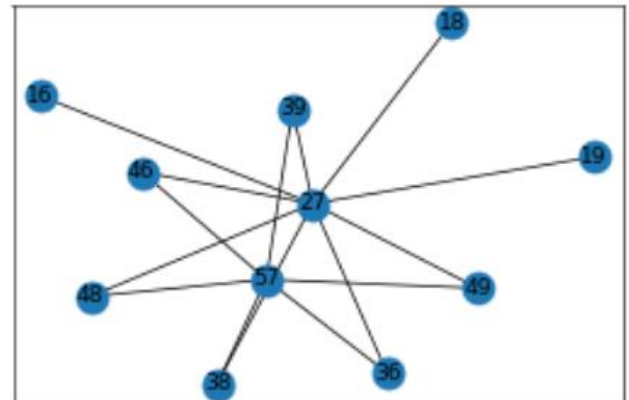
En este caso el producto tensorial está conformado por dos grafos que no se unen por ninguna arista, así que habrá que colorear los grados por separado.

```
%matplotlib notebook

import networkx as nx
import matplotlib.pyplot as plt

G=nx.Graph()
G.add_node('17') #1
G.add_node('26') #4
G.add_node('28') #6
G.add_node('29') #7
G.add_node('37') #9
G.add_node('47') #13
G.add_node('56') #16
G.add_node('58') #18
G.add_node('59') #19

G.add_edge('17' , '26')
G.add_edge('17' , '28')
G.add_edge('17' , '29')
G.add_edge('26' , '37')
G.add_edge('26' , '47')
G.add_edge('28' , '37')
G.add_edge('28' , '47')
G.add_edge('29' , '37')
G.add_edge('29' , '47')
G.add_edge('37' , '56')
G.add_edge('37' , '58')
G.add_edge('37' , '59')
G.add_edge('47' , '56')
G.add_edge('47' , '58')
G.add_edge('47' , '59')
nx.draw_networkx(G)
```



En este caso, el reto es colorear el grafo con 2 colores, ya que dos colores fueron necesarios para colorear los grafos originales.

⁵ [Graph Coloring | Set 2 \(Greedy Algorithm\) - GeeksforGeeks](#)


```

%matplotlib notebook

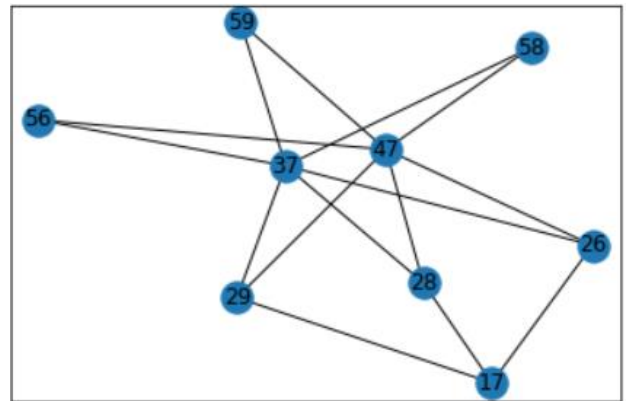
import networkx as nx
import matplotlib.pyplot as plt

G=nx.Graph()

G.add_node('16') #0
G.add_node('18') #2
G.add_node('19') #3
G.add_node('27') #5
G.add_node('36') #8
G.add_node('38') #10
G.add_node('39') #11
G.add_node('46') #12
G.add_node('48') #14
G.add_node('49') #15
G.add_node('57') #17

G.add_edge('19' , '27')
G.add_edge('16' , '27')
G.add_edge('18' , '27')
G.add_edge('27' , '36')
G.add_edge('27' , '38')
G.add_edge('27' , '39')
G.add_edge('27' , '46')
G.add_edge('27' , '48')
G.add_edge('27' , '49')
G.add_edge('36' , '57')
G.add_edge('38' , '57')
G.add_edge('39' , '57')
G.add_edge('48' , '57')
G.add_edge('46' , '57')
G.add_edge('49' , '57')
nx.draw_networkx(G)

