malabares

Release 4.6

Pablo Angulo

January 22, 2011

CONTENTS

CHAPTER

ONE

MALABARES Y TEORIA DE GRAFOS

1.1 Grafo de estados de k bolas y altura h

1.1.1 Hipótesis:

- El tiempo está compartimentado en instantes de igual longitud
- En cada instante el malabarista puede hacer a lo sumo una acción (recoger una bola y volverla a lanzar)
- En los instantes impares usa la mano izquierda y en los pares la derecha (se puede relajar)
- El malabarista nunca tiene más de una bola en una sóla mano (se puede relajar)

Nota: en un principio no hablamos de cómo empiezan los trucos, asumimos que los trucos ya están en marcha, con todas las pelotas en el aire de modo que en ningún instante futuro esté prevista la caída de dos bolas. Sin embargo, en todos los grafos que dibujaremos hay una forma sencilla de empezar.

1.1.2 Estado

Un instante está caracterizado por los tiempos esperados de caída de cada bola. Un estado sólo es válido si cada bola en el aire caerá en un instante futuro distinto. Representamos un estado mediante una secuencia de bola '*' y no bola '_'. En la posición i-ésima ponemos una bola si se espera que una bola caiga dentro de i instantes y no-bola si ninguna bola caerá dentro de i instantes. Por ejemplo `'**__*'` quiere decir que hay tres bolas en el aire y que caerán dentro de 1, 2 y 5 instantes.

Nota: Al estado en el que las bolas caerán en los instantes sucesivos inmediatos, lo denominamos **estado fundamental** (`'***_'` , ó `'***__'` , ó lo que toque).

1.1.3 Altura

La fuerza y la precisión del malabarista limitan el número máximo de instantes en el que futuro al que puede enviar una bola. A este número lo llamamos **altura** .

Hay una cantidad finita de estados posibles con k bolas y una altura h. Concretamente, cada una de las k bolas caerá en un instante entre 1 y h que es distinto para cada bola, luego se trata de elegir k números entre 1 y h, y la cantidad de formas de hacer esta elección es $\binom{h}{k}$

```
sage: bolas = 3
sage: altura = 4
sage: estados = [ ''.join(('*' if j in ss else '_') for j in range(altura))
...               for ss in Subsets(range(altura), bolas) ]
```

```
sage: estados
['***_', '**_*', '*_**', '_***']
```

1.1.4 Transiciones

¿A qué estados podemos cambiar partiendo de un estado dado? En el instante siguiente todas las bolas están un instante más próximas a caer en la mano del malabarista, luego pasamos por ejemplo de `'_*_**'` a `'*_**_'` , pero si teníamos una bola a punto de caer, el malabarista la recibe y la lanza de forma que caiga dentro de un cierto número de instantes de tiempo, y puede elegir cualquier momento futuro en el que no esté prevista la caída de ninguna otra bola. Por ejemplo, del estado `'*_**_'` podemos pasar a `'**__'`, `'_** * _'` o `'_**_ * '`.

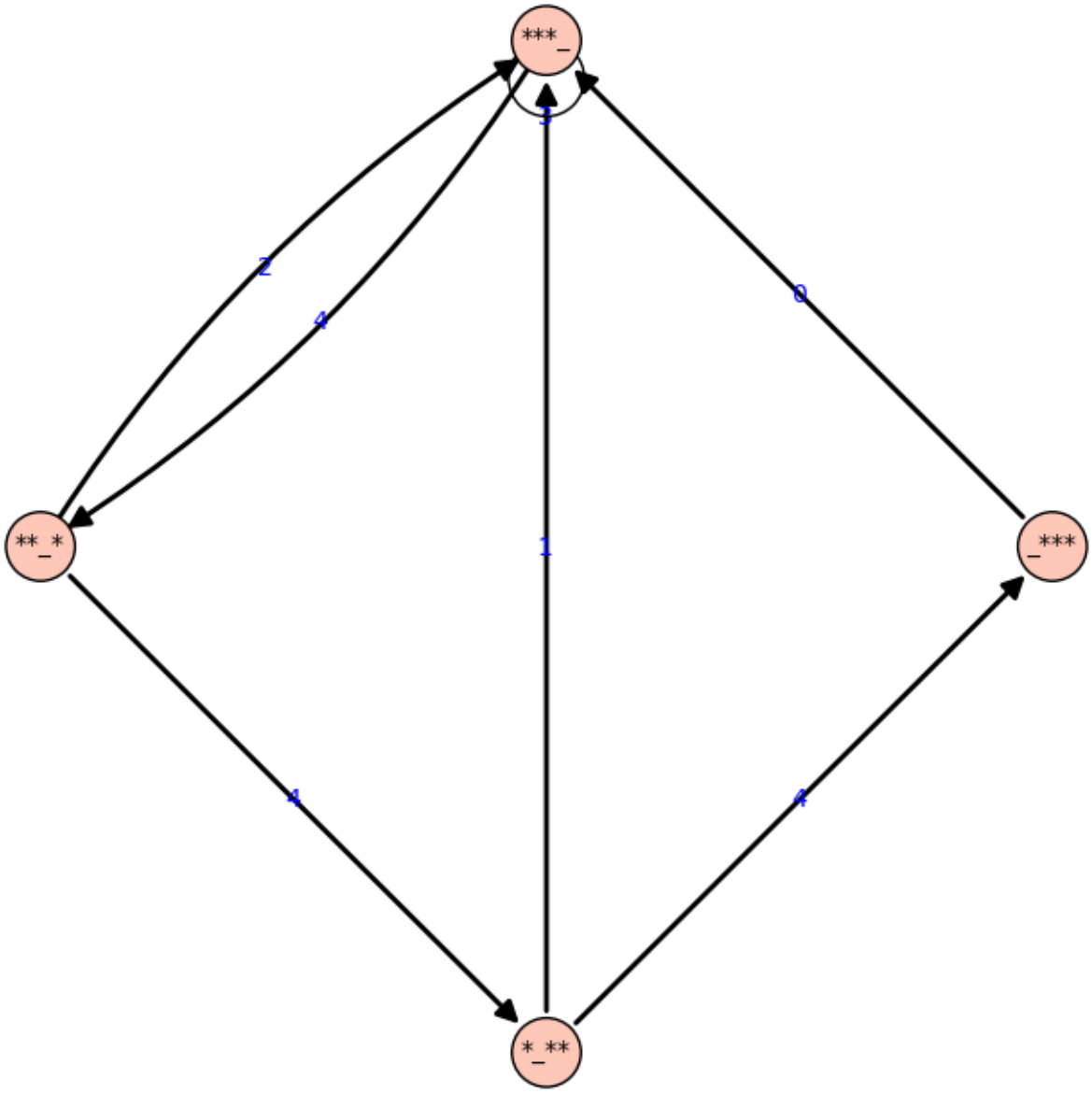
Junto a cada posible transición, guardamos la altura a la que lanzamos la bola (usando un 0 cuando no hacemos nada).

```
sage: def opciones(estado):
...       pasa = estado[1:] + '_'
...       if estado[0]=='_': return {pasa:0}
...       opciones = {}
...       for j,s in enumerate(pasa):
...           if s=='_':
...               opciones[pasa[:j] + '*'+ pasa[j+1:]] = j + 1
...       return opciones
sage: transiciones = dict((estado,opciones(estado)) for estado in estados)
sage: transiciones
{'***_': {'***_': 3, '**_*': 4}, '_***': {'***_': 0}, '*_**': {'***_': 1, '_***': 4}, '**_*': {'***_'
```

```
sage: def grafo_malabar(bolas, altura):
...       '''Crea el grafo de malabares con numero de bolas
...       y altura dadas'''
...       estados = [ ''.join(('*' if j in ss else '_') for j in range(altura))
...                   for ss in Subsets(range(altura), bolas) ]
...
...       transiciones = dict((estado,opciones(estado)) for estado in estados)
...       return DiGraph(transiciones)
```

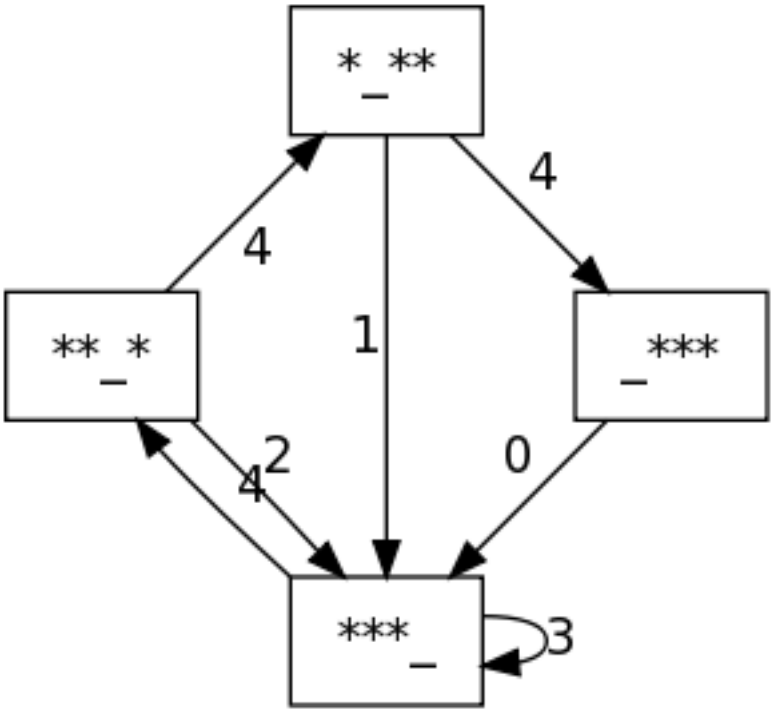
Dibujamos el grafo

```
sage: g = grafo_malabar(3,4)
sage: g.show(edge_labels=True, layout='circular', vertex_size=300, figsize=8)
```

