GIIT_AMA_2223_Practica3_jelanang_alokenveo

Práctica 3. Mapas y Coloraciones

Temporalización de la práctica: Lunes 24 de octubre de 2022

Entrega de la práctica: Desde el 24 de octubre hasta el 31 de octubre (ver tarea en el campus virtual o agenda de la asignatura)

Instrucciones:

1. Haz una copia de la hoja pública y renómbrala: si tu correo es **mariomp@alumnos.unex.es** añade al final del título **_mariomp**, por ejemplo GIIT_AMA_2223_Practica3_mariomp

- Para cambiar el nombre pulsa en el título de la hoja (arriba del todo, entre el logo de Sage y el menú "Archivo/File...")

2. Comparte la hoja de trabajo con el usuario **mariomp2223** mediante el botón Compartir/Share de arriba a la derecha. Si lo hacéis en pareja, compartid la hoja con el usuario de tu compañero (ambos usuarios separados por comas), así como añadir también el usuario en el título de la hoja (separados por _).

3. Completa la primera celda y trabaja la práctica.

4. Cuando hayas terminado, haz una copia en un único fichero PDF y ponlo en el campus virtual (Si lo hacéis en pareja, basta que lo suba uno). **Esa será la versión que se evaluará**. La hoja no se considera entregada si no se ha renombrado y compartido (pasos 1 y 2).

- Para generar el PDF lo más sencillo es usar el botón Imprimir/Print de arriba e imprimir la nueva página a fichero.

5. Una vez subido el PDF al campus virtual, no podrá modificarse esta hoja de trabajo. **Hacerlo conllevará la calificación de 0 en esta práctica**.

6. Los ejercicios a entregar se representan en Rojo y deben estar correctamente explicados. Los ejercicios indicados con ** **NO** serán obligatorios, pero aquellos que los hagan podrán ir sumando por cada uno de ellos 0.1 puntos adicionales en la calificación de la práctica.

Alumno/s: Jose Luis Obiang Ela Nanguan y Alfredo Mituy Okenve Obiang

Mapas

En primer lugar, se dice que un grafo es **plano** si admite una representación gráfica, llamada mapa, en el plano, de tal modo que no se corten las aristas. En Sage, la función ***.is_planar()*** permite conocer si un grafo es plano o no.

\r\n
\r\n

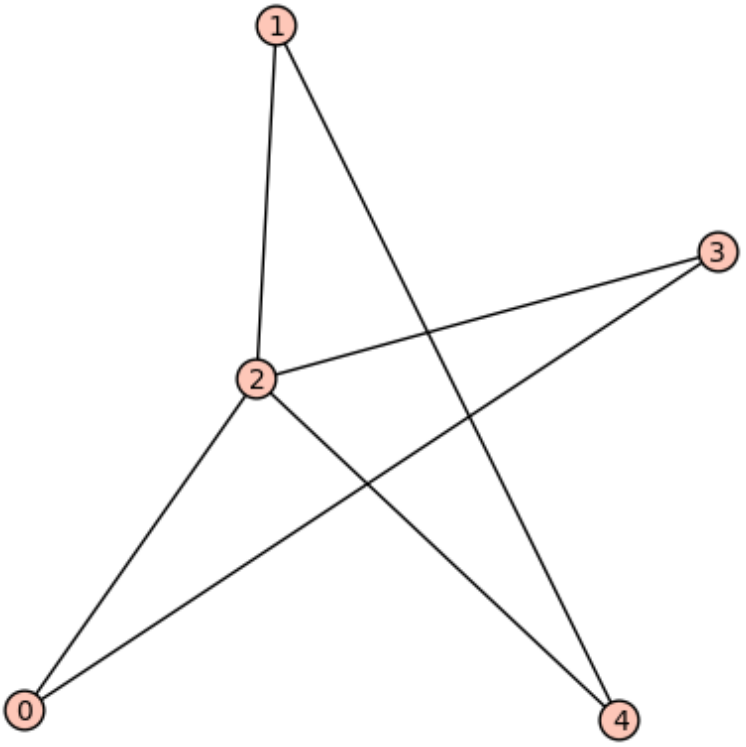
\r\n

\r\n

\r\n\r\n\r\n\r\n\r\n\r\n
' });

```
grafo = Graph({0:[2,3],1:[2,4],2:[4,3,0,1],3:[2,0],4:[2,1]}); grafo.set_pos({0:
[63,93],1:[140,302],2:[134,194],3:[275,233],4:[245,90]});  #graph_editor(grafo);
```

```
grafo.show()
```



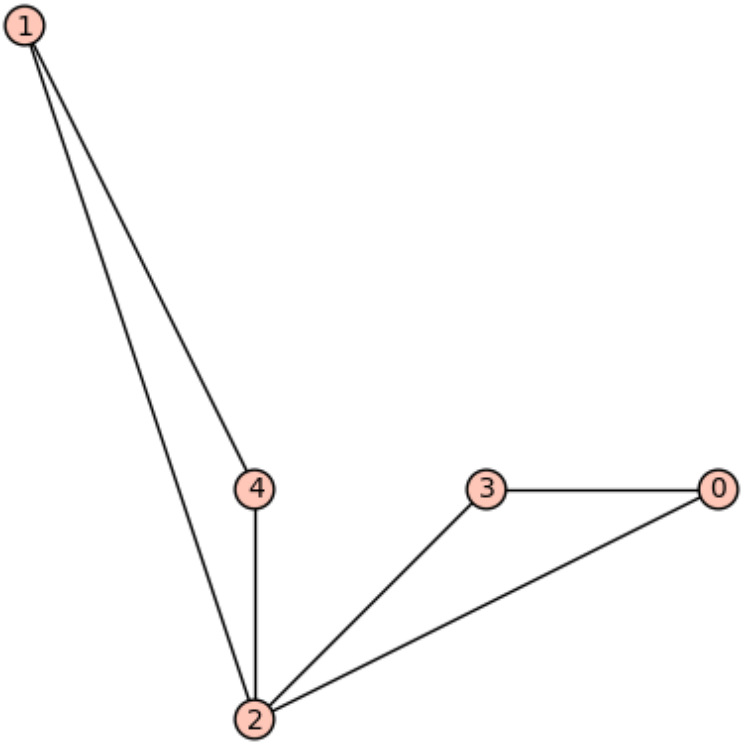

```
grafo.is_planar(),
len(grafo.edges());3*(len(grafo.vertices())-2)
```

```
6
9
```

En el caso de grafos planos, podemos representar el mapa asociado, para ello debemos añadir el argumento ***layout='planar'*** en la función *.show()*. Al representar el mapa, podremos deducir el número de ***regiones*** que presenta el grafo, recordando que una región es un camino cerrado tal que en su 'interior' no hay vértices ni

aristas. Además, se pueden calcular el ***grado de la región*** determinando la longitud del camino que la rodea (esto es, el número de aristas que la delimitan). La función ***.faces()*** proporciona las regiones del mapa de un grafo plano, si además, queremos determinar el grado de una región, basta comprobar el número de aristas con la función ***.num_edges()***.

```
grafo.show(layout='planar')
```