```



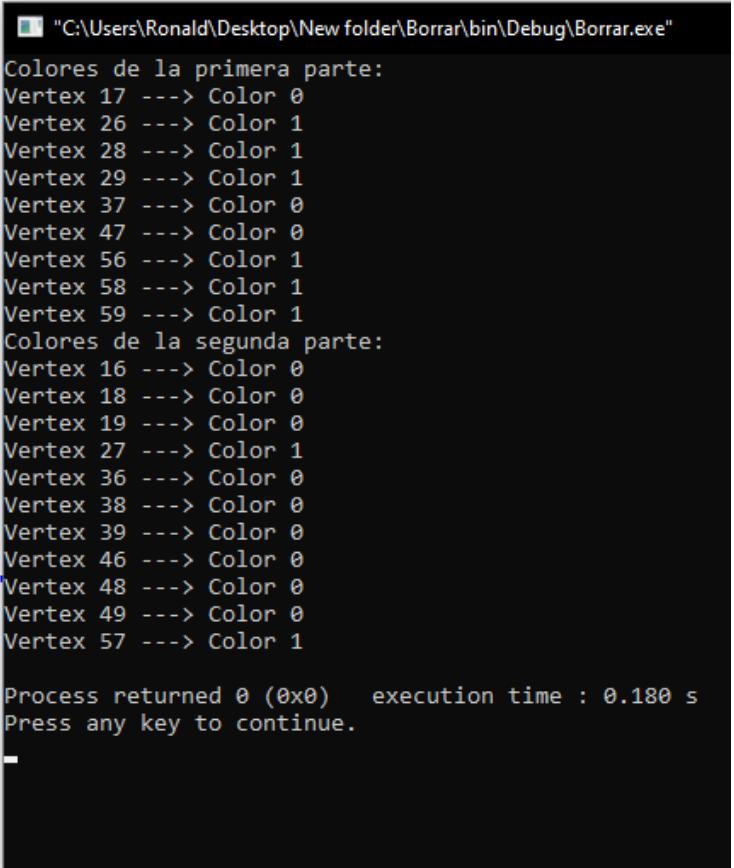
Para este caso a pesar de que el resultado es un grafo de 20 vertices (5 verties de G1 * 4 vertices de H1) y de 30 aristas (5 aristas de G1 * 3 aristas de H1 * 2), podria colorearse fácilmente probando colores.

Sin embargo, nos basaremos en soluciones algoritmicas que puedan ser reutilizables a futuro y de paso aprender un nuevo algoritmo.

Modificando el algoritmo de GeekForGeeks para colorear grafos, he obtenido los siguientes resultados:

```
Graph G(9);
string VerticesG1[9] = {"17","26","28","29","37","47","56","58","59"};
G.addEdge(0, 1);
G.addEdge(0, 2);
G.addEdge(0, 3);
G.addEdge(1, 4);
G.addEdge(1, 5);
G.addEdge(2, 4);
G.addEdge(2, 5);
G.addEdge(3, 4);
G.addEdge(3, 5);
G.addEdge(4, 6);
G.addEdge(4, 7);
G.addEdge(4, 8);
G.addEdge(5, 6);
G.addEdge(5, 7);
G.addEdge(5, 8);
cout << "Colores de la primera parte:\n";
G.greedyColoring(VerticesG1);

Graph G2(11);
string VerticesG2[11] = {"16","18","19","27","28","29","36","37","38","39","46"};
G2.addEdge(2, 3);
G2.addEdge(0, 3);
G2.addEdge(1, 3);
G2.addEdge(3, 4);
G2.addEdge(3, 5);
G2.addEdge(3, 6);
G2.addEdge(3, 7);
G2.addEdge(3, 8);
G2.addEdge(3, 9);
G2.addEdge(4, 10);
```



Donde se puede observar claramente que color asignar a cada vertice. Obteniendo como resultado que el grafo puede colorearse perfectamente con 2 colores (color 0 y color 1). Cumpliendose así lo que Hedetniemi había conjeturado.

A continuación se muestra el producto tensorial ya coloreado y el código final usado en Jupyter:


```
%matplotlib notebook
```

```
import networkx as nx
import matplotlib.pyplot as plt
```