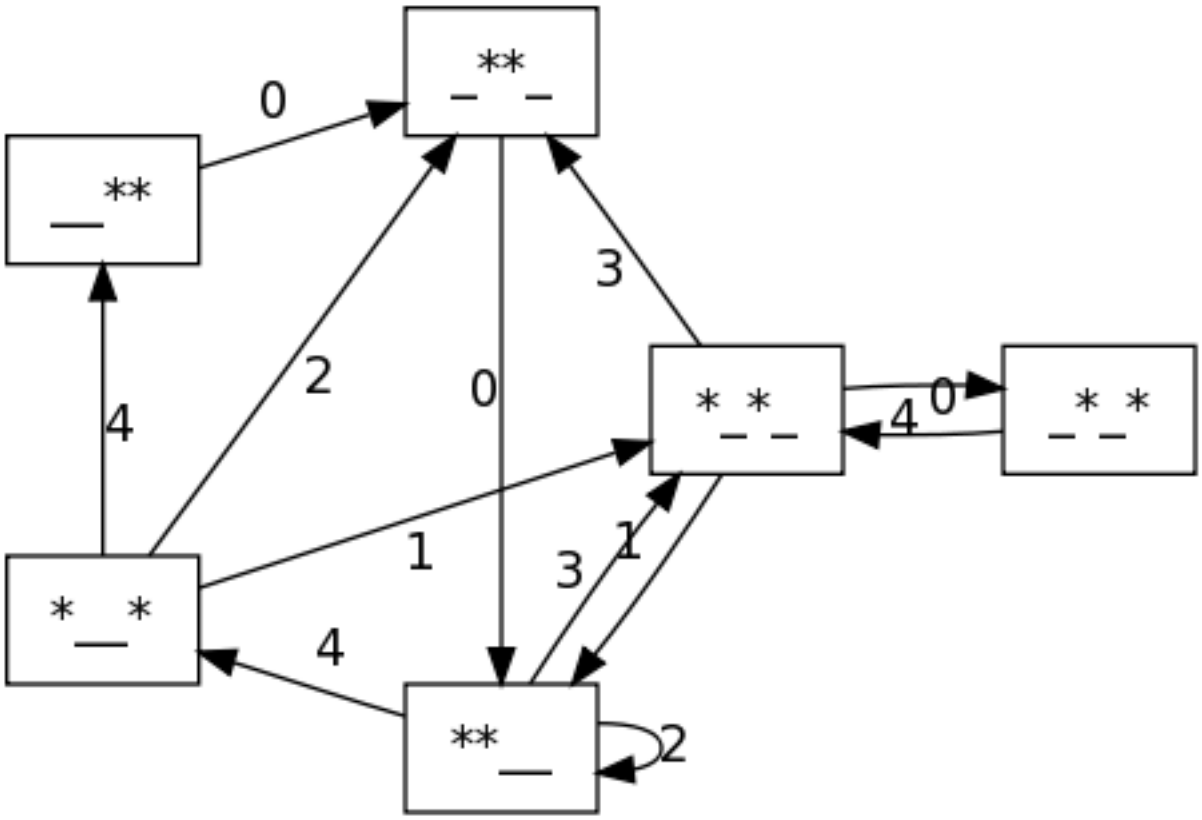


También dibujaremos los diagramas con graphviz, porque las etiquetas se ven más claras (graphviz debe estar instalado).

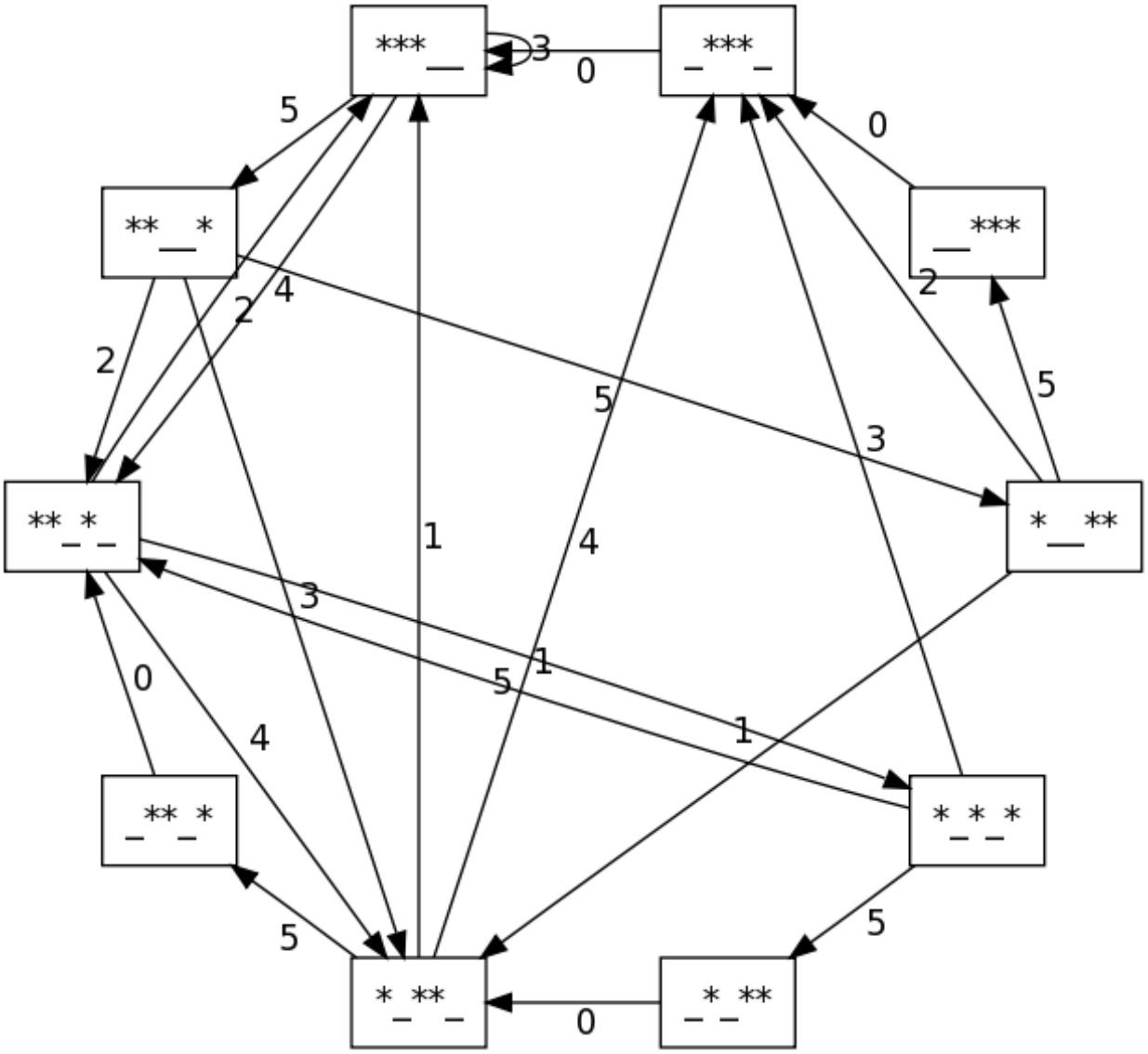
```
sage: attach(DATA + 'graphviz.sage')
sage: graphviz(grafo_malabar(3,4))
```




```
sage: graphviz(grafo_malabar(2,4))
```



```
sage: graphviz(grafo_malabar(3,5))
```



1.2 ¿Qué ventajas ofrece la teoría de grafos sobre el site swap?