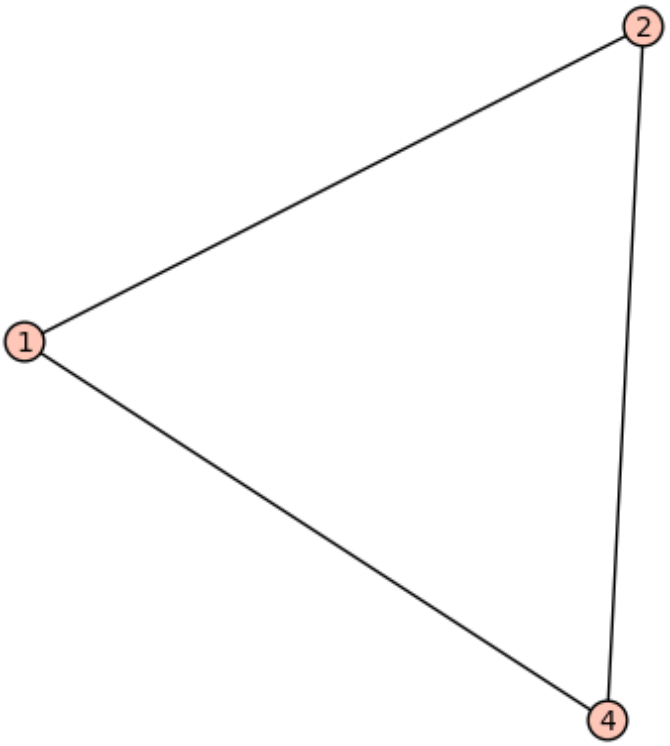



```
grafo.faces()
```

```
[[(1, 2), (2, 4), (4, 1)],
 [(3, 2), (2, 0), (0, 3)],
 [(2, 3), (3, 0), (0, 2), (2, 1), (1, 4), (4, 2)]]
```

```
region1=Graph(grafo.faces()[0])
region1.show()
print "Grado de la region 1:", region1.num_edges()
```

```
Grado de la region 1: 3
```



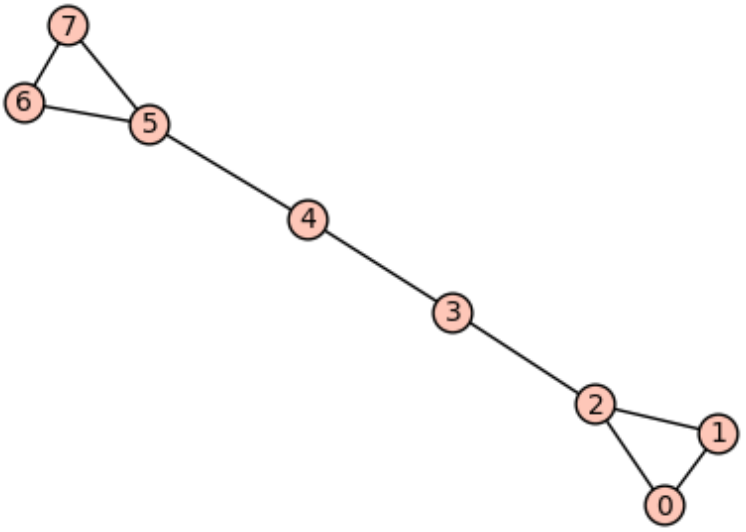
Al representar el mapa, observamos que este grafo presenta 2 regiones 'internas': una formada por el camino 1241 (con grado 3) y otra formada por el camino 0320 (con grado 3). Además, recordemos que la parte exterior del grafo también es una región, por lo que tiene una tercera región de grado 6. Y como las dos regiones 'internas' no tienen aristas en común, entonces podemos decir que no son adyacentes.

Ejercicio 1. Dado el grafo, representa el grafo y comprueba si se trata de un grafo plano. En caso afirmativo, representa el mapa e indica el número de regiones, así como su grado.

```
grafo1 = Graph({0:[1,2],1:[0,2],2:[0,1,3],3:[2,4],4:[3,5],5:[4,6,7],6:[5,7],7:
[5,6]}); grafo.set_pos({0:[30,119],1:[122,33],2:[272,263],3:[191,296],4:
[19,161],5:[207,29],6:[337,310],7:[128,322]});  #graph_editor(grafo1);
```

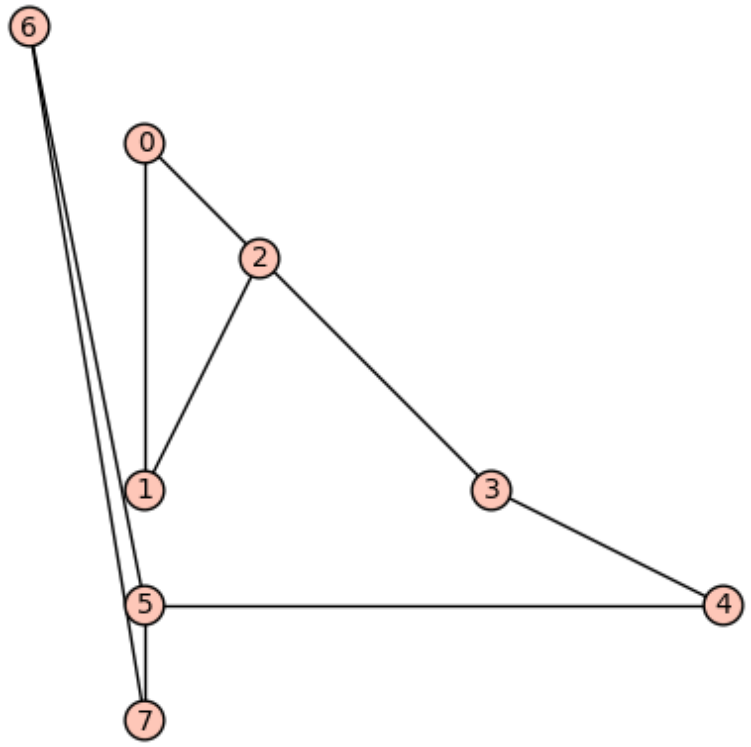
```
#Realiza el ejercicio en esta celda
#Representamos el grafo y mostramos si es plano o no
def isPlanoShowGrafo(g):    #Esta función muestra el grafo y dice si es plano o
no
    repMap = false
    print "Representacion del grafo: ";g.show()    #Muestra el grafo
    if g.is_planar():    #Comprobamos si el grafo es plano
        print "El grafo es plano"
        repMap = true
    else:
        print "El grafo no es plano"    #Mostramos este mensaje en caso de no
ser plano y se termine la ejecucion del ejercicio
    return repMap

isPlanar = isPlanoShowGrafo(grafo1)    #Llamamos a la funcion creada
recientemente pasándole el grafo como parámetro, dicha función devuelve un valor
```

```
Representacion del grafo:
El grafo es plano
```