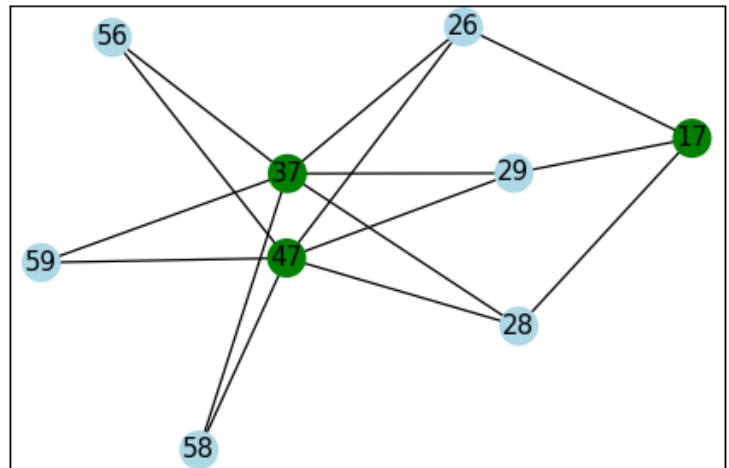
```
G=nx.Graph()
G.add_node('17') #1
G.add_node('26') #4
G.add_node('28') #6
G.add_node('29') #7
G.add_node('37') #9
G.add_node('47') #13
G.add_node('56') #16
G.add_node('58') #18
G.add_node('59') #19
```

```
G.add_edge('17' , '26')
G.add_edge('17' , '28')
G.add_edge('17' , '29')
G.add_edge('26' , '37')
G.add_edge('26' , '47')
G.add_edge('28' , '37')
G.add_edge('28' , '47')
G.add_edge('29' , '37')
G.add_edge('29' , '47')
G.add_edge('37' , '56')
G.add_edge('37' , '58')
G.add_edge('37' , '59')
G.add_edge('47' , '56')
G.add_edge('47' , '58')
G.add_edge('47' , '59')
```

```
color_map = []
color_map.append('green')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('green')
color_map.append('green')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
```

```
nx.draw_networkx(G, node_color = color_map)
```

Acá la solución donde se observa que ningún vertice tiene una arista con otro vertice del mismo color.



```

%matplotlib notebook

import networkx as nx
import matplotlib.pyplot as plt

G=nx.Graph()

G.add_node('16') #0
G.add_node('18') #2
G.add_node('19') #3
G.add_node('27') #5
G.add_node('36') #8
G.add_node('38') #10
G.add_node('39') #11
G.add_node('46') #12
G.add_node('48') #14
G.add_node('49') #15
G.add_node('57') #17

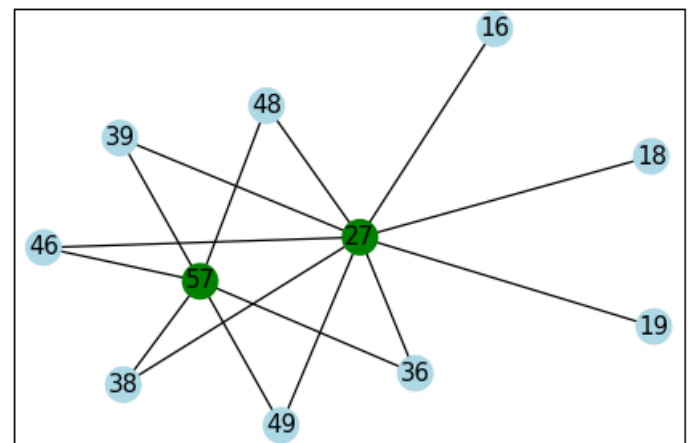
G.add_edge('19' , '27')
G.add_edge('16' , '27')
G.add_edge('18' , '27')
G.add_edge('27' , '36')
G.add_edge('27' , '38')
G.add_edge('27' , '39')
G.add_edge('27' , '46')
G.add_edge('27' , '48')
G.add_edge('27' , '49')
G.add_edge('36' , '57')
G.add_edge('38' , '57')
G.add_edge('39' , '57')
G.add_edge('48' , '57')
G.add_edge('46' , '57')
G.add_edge('49' , '57')
nx.draw_networkx(G)

color_map = []
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('green')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('lightblue')
color_map.append('green')

nx.draw_networkx(G, node_color = color_map)

```

Acá la segunda parte de la solución donde se observa que ningún vertice tiene una arista con otro vertice del mismo color.



Anexos

Programas, ejecutables y código usados

Los programas usados pueden ser encontrados en mi Github en el siguiente enlace:

[ronaldris21/MatematicaDiscretas_Grafos_ColoracionMinima_ProductoTensorial: Archivos usados para la solución de una tarea de la materia de Matemáticas Discretas en la Universidad Católica San Antonio de Murcia con el Dr Jesús Soto. \(github.com\)](https://github.com/ronaldris21/MatematicaDiscretas_Grafos_ColoracionMinima_ProductoTensorial)⁶

- MAD_Grafos_ProductoTensorial (C)
 - Aquí se han hecho los productos tensoriales y se han obtenido las aristas
- MAD_Grafos_ColoracionMinima
 - Se ha usado para en base a los resultados obtenidos de ejecutar el proyecto anterior, ahora se pueda encontrar una coloración optima. Este código se ha basado en el algoritmo de coloración de grafos de GeekForGeeks
- Grafos.ipynb
 - Es el archivo que se ejecuta en Jupyter donde están todos los códigos usados para dibujar grafos que se han usado de ejemplos en este trabajo.

⁶ https://github.com/ronaldris21/MatematicaDiscretas_Grafos_ColoracionMinima_ProductoTensorial