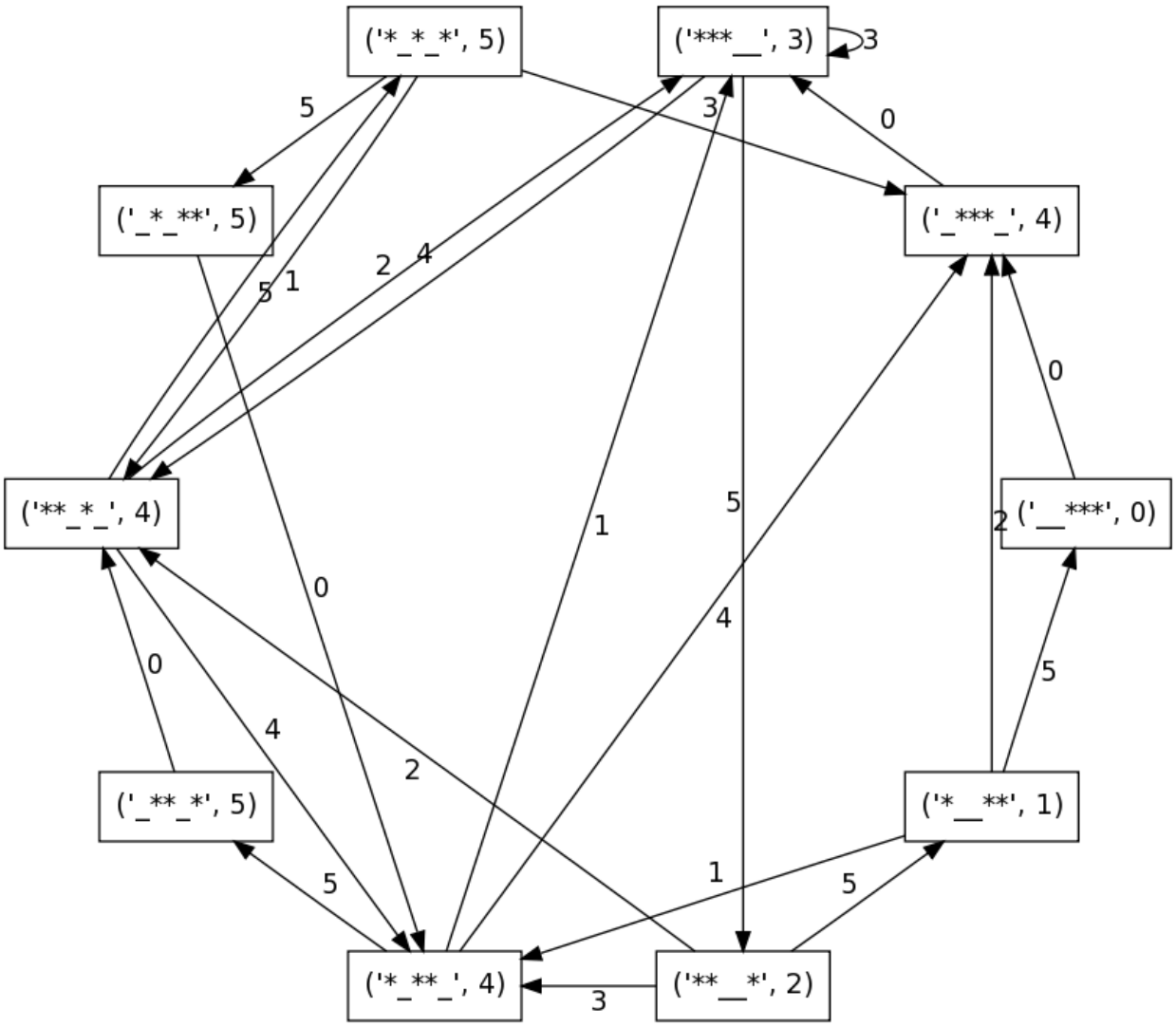
En pocas palabras, los grafos permiten representar situaciones muy diversas de forma muy uniforme.

Requerimientos del espectáculo de lo más variopinto se pueden traducir en *restricciones sobre el grafo* de posibles malabares.

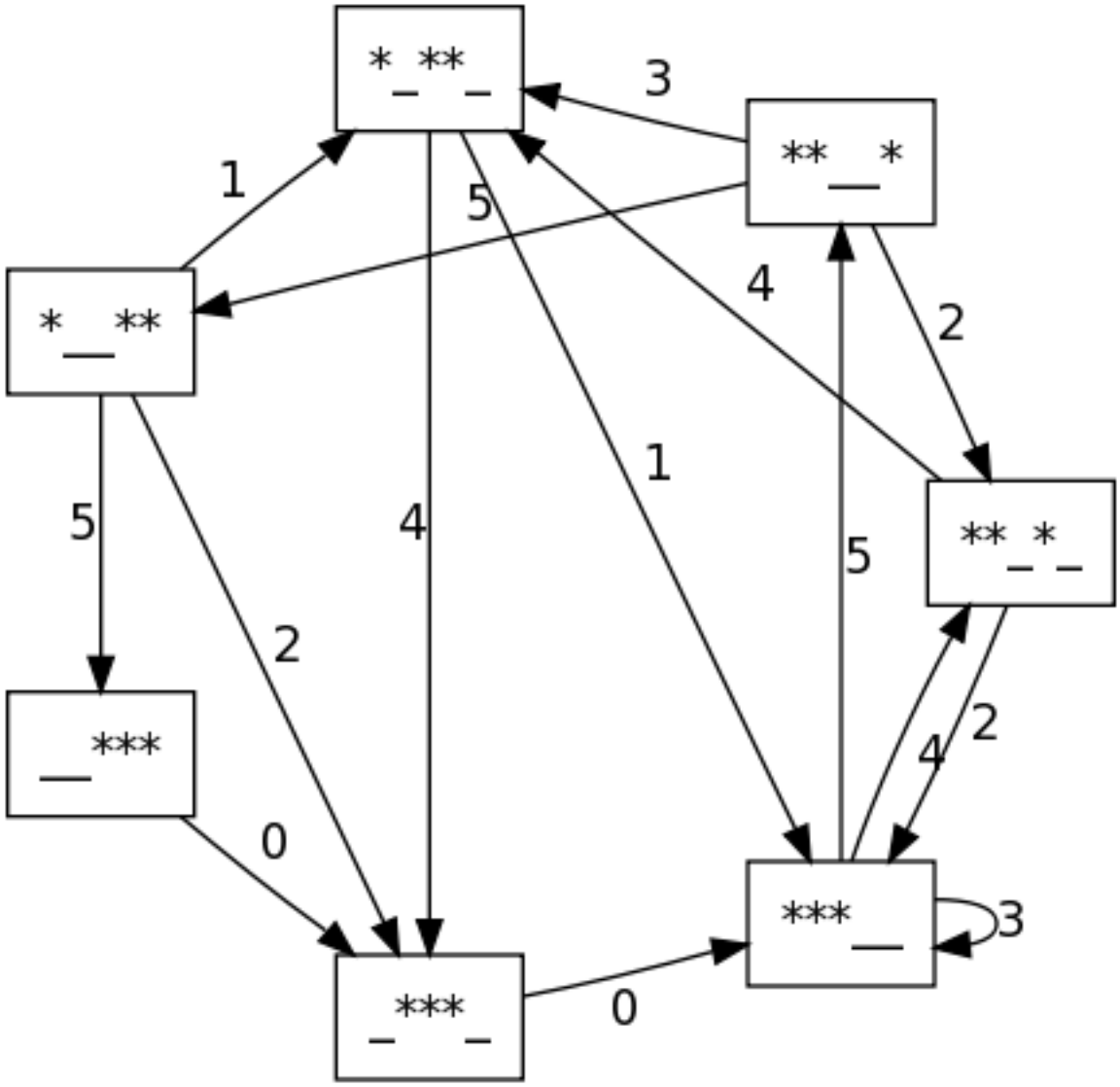
1.2.1 Ejemplo práctico

Queremos estar *a distancia menor o igual que 4* del estado `'___***'`, por ejemplo porque ese estado nos da tiempo para hacer un truco que hemos preparado, o para saludar a un viandante que ha echado una moneda en el plato.

```
sage: g = grafo_malabar(3,5)
sage: ds = g.distance_all_pairs()
sage: v0 = '__***'
sage: new_labels = {}
sage: for v in g:
...       new_labels[v] = (v, ds[v][v0])
sage: g.relabel(new_labels)
sage: graphviz(g)
```




```
sage: g = grafo_malabar(3,5)
sage: ds = g.distance_all_pairs()
sage: v0 = '__***'
sage: vs0 = [v for v in g if ds[v][v0]<=4]
sage: sg = g.subgraph(vs0)
sage: graphviz(sg)
```



1.2.2 Otro ejemplo: poner una pelota en la cabeza

Ahora podemos *dejar una pelota en la cabeza* por tiempo indefinido. Representamos por un '8' una cabeza con una pelota encima, y por una 'o' una cabeza sin pelota encima.

Por ejemplo, para tres pelotas y altura 4, los posibles estados tienen dos pelotas en el aire y una en la cabeza, o tres pelotas en el aire y ninguna en la cabeza.

```
sage: bolas = 3
sage: altura = 4
sage: estados_con_cabeza = ([('o' + ''.join(('*' if j in ss else '_')
...                                         for j in range(altura)))
...                          for ss in Subsets(range(altura), bolas) ] +
...                         [('8' + ''.join(('*' if j in ss else '_')
...                                         for j in range(altura)))
...                          for ss in Subsets(range(altura), bolas-1) ])
```