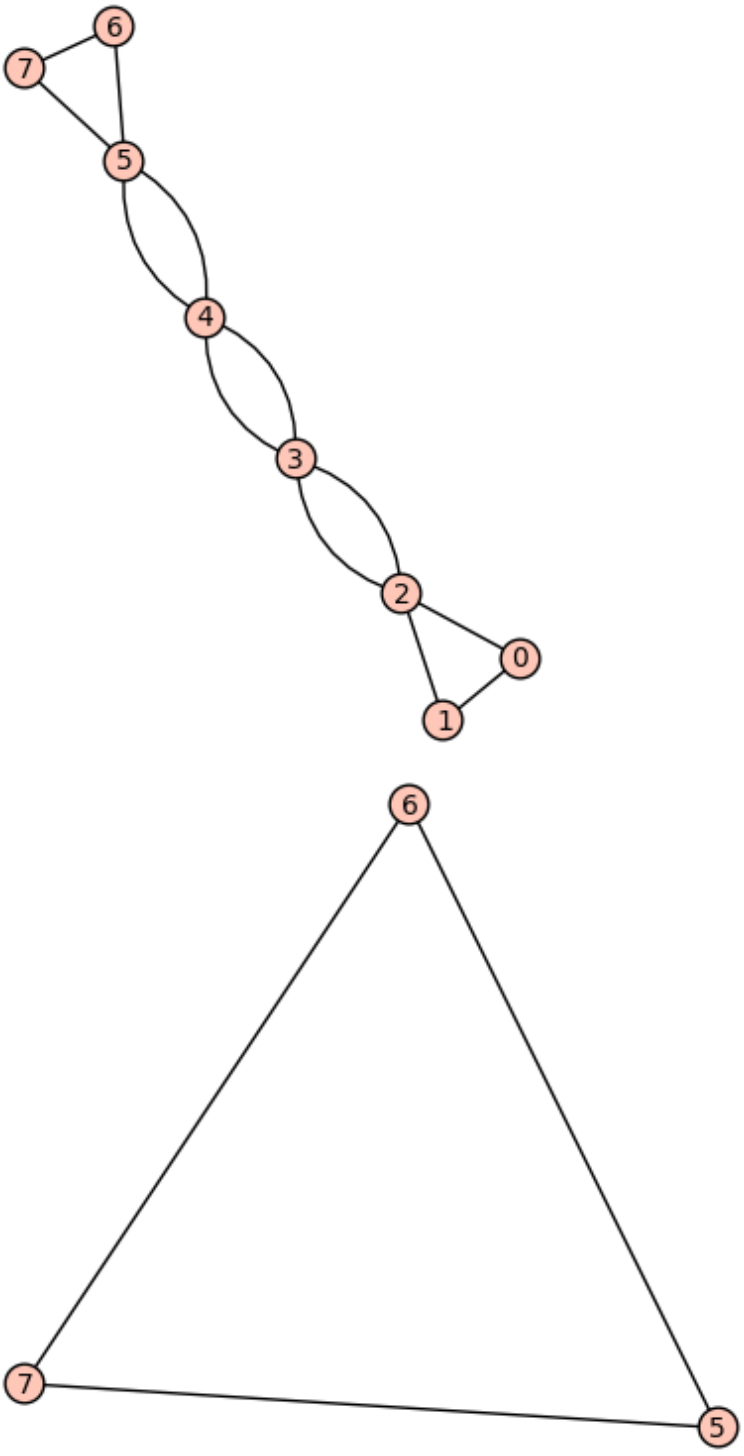

```
#Representamos el mapa
def getRegionesShowGrad(g):    #Funcion que recibe un grafo y muestra el mapa
siempre y cuando el grafo sea plano
    showMap = false
    if isPlanar:    #Esta es la variable que almacena el retorno de la función
anterior, si es True representamos el mapa
        print "Representacion del mapa: ";g.show(layout='planar')
        showMap = true
    else:
        print "No se puede mostrar el mapa porque el grafo no es plano"
    return showMap     #Retorna un valor booleano, True: si se muestra el mapa y
False en caso contrario

showMap = getRegionesShowGrad(grafo1)    #Variable que almacena el retorno de la
```

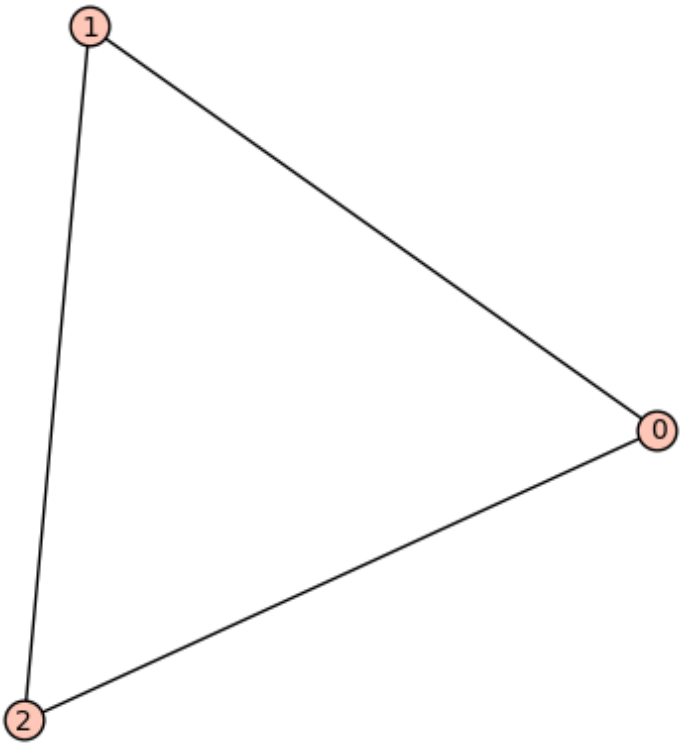
```
Representacion del mapa:
```




```
#Mostramos el grado de cada region
def getRegionesShowDegree(g):    #Funcion que devuelve el grado de cada una de
las regiones del grafo
    if showMap:
        nRegiones=len(g.faces())    #Almacenamos el valor de la longitud de la
lista de regiones para mostrar tantas regiones y grados como longitud haya.
        print "Numero de regiones: ",nRegiones
        for i in [0..nRegiones-1]:
            region = Graph(g.faces()[i])
            region.show()
            print "Grado de la region ", i+1,": ", region.num_edges()
#Incremento el contador i para que el print muestra primero el 1
    else:
        print "No se puede mostrar las regiones o los grados de las regiones
debido a que el grafo no representa un mapa"

getRegionesShowDegree(grafo1)    #llamamos a la función para mostrar las
```

```
Numero de regiones:  3
Grado de la region  1 :  12
Grado de la region  2 :  3
Grado de la region  3 :  3
```