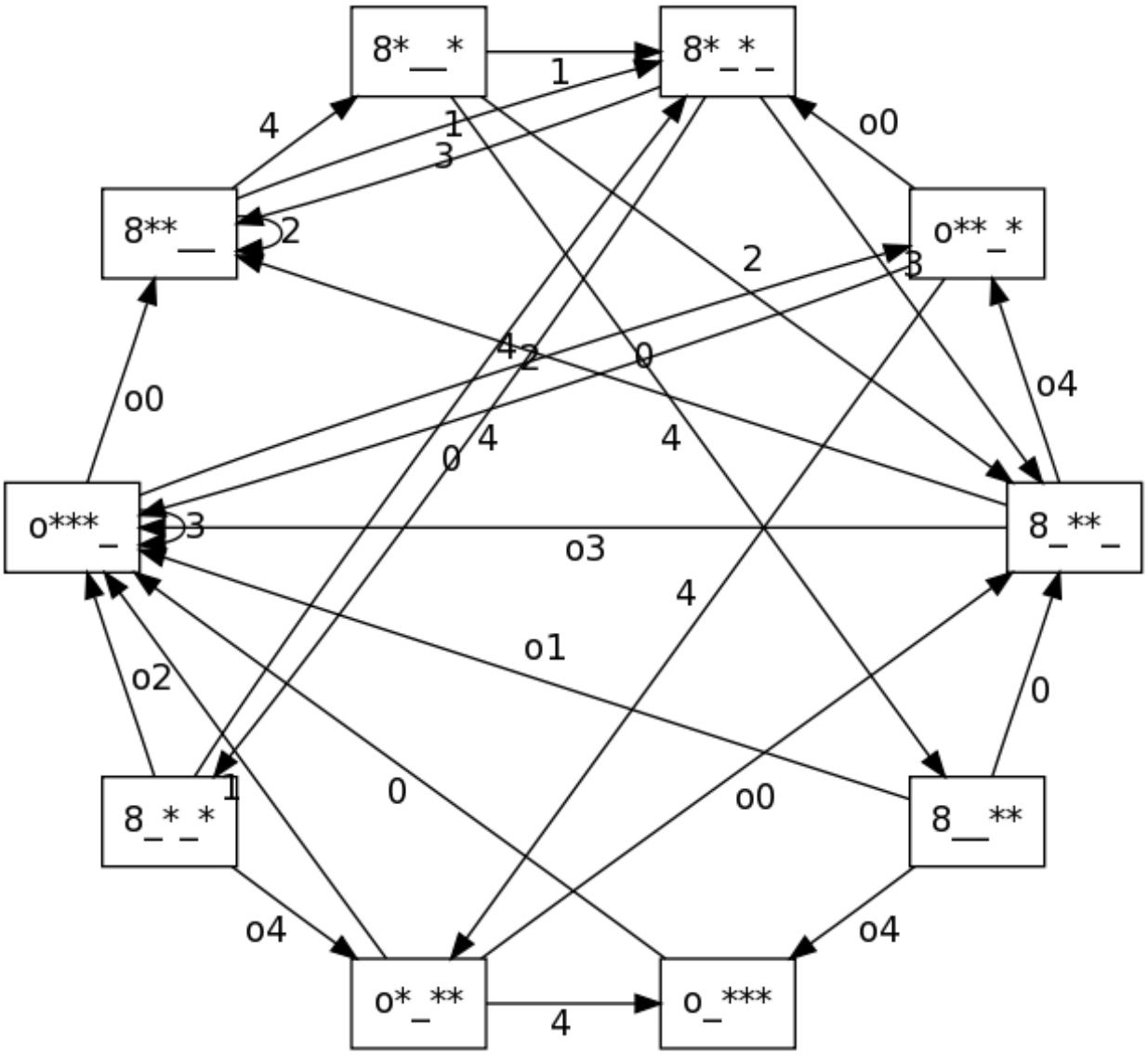
```
sage: estados_con_cabeza
['o***_', 'o**_*', 'o*_**', 'o_***', '8**__', '8*_*_', '8*__*', '8_**_', '8_*_*', '8__**']
```

1.2.3 Transiciones con cabeza

Ahora tenemos opciones nuevas para pasar de un estado a otro:

- Si tenemos la mano vacía y una pelota en la cabeza, podemos cogerla y lanzarla.
- Si tenemos una pelota en la mano y ninguna en la cabeza, podemos dejar la pelota en la cabeza.
- Si tenemos una pelota en la mano y otra en la cabeza, tenemos que lanzar la pelota como de costumbre.

```
sage: def opciones_con_cabeza(estado):
...       cabeza = estado[0]
...       mano = estado[1]
...       bolas1 = estado[2:] + '_'
...       opciones = {}
...       if mano=='_' and cabeza=='o':
...           opciones = {('o' + bolas1):'0'}
...       elif mano=='_': #and cabeza=='8'
...           opciones['8' + bolas1] = '0'
...           for j,s in enumerate(bolas1):
...               if s=='_':
...                   opciones[('o' + bolas1[:j] +
...                             '*'+ bolas1[j+1:])] = 'o%d'%(j + 1)
...       elif cabeza=='8': #and mano=='*'
...           for j,s in enumerate(bolas1):
...               if s=='_':
...                   opciones[('8' + bolas1[:j] +
...                             '*'+ bolas1[j+1:])] = '%d'%(j + 1)
...       else: #cabeza=='o': #and mano=='*'
...           opciones['8' + bolas1] = 'o0'
...           for j,s in enumerate(bolas1):
...               if s=='_':
...                   opciones[('o' + bolas1[:j] +
...                             '*'+ bolas1[j+1:])] = '%d'%(j + 1)
...       return opciones
sage: def grafo_malabar_con_cabeza(bolas, altura):
...       estados_con_cabeza = ([('o' + ''.join(('*' if j in ss else '_')
...                                             for j in range(altura)))
...                              for ss in Subsets(range(altura), bolas) ] +
...                             [('8' + ''.join(('*' if j in ss else '_')
...                                             for j in range(altura)))
...                              for ss in Subsets(range(altura), bolas-1) ])
...       transiciones = dict((estado,opciones_con_cabeza(estado))
...                           for estado in estados_con_cabeza)
...       g = DiGraph(transiciones)
...       return g

sage: g = grafo_malabar_con_cabeza(3, 4)
sage: graphviz(g)
```



1.2.4 Y mucho más

- Bolas distintas (por su color o por su naturaleza).
- Varios malabaristas.
- ...

Más ejemplos en los ejercicios del final.

1.3 Propiedades de los grafos malabares

Y bien, ¿qué hacemos una vez tenemos un grafo? Para empezar, nos podemos preguntar si el grafo tiene algunas propiedades típicas de la teoría de grafos que pueden ser interesantes:

- ¿El grafo es **fuertemente conexo** ? Es decir, ¿se puede pasar de un estado a cualquier otro?

- ¿El grafo de malabares es **euleriano** ? Es decir, ¿existe un circuito que pasa por cada **arista** *exactamente una vez* ?
- ¿El grafo de malabares es **hamiltoniano** ? Es decir, ¿existe un circuito que pasa por cada **vértice** *exactamente una vez* ?

```
sage: for j in range(2,5):
...       for k in range(j+1,j+4):
...           print 'el grafo malabar con %d bolas y altura %d\
...     %ses fuertemente conexo'%(j,k,
...           '' if grafo_malabar(j,k).is_strongly_connected() else 'no ')
el grafo malabar con 2 bolas y altura 3 es fuertemente conexo
el grafo malabar con 2 bolas y altura 4 es fuertemente conexo
el grafo malabar con 2 bolas y altura 5 es fuertemente conexo
el grafo malabar con 3 bolas y altura 4 es fuertemente conexo
el grafo malabar con 3 bolas y altura 5 es fuertemente conexo
el grafo malabar con 3 bolas y altura 6 es fuertemente conexo
el grafo malabar con 4 bolas y altura 5 es fuertemente conexo
el grafo malabar con 4 bolas y altura 6 es fuertemente conexo
el grafo malabar con 4 bolas y altura 7 es fuertemente conexo
```

```
sage: for j in range(2,5):
...       for k in range(j+1,j+4):
...           print 'el grafo malabar con %d bolas y altura %d\
...     %ses euleriano'%(j,k,
...           '' if grafo_malabar(j,k).is_eulerian() else 'no ')
el grafo malabar con 2 bolas y altura 3 no es euleriano
el grafo malabar con 2 bolas y altura 4 no es euleriano
el grafo malabar con 2 bolas y altura 5 no es euleriano
el grafo malabar con 3 bolas y altura 4 no es euleriano
el grafo malabar con 3 bolas y altura 5 no es euleriano
el grafo malabar con 3 bolas y altura 6 no es euleriano
el grafo malabar con 4 bolas y altura 5 no es euleriano
el grafo malabar con 4 bolas y altura 6 no es euleriano
el grafo malabar con 4 bolas y altura 7 no es euleriano
```

```
sage: for j in range(2,5):
...       for k in range(j+1,j+4):
...           print 'el grafo malabar con %d bolas y altura %d\
...     %ses hamiltoniano'%(j,k,
...           '' if grafo_malabar(j,k).is_hamiltonian() else 'no ')
el grafo malabar con 2 bolas y altura 3 es hamiltoniano
el grafo malabar con 2 bolas y altura 4 no es hamiltoniano
el grafo malabar con 2 bolas y altura 5 no es hamiltoniano
el grafo malabar con 3 bolas y altura 4 es hamiltoniano
el grafo malabar con 3 bolas y altura 5 no es hamiltoniano
el grafo malabar con 3 bolas y altura 6 no es hamiltoniano
el grafo malabar con 4 bolas y altura 5 es hamiltoniano
el grafo malabar con 4 bolas y altura 6 no es hamiltoniano
el grafo malabar con 4 bolas y altura 7 no es hamiltoniano
```

```
sage: for j in range(2,5):
...       for k in range(j+1,j+4):
...           print 'el grafo malabar con %d bolas, altura %d y con cabeza\
...     %ses hamiltoniano'%(j,k,
...           '' if grafo_malabar_con_cabeza(j,k).is_hamiltonian() else 'no ')
```

```
el grafo malabar con 2 bolas, altura 3 y con cabeza es hamiltoniano
el grafo malabar con 2 bolas, altura 4 y con cabeza es hamiltoniano
el grafo malabar con 2 bolas, altura 5 y con cabeza es hamiltoniano
el grafo malabar con 3 bolas, altura 4 y con cabeza es hamiltoniano
el grafo malabar con 3 bolas, altura 5 y con cabeza es hamiltoniano
el grafo malabar con 3 bolas, altura 6 y con cabeza es hamiltoniano
el grafo malabar con 4 bolas, altura 5 y con cabeza es hamiltoniano
el grafo malabar con 4 bolas, altura 6 y con cabeza es hamiltoniano
el grafo malabar con 4 bolas, altura 7 y con cabeza es hamiltoniano
```

1.4 Camino aleatorio en el grafo

Podemos movernos libremente sobre el grafo, sin necesidad de seguir uno de los patrones de siteswap como 441 o 531. Creamos a continuación una animación que cada segundo muestra el estado en que nos encontramos dentro del grafo y el número asociado a la arista que seguimos para llegar al siguiente estado (que es la altura a la que tenemos que lanzar la bola).

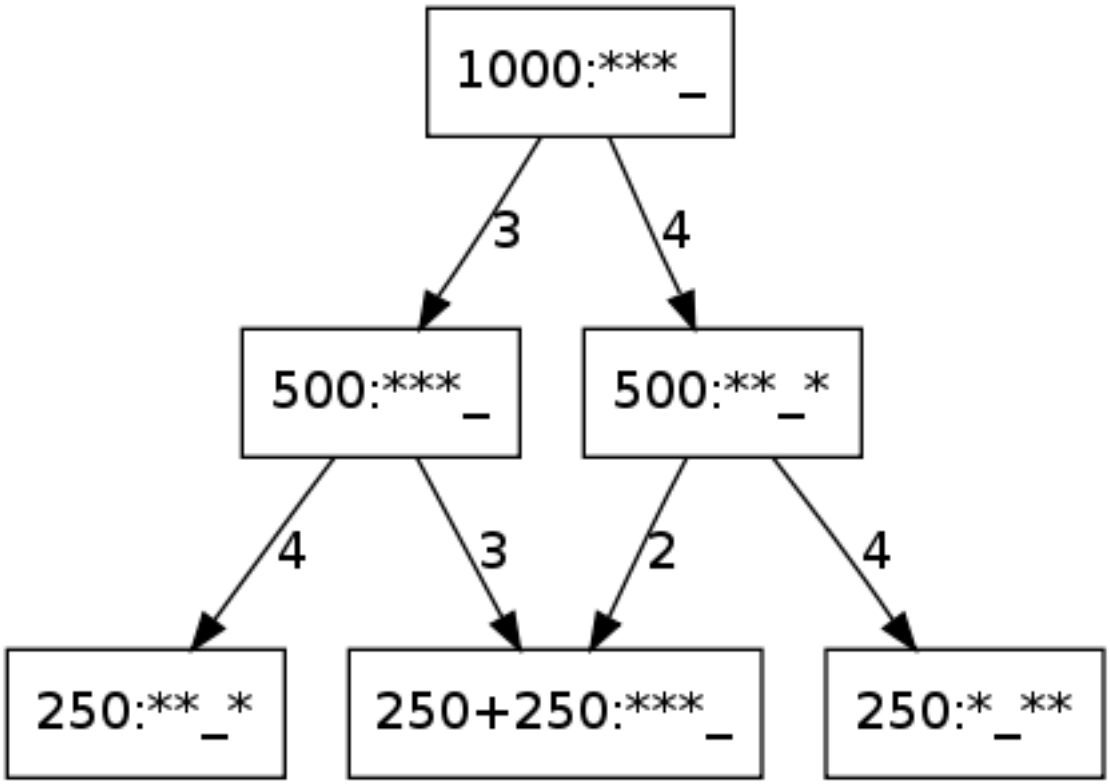
```
sage: def camino_aleatorio(g, estado_inicial, longitud=10):
...       vs = g.vertices()
...       vs.remove(estado_inicial)
...       ps = [g.plot(edge_labels=True, layout='circular',
...                    vertex_size = 400,
...                    vertex_colors={'red':[estado_inicial],
...                                   'green':vs})+
...             text('...', (0,0), fontsize=200)]
...       v0 = estado_inicial
...       for j in xrange(longitud):
...           v0,v1,altura = choice(g.outgoing_edges(v0))
...           if v1 != v0:
...               vs.remove(v1)
...               vs.append(v0)
...           ps.append(g.plot(edge_labels=True, layout='circular',
...                            vertex_size = 400,
...                            vertex_colors={'red':[v1],
...                                           'green':vs}) +
...                     text(altura, (0,0), fontsize=200))
...           v0 = v1
...       return animate(ps, axes = False)


sage: g = grafo_malabar(3,5)
sage: #Partimos del estado fundamental
sage: a = camino_aleatorio(g, '***__', longitud = 10)
sage: a.show(delay = 200)
```

1.5 Cadena de Markov

Si cada vez que lanzamos una bola escogemos la altura entre las posibilidades viables *con igual probabilidad* , ¿cuánto tiempo pasaremos en cada estado?

Para empezar a entender el problema, trabajamos con 3 bolas y altura 4, y nos planteamos qué pasaría si repetimos el experimento 1000 veces, partiendo por ejemplo del estado fundamental `'***_'`. En el primer lanzamiento, podemos elegir entre hacer un lanzamiento a altura 3 ó 4, de modo que 500 veces estaremos en el estado `'***_'` y otras 500 en

el estado `'**_*'`. Si hacemos dos lanzamientos, tenemos dos decisiones, lo que da lugar a 4 caminos, y seguiremos cada uno de los cuatro en 250 ocasiones. Sin embargo, dos de esos caminos llevan al estado fundamental, luego en 500 ocasiones nos encontramos en `'***_'`, en 250 en `'**_*'` y en otras 250 en `'*_**'`.



Por supuesto, lo importante no es el número excato, sino la proporción sobre el número de lanzamientos (es decir, 1/2 de las veces nos encontramos en `'***_'`, 1/4 de las veces en `'**_*'` y el otro 1/4 de las veces en `'*_**'`.). La manera habitual de registrar el experimento anterior en matemáticas es guardar estas proporciones, o **probabilidades** , en un vector donde guardamos las probabilidades en el orden correspodiente a `['***_', '_*', '*_', '_***']`. En el ejemplo anterior, decimosque estamos en el estado [1/2, 1/4, 1/4, 0]. El estado inicial era el estado determinista [1,0,0,0], y el estado en el primer instante después de empezar era [1/2, 1/2, 0, 0].

Calculamos el vector de probabilidades hasta después de 10 instantes.

```
sage: g = grafo_malabar(3,4)
sage: print g.vertices()
sage: print 'Partiendo de un estado inicial [1,0,...0])'
sage: M = matrix([row/sum(row) for row in g.adjacency_matrix().rows()])
sage: estado = vector( [1]+[0]*(len(g.vertices())-1) )
sage: n3 = lambda x: x.n(digits = 3)
sage: for k in range(10):
...       print k, ':', map(n3, estado)
...       estado = estado*M
['***_', '**_*', '*_**', '_***']
Partiendo de un estado inicial [1,0,...0])
0 : [1.00, 0.000, 0.000, 0.000]
1 : [0.500, 0.500, 0.000, 0.000]
2 : [0.500, 0.250, 0.250, 0.000]
3 : [0.500, 0.250, 0.125, 0.125]
4 : [0.562, 0.250, 0.125, 0.0625]
5 : [0.531, 0.281, 0.125, 0.0625]
```

```
6 : [0.531, 0.266, 0.141, 0.0625]
7 : [0.531, 0.266, 0.133, 0.0703]
8 : [0.535, 0.266, 0.133, 0.0664]
9 : [0.533, 0.268, 0.133, 0.0664]
```

Observa que hemos usado un truco para calcular las probabilidades en un estado a partir de las probabilidades en el anterior: hemos construido una matriz con las probabilidades de transición a partir de cada estado. Para obtener las probabilidades en un estado a partir de las probabilidades en el anterior, multiplicamos el vector de probabilidades por la matriz de probabilidades de transición.

Esta forma de calcular las transiciones se basa en el **teorema de la probabilidad total** . La probabilidad de estar en el estado I en tiempo k+1 es igual a la suma para cada estado J del producto de la probabilidad del estado J en tiempo k por la probabilidad de transición de J a I:

$$P_{k+1}(I) = \sum_J P_k(J) P_{k \to k+1}(I|J)$$

escrito en forma vectorial:

$$P_{k+1} = P_k \cdot M$$

La matriz de probabilidades de transición se obtiene de forma sencilla a partir de la *matriz de adyacencia* del grafo, que tiene un 1 en la posición (i,j) si hay una flecha del vértice i-ésimo al j-ésimo, y 0 en otro caso.

```
sage: show(g.adjacency_matrix())
sage: M = matrix([row/sum(row) for row in g.adjacency_matrix().rows()])
sage: show(M)
```