6
7
5
4
3
2
0
1
6
7
5

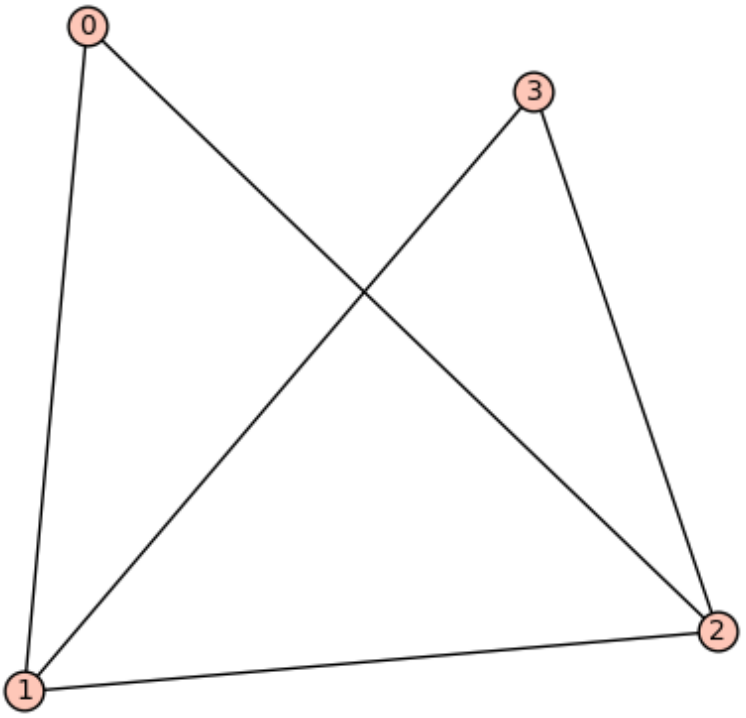


Ejercicio 2. Dado el grafo, representa el grafo y comprueba si se trata de un grafo plano. En caso afirmativo, representa el mapa e indica el número de regiones, así como su grado.

```
grafo2 = Graph({0:[1,2],1:[0,2,3],2:[0,1,3],3:[1,2]}); grafo2.set_pos({0:
[57,331],1:[33,77],2:[298,100],3:[228,306]});  #graph_editor(grafo2);
```

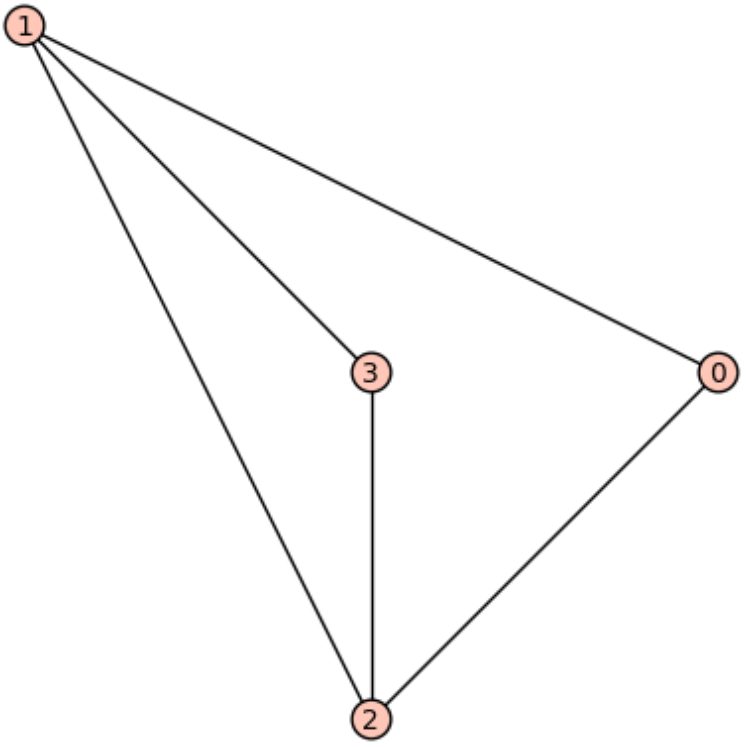
```
#Realiza el ejercicio en esta celda
#Representamos el grafo y mostramos si es plano o no
isPlanar = isPlanoShowGrafo(grafo2)
```

```
Representacion del grafo:
El grafo es plano
```




```
#Representamos el mapa
showMap = getRegionesShowGrad(grafo2)
```

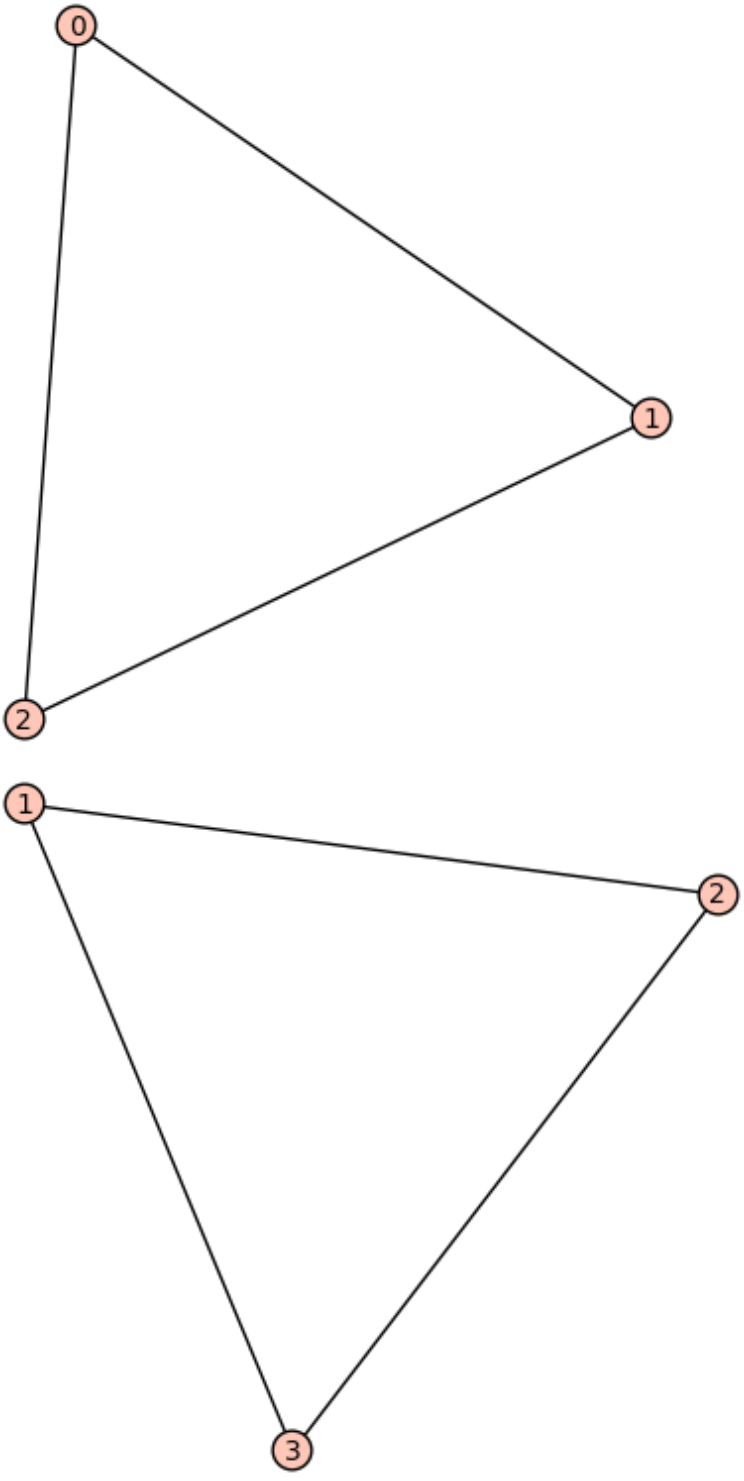
Representacion del mapa:

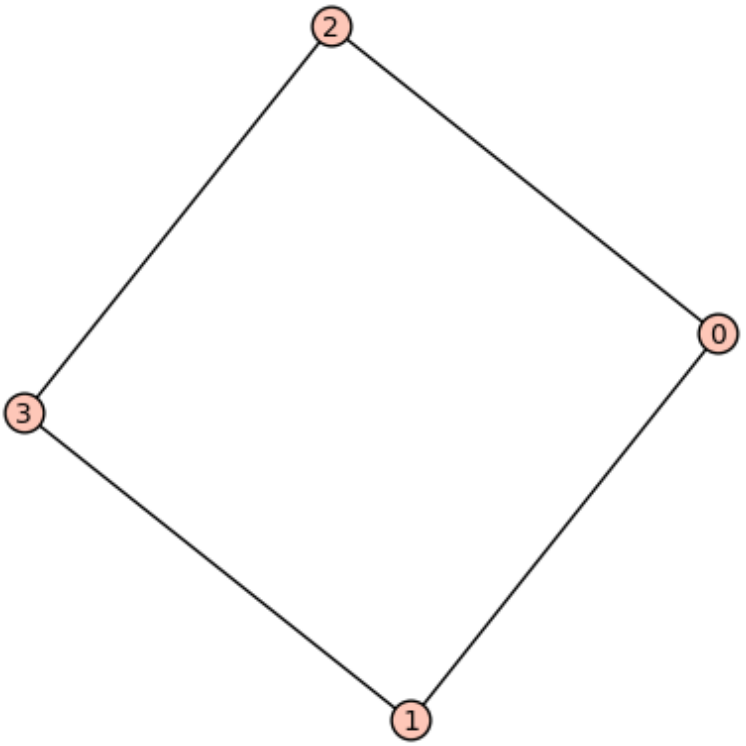



```
#Mostramos el grado de cada region
getRegionesShowDegree(grafo2)
```

Numero de regiones: 3
Grado de la region 1 : 3

```
Grado de la region  2 :  3
Grado de la region  3 :  4
```



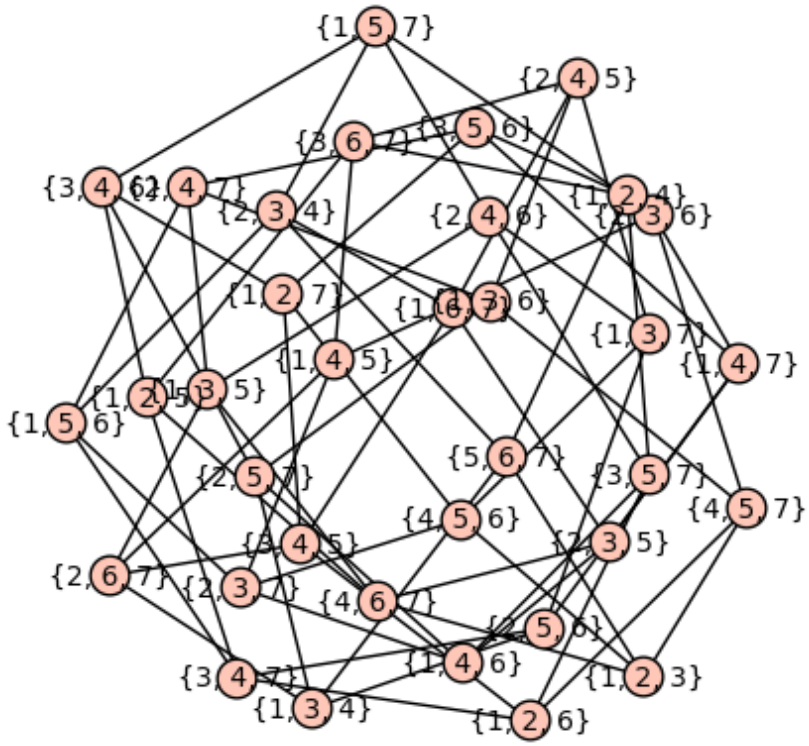


Ejercicio 3. Dado el grafo, representa el grafo y comprueba si se trata de un grafo plano. En caso afirmativo, representa el mapa e indica el número de regiones, así como su grado.

```
grafo3 = graphs.OddGraph(4)
```

```
#Realiza el ejercicio en esta celda
#Representamos el grafo y mostramos si es plano o no
isPlanar = isPlanoShowGrafo(grafo3)
```

```
Representacion del grafo:
El grafo no es plano
```



```
#Representamos el mapa
showMap = getRegionesShowGrad(grafo3)    #No nos muestra el mapa porque el grafo
no es plano
```

```
No se puede mostrar el mapa porque el grafo no es plano
```

```
#Mostramos el grado de cada region
getRegionesShowDegree(grafo3)    #No nos muestra regiones debido a que no existe
el mapa
```

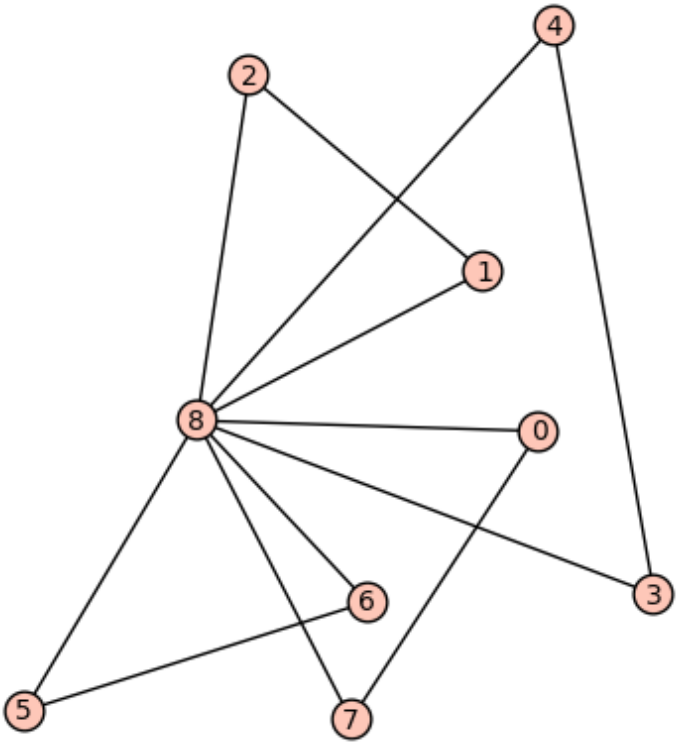
```
No se puede mostrar las regiones o los grados de las regiones debido a
que el grafo no representa un mapa
```

Ejercicio 4. Dado el grafo, representa el grafo y comprueba si se trata de un grafo plano. En caso afirmativo, representa el mapa e indica el número de regiones, así como su grado.

```
grafo4 = Graph({0:[7,8],1:[2,8],2:[1,8],3:[4,8],4:[3,8],5:[6,8],6:[5,8],7:
[0,8],8:[0,1,2,3,4,5,6,7]}); grafo4.set_pos({0:
[257.48834872229406,180.15030524730759],1:[236,242],2:
[145.26264270704667,317.8052442385753],3:[302,117],4:[263,337],5:
[58.300261358432344,71.82469112907705],6:[191,114],7:
```