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Otra cosa curiosa que observamos al calcular las probabilidades de transición es que parecen acercarse al vector [0.533, 0.267, 0.133, 0.0667], y que una vez se llega a ese vector las probabilidades ya no cambian. Para más inri, comprobamos que se llega al mismo vector *independientemente del estado inicial* .

```
sage: print 'Partiendo de un estado inicial [0,...0,1]'
sage: estado_inicial = vector( [0]*(len(g.vertices())-1) + [1] )
sage: print map(n3, estado_inicial*M)
sage: print map(n3, estado_inicial*M^2)
sage: print map(n3, estado_inicial*M^5)
sage: print map(n3, estado_inicial*M^10)
sage: print map(n3, estado_inicial*M^20)
Partiendo de un estado inicial [0,...0,1]
[1.00, 0.000, 0.000, 0.000]
[0.500, 0.500, 0.000, 0.000]
[0.562, 0.250, 0.125, 0.0625]
[0.533, 0.268, 0.133, 0.0664]
[0.533, 0.267, 0.133, 0.0667]
```

```
sage: n3 = lambda x: x.n(digits = 3)
sage: show((M).apply_map(n3))
sage: show((M^2).apply_map(n3))
sage: show((M^5).apply_map(n3))
sage: show((M^10).apply_map(n3))
sage: show((M^30).apply_map(n3))
```

$$\begin{pmatrix} 0.500 & 0.500 & 0.000 & 0.000 \\ 0.500 & 0.000 & 0.500 & 0.000 \\ 0.500 & 0.000 & 0.000 & 0.500 \\ 1.00 & 0.000 & 0.000 & 0.000 \end{pmatrix}$$

$$\begin{pmatrix} 0.500 & 0.250 & 0.250 & 0.000 \\ 0.500 & 0.250 & 0.000 & 0.250 \\ 0.750 & 0.250 & 0.000 & 0.000 \\ 0.500 & 0.500 & 0.000 & 0.000 \end{pmatrix}$$

$$\begin{pmatrix} 0.531 & 0.281 & 0.125 & 0.0625 \\ 0.531 & 0.250 & 0.156 & 0.0625 \\ 0.531 & 0.250 & 0.125 & 0.0938 \\ 0.562 & 0.250 & 0.125 & 0.0625 \end{pmatrix}$$

$$\begin{pmatrix} 0.533 & 0.267 & 0.134 & 0.0664 \\ 0.533 & 0.267 & 0.133 & 0.0674 \\ 0.534 & 0.267 & 0.133 & 0.0664 \\ 0.533 & 0.268 & 0.133 & 0.0664 \end{pmatrix}$$

$$\begin{pmatrix} 0.533 & 0.267 & 0.133 & 0.0667 \\ 0.533 & 0.267 & 0.133 & 0.0667 \\ 0.533 & 0.267 & 0.133 & 0.0667 \\ 0.533 & 0.267 & 0.133 & 0.0667 \end{pmatrix}$$

La teoría de cadenas de Markov nos dice que se alcanza una distribución de probabilidad estable, que es única e independiente del punto de partida porque el grafo es fuertemente conexo. El vector con las probabilidades es un autovector por la izquierda de autovalor 1. Como el grafo es fuertemente conexo y tiene una arista que une un vértice consigo mismo, la cadena es irreducible y este autovector es único salvo escala. Escogemos el vector tal que la suma de sus componentes es uno.

```
sage: #print M.eigenvectors_left()[0][1]
sage: distribucion_estable = (M - 1).left_kernel().basis()[0]
sage: distribucion_estable /= sum(distribucion_estable)
sage: print g.vertices()
sage: print distribucion_estable
sage: print [n3(p) for p in distribucion_estable]
['***_', '**_*', '*_**', '_***']
(8/15, 4/15, 2/15, 1/15)
[0.533, 0.267, 0.133, 0.0667]
```

1.5.1 Palabras finales sobre las cadenas de Markov

Como vemos, la probabilidad de estar en cada nodo al cabo de un número grande de iteraciones no depende del estado inicial. Esta probabilidad se puede usar por tanto para medir la *importancia* de cada nodo. Por ejemplo, Google afirma que su algoritmo original para asignar una importancia a cada página web estaba basado en esta idea.

Referencia: la wikipedia en inglés menciona la interpretación del page rank como una cadena de Markov.

1.6 Circuitos cerrados

Aunque podemos movernos indefinidamente por el grafo sin repetirnos, antes o después pasaremos dos veces por el mismo nodo y en ese momento habremos completado un **circuito** .

1.6.1 Circuitos primos

Si un circuito pasa dos veces por el mismo sitio, es composición de **circuitos elementales o primos**, en los que no se pasa dos veces por el mismo vértice.

Los caminos en el grafo son composición de estos caminos elementales. En cierto modo, es equivalente a cómo el desarrollo decimal de un número puede no ser periódico (por ejemplo, el desarollo de π), pero todos los números se pueden desarrollar con los mismo dígitos.

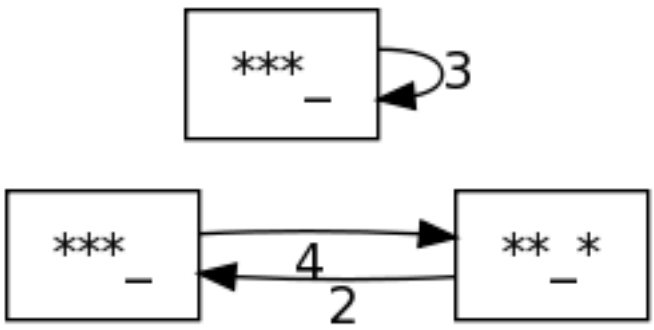
1.6.2 Algoritmo de Johnson

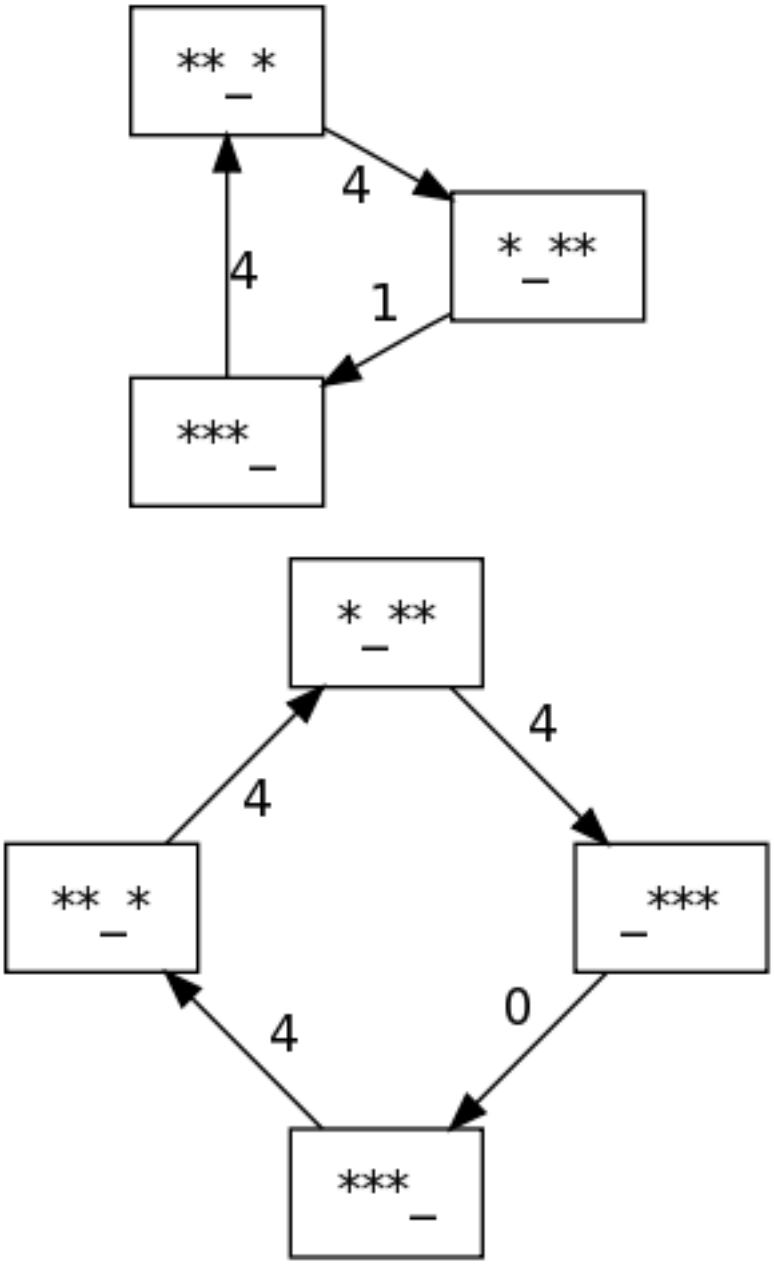
La teoría de circuitos cerrados en un grafo **no dirigido** es preciosa y tiene una descripción matemática muy elegante, que en última instancia se debe a se puede definir una suma de caminos. Sin embargo, para grafos dirigidos no se puede definir la suma de caminos y las matemáticas no dan una visión tan clara del conjunto de circuitos. Afortunadamente, existen algoritmos eficientes para encontrar todos los circuitos elementales en un grafo dirigido.

Nota: si listamos la altura a la que debemos lanzar cada pelota (la etiqueta de cada arista), recuperamos la notación **siteswap** .

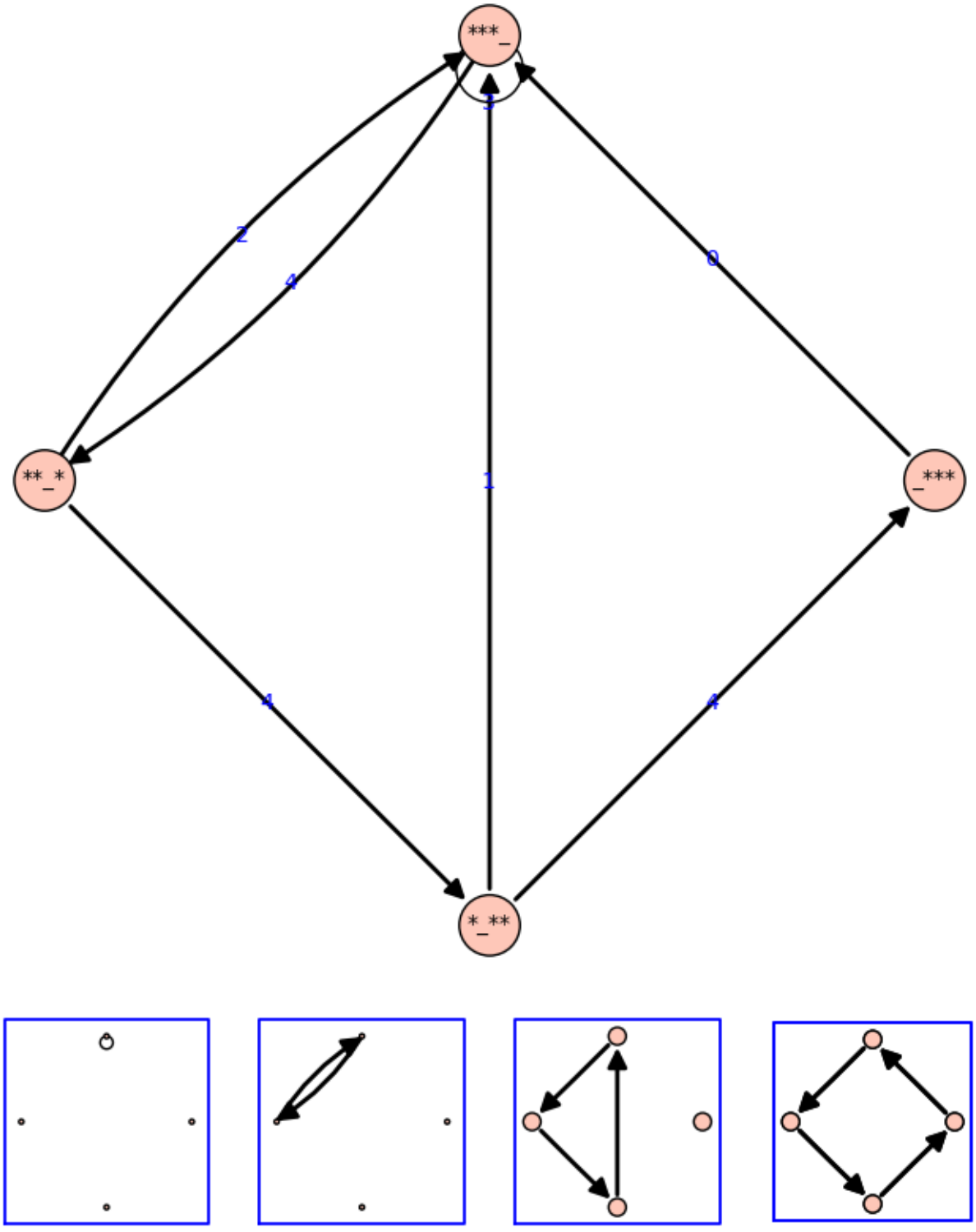
Todos los posibles trucos con 3 bolas y altura a lo sumo 4: listamos los circuitos del grafo malabar con 3 bolas y altura máxima 4, en notación siteswap.

```
sage: attach(DATA + 'circuits.py')
sage: g = grafo_malabar(3,4)
sage: for ls in circuits(g):
...       aristas = [(ls[j],ls[j+1])
...                  for j in range(len(ls)-1)]
...       sg = g.subgraph(edges = aristas)
...       print [g.edge_label(v1,v2) for v1,v2 in aristas]
...       graphviz(sg)
[3]
[4, 2]
[4, 4, 1]
[4, 4, 4, 0]
```






```
sage: g = grafo_malabar(3,4)
sage: g.show(edge_labels=True, layout='circular', vertex_size=300, figsize=8)
sage: graphs_list.show_graphs([g.subgraph(edges = [(ls[j],ls[j+1])
...                                                for j in range(len(ls)-1)])
...                            for ls in circuits(g)])
```



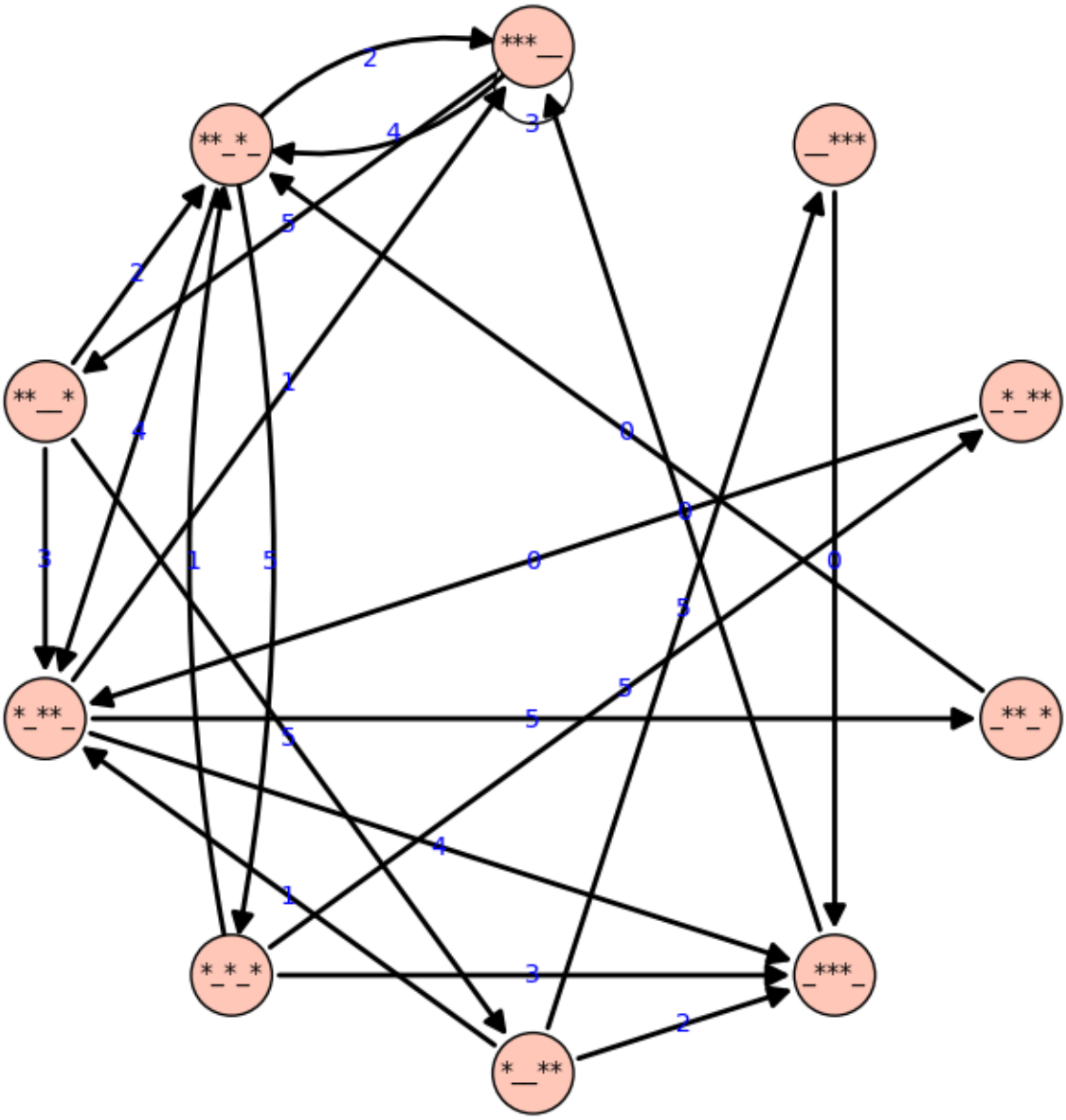
Todos los posibles trucos con 3 bolas y altura a lo sumo 5.