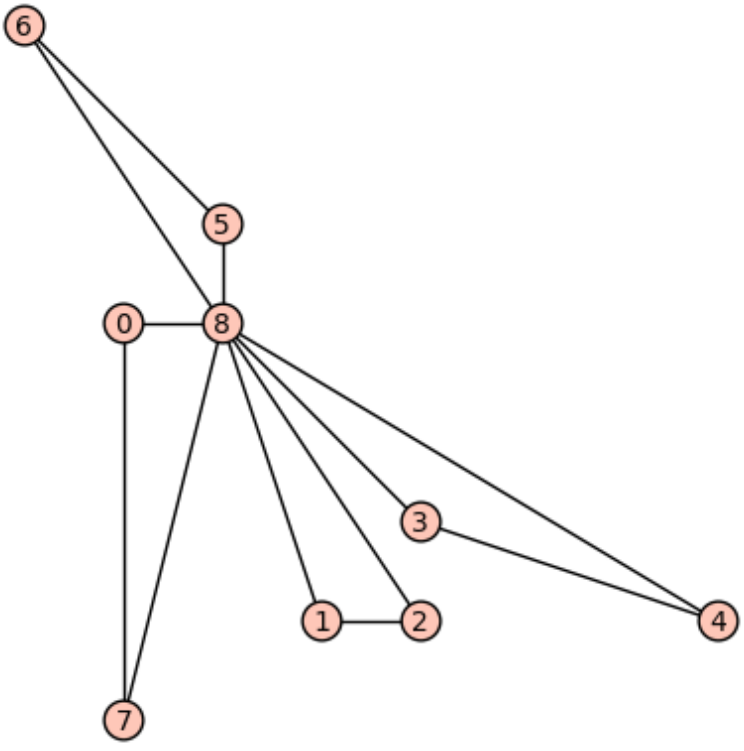
```
#Realiza el ejercicio en esta celda
#Representamos el grafo y mostramos si es plano o no
isPlanar = isPlanoShowGrafo(grafo4)
```

Representacion del grafo:
El grafo es plano



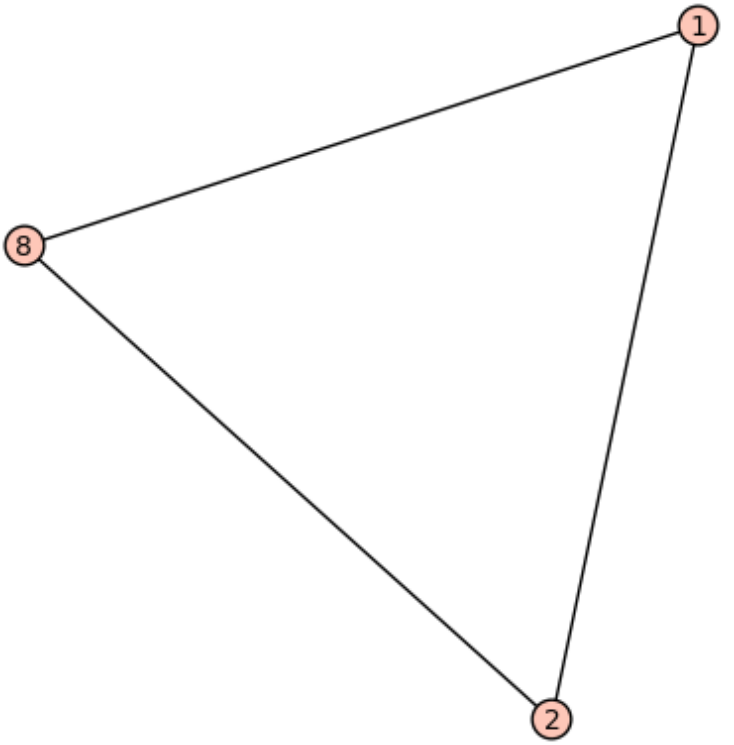

```
#Representamos el mapa
showMap = getRegionesShowGrad(grafo4)
```

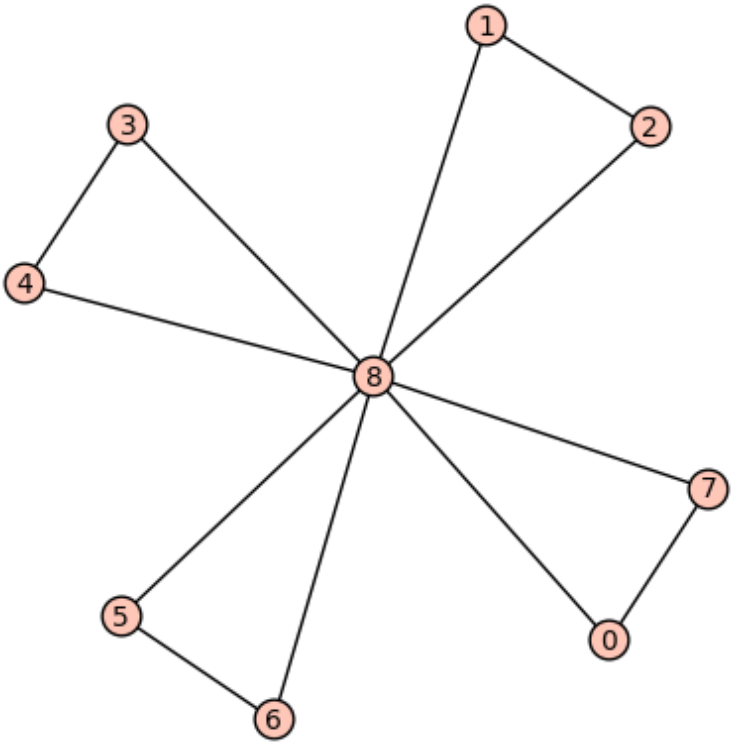
Representacion del mapa:

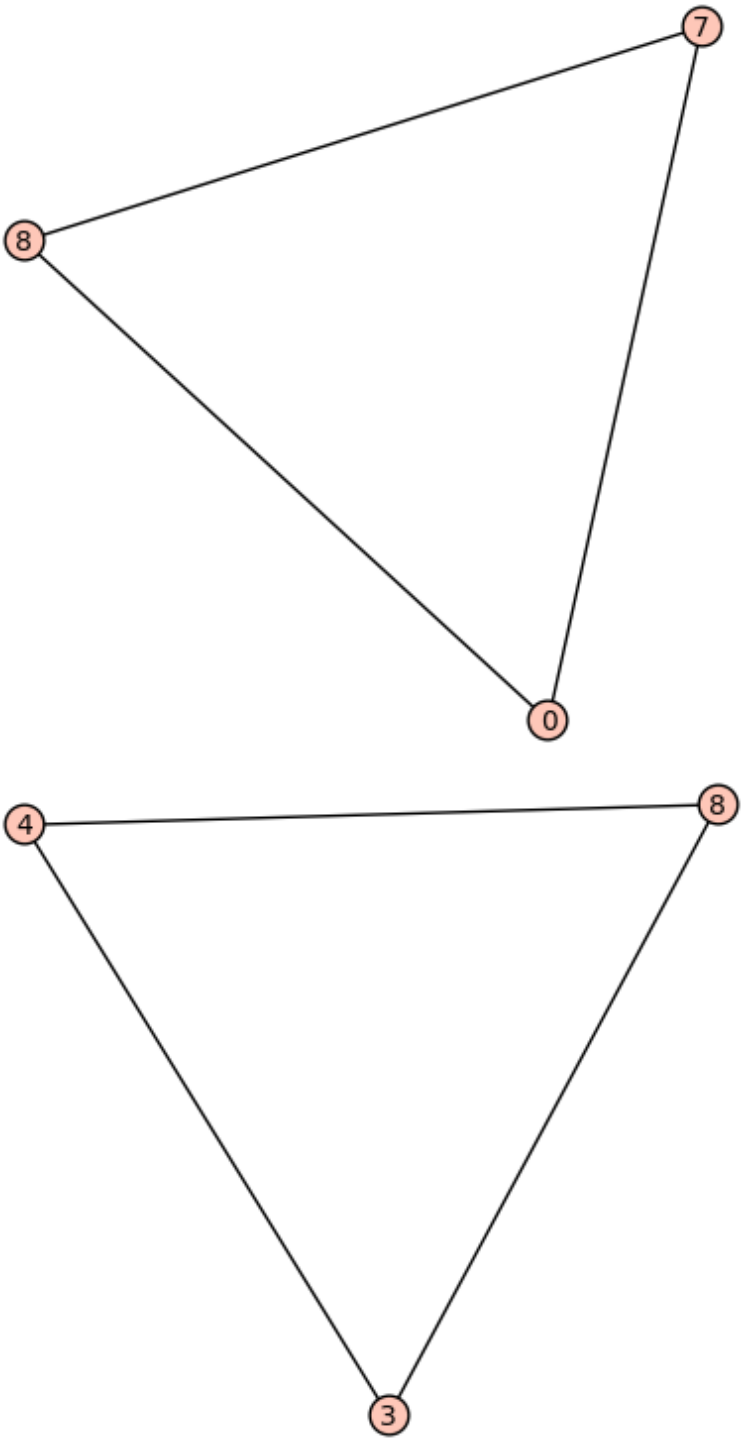


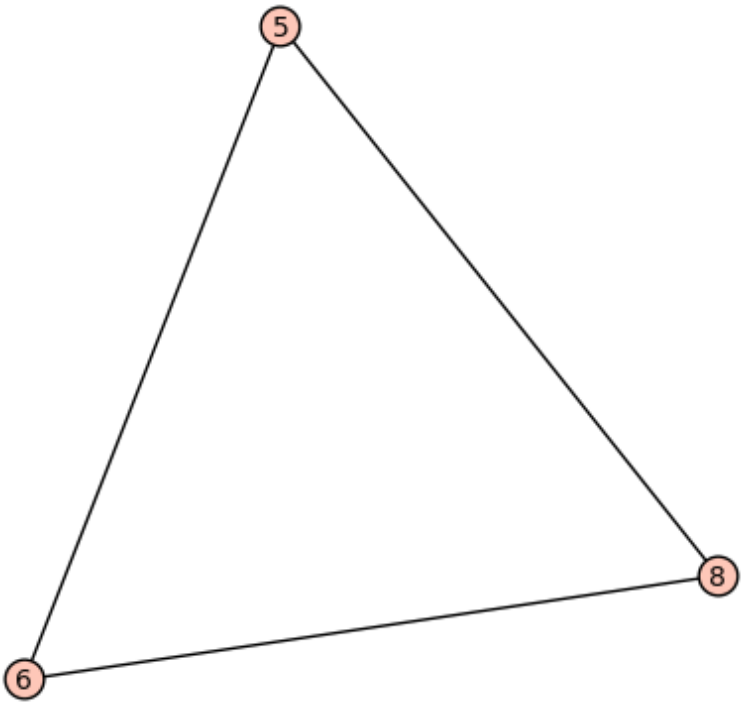
```
#Mostramos el grado de cada region
getRegionesShowDegree(grafo4)
```

```
Numero de regiones:  5
Grado de la region  1 :  3
Grado de la region  2 :  12
Grado de la region  3 :  3
Grado de la region  4 :  3
Grado de la region  5 :  3
```






7
8
0
4
8
3



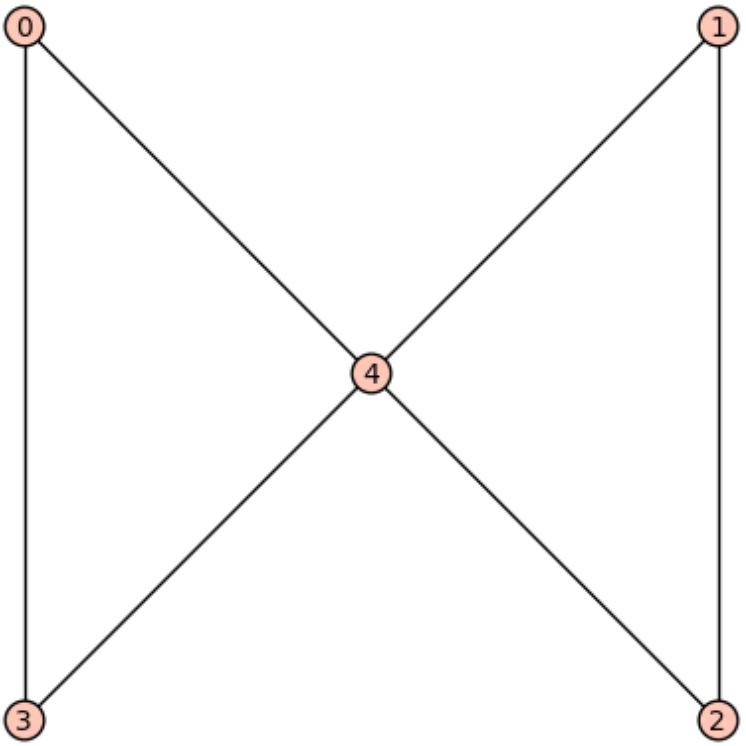
Coloraciones

Recordemos que para una paleta de colores C, una coloración por vértices de un grafo G con k colores es una aplicación

$$\gamma : V \to C$$

tal que si dos vértices $u, v \in V$ son adyacentes, entonces $\gamma(u) \neq \gamma(v)$. En Sage, podemos obtener una coloración por vértices de un grafo utilizando la función ***.coloring()*** que agrupa los vértices que tendrían el mismo color y devuelve una lista de vértices.

```
grafo = graphs.ButterflyGraph(); grafo.show()
```




```
grafo.coloring()
```

```
    [[3, 2], [0, 1], [4]]
```