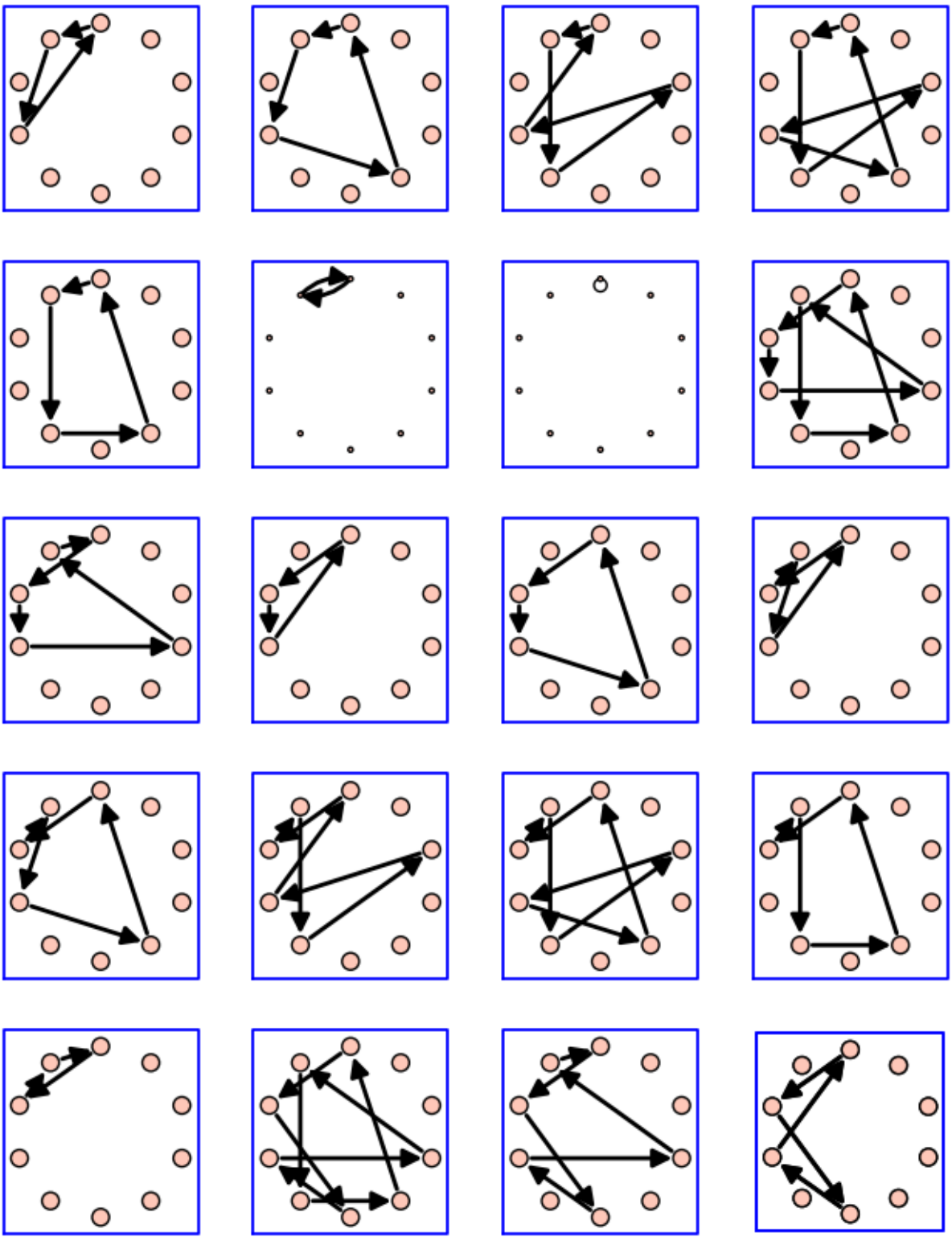
```
sage: attach(DATA + 'circuits.py')
sage: g = grafo_malabar(3,5)
sage: for ls in circuits(g):
```

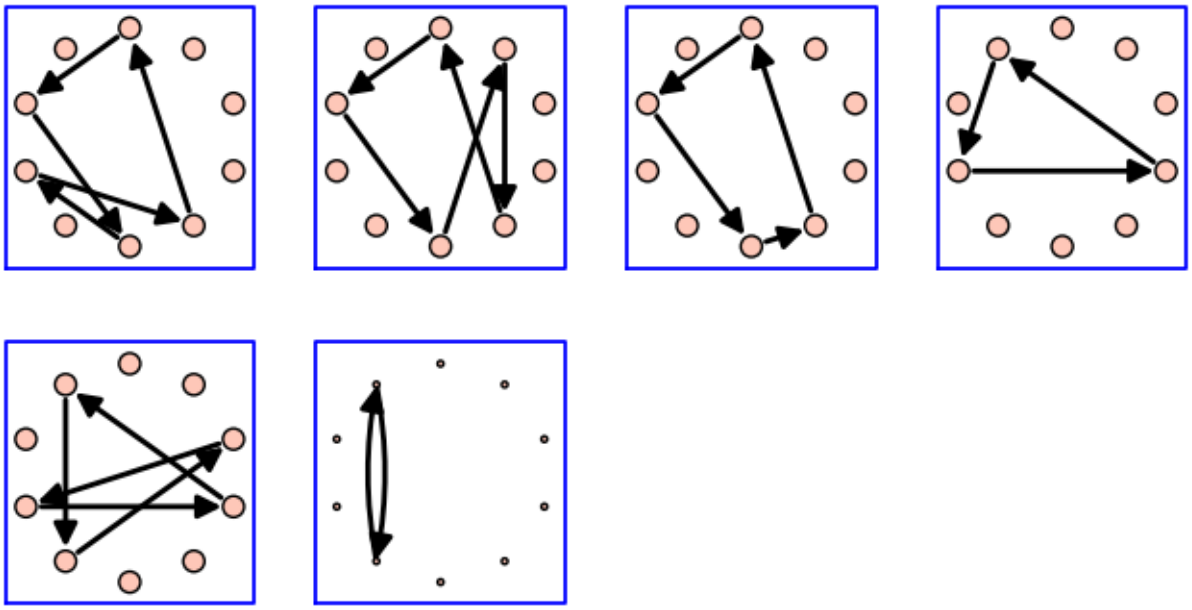
```
...       aristas = [(ls[j],ls[j+1])
...                  for j in range(len(ls)-1)]
...       print ''.join([str(g.edge_label(v1,v2)) for v1,v2 in aristas])
441
4440
45501
455040
4530
42
3
5350530
53502
531
5340
5241
52440
525501
5255040
52530
522
55150530
551502
5511
55140
55500
5520
450
55050
51
```

La misma información, de forma gráfica.

```
sage: g = grafo_malabar(3,5)
sage: show(g, edge_labels=True, layout='circular', figsize = 8, vertex_size=400)
sage: graphs_list.show_graphs([g.subgraph(edges = [(ls[j],ls[j+1])
...                                                for j in range(len(ls)-1)])
...                            for ls in circuits(g)])
```



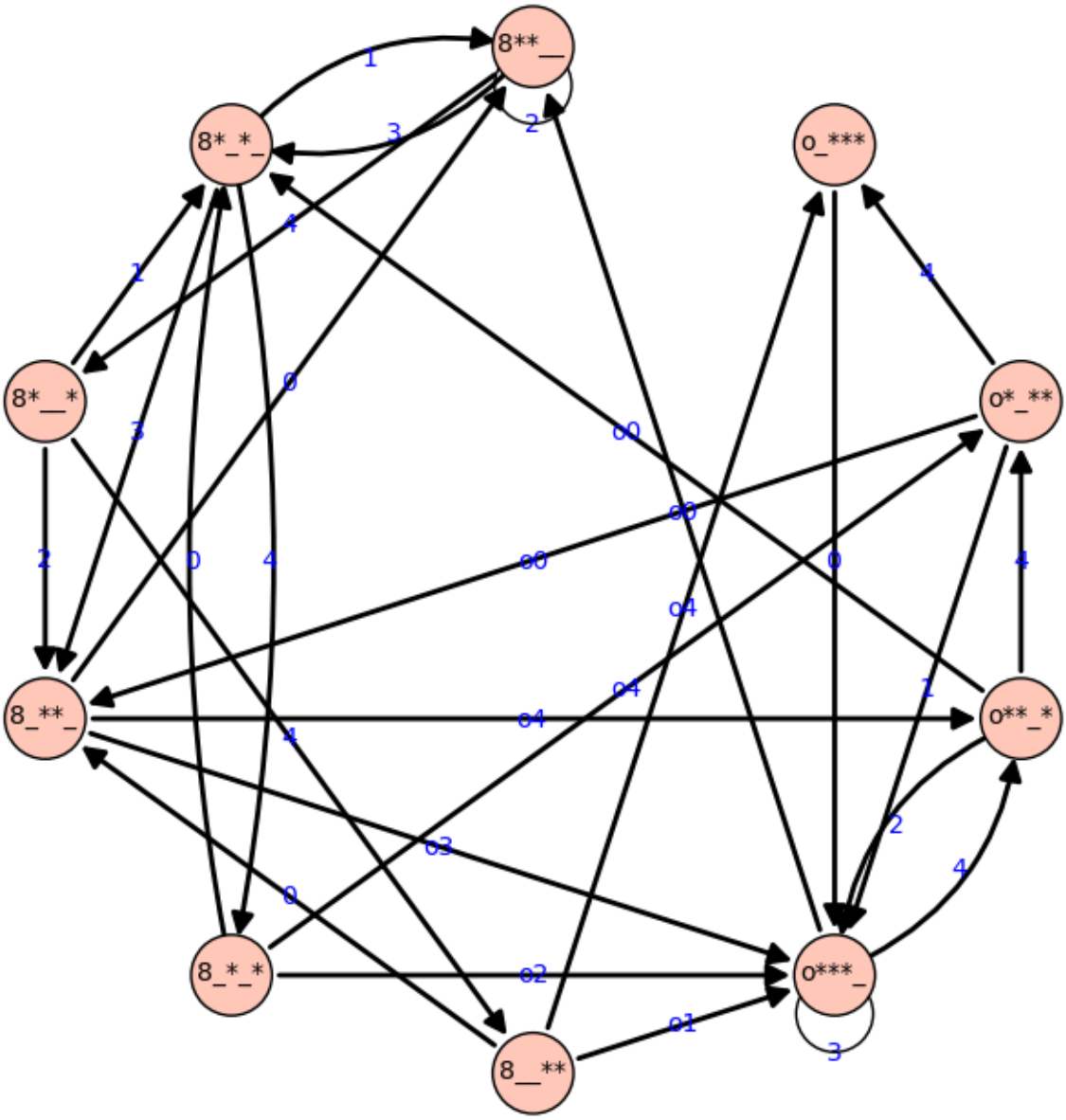