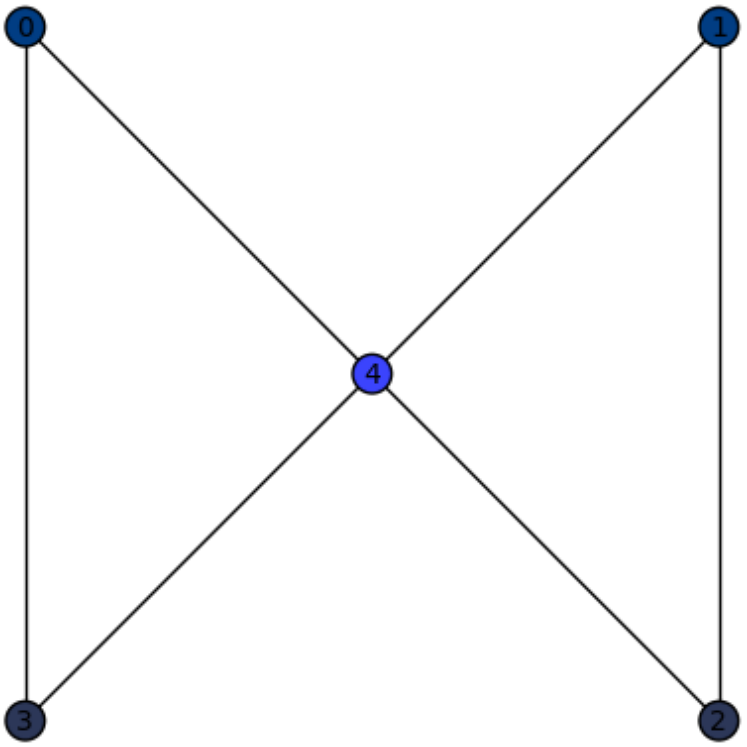
Si observamos para el grafo considerado, al utilizar la función de coloración, hemos obtenido una lista de vértices, en las que los vértices 3 y 2 tendrán el mismo color, así como los vértices 0 y 1, y distinto al vértice 4.

Para colorear los vértices de un grafo, debemos crear una lista de colores en la que se asigne cada color a un conjunto de los vértices de igual color obtenidos en la lista anterior.

```
vert_colores = grafo.coloring()
colores = {(random(),random(),random()):vertices for vertices in vert_colores}
```

Finalmente, para representar el grafo coloreado por vértices se utiliza el argumento ***vertex_colors*** = ***c***, siendo c la lista de colores generada, en la función *.show()*.

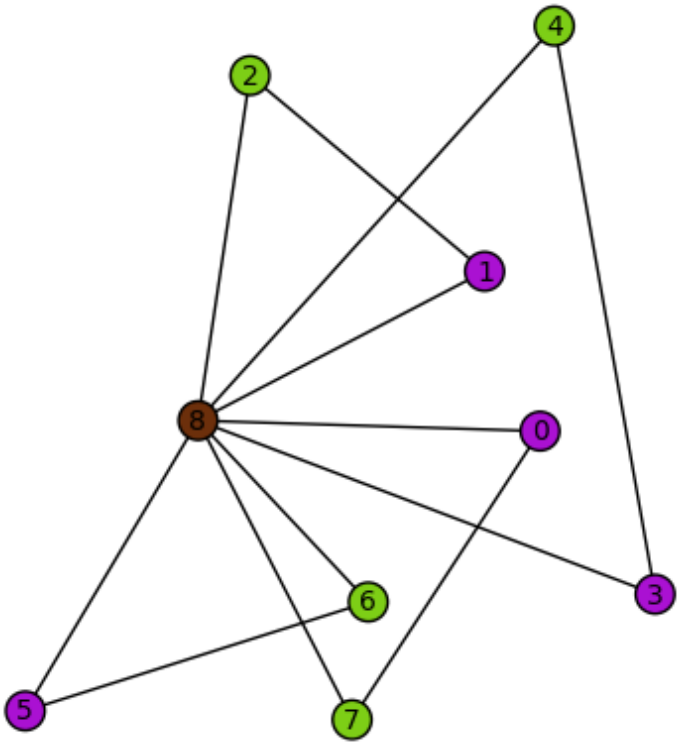
```
grafo.show(vertex_colors=colores)
```



Ejercicio 5. Considerando uno de los grafos de los ejercicios anteriores, representa una coloración por vértices de él.

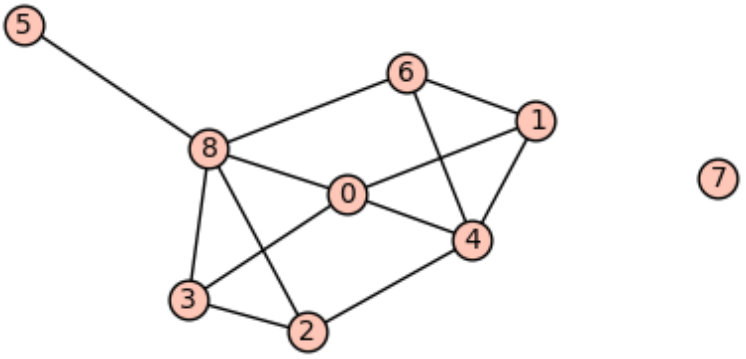
```
#Realiza el ejercicio en esta celda
def coloracion(g):
    vertices_color = g.coloring()    #Primero obtenemos la lista de vertices.
    print vertices_color    #Mostramos la lista de vertices
    colores = {(random(),random(),random()): vertices for vertices in
vertices_color}    #Obtenemos los distintos colores
    g.show(vertex_colors=colores)    #Mostramos el grafo coloreado

coloracion(grafo4)    #Llamamos a la funcion pasándole como parámetro el grafo a
```

```
[[7, 2, 4, 6], [0, 1, 3, 5], [8]]
```



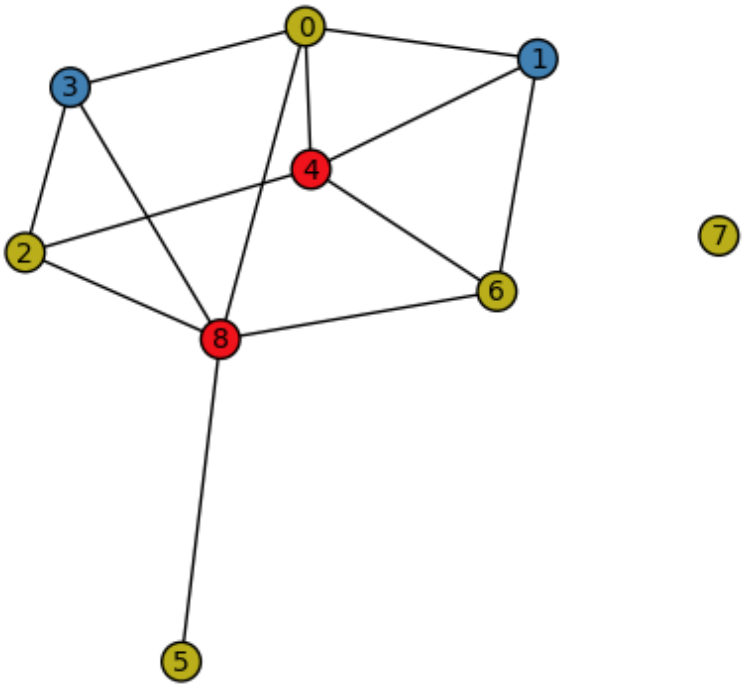
Ejercicio 6. Genera un grafo conexo con 9 vértices y 13 aristas. Representa dicho grafo y determina (y representa) una coloración por vértices de dicho grafo.

```
#Realiza el ejercicio en esta celda
grafo = graphs.RandomGNM(9,13)
grafo.show() #Representamos el grafo sin coloracion
```




```
coloracion(grafo)    #Representamos el grafo con coloracion
```

```
[[1, 3], [0, 6, 2, 5, 7], [4, 8]]
```


0
1
3
4
2
7
6
8
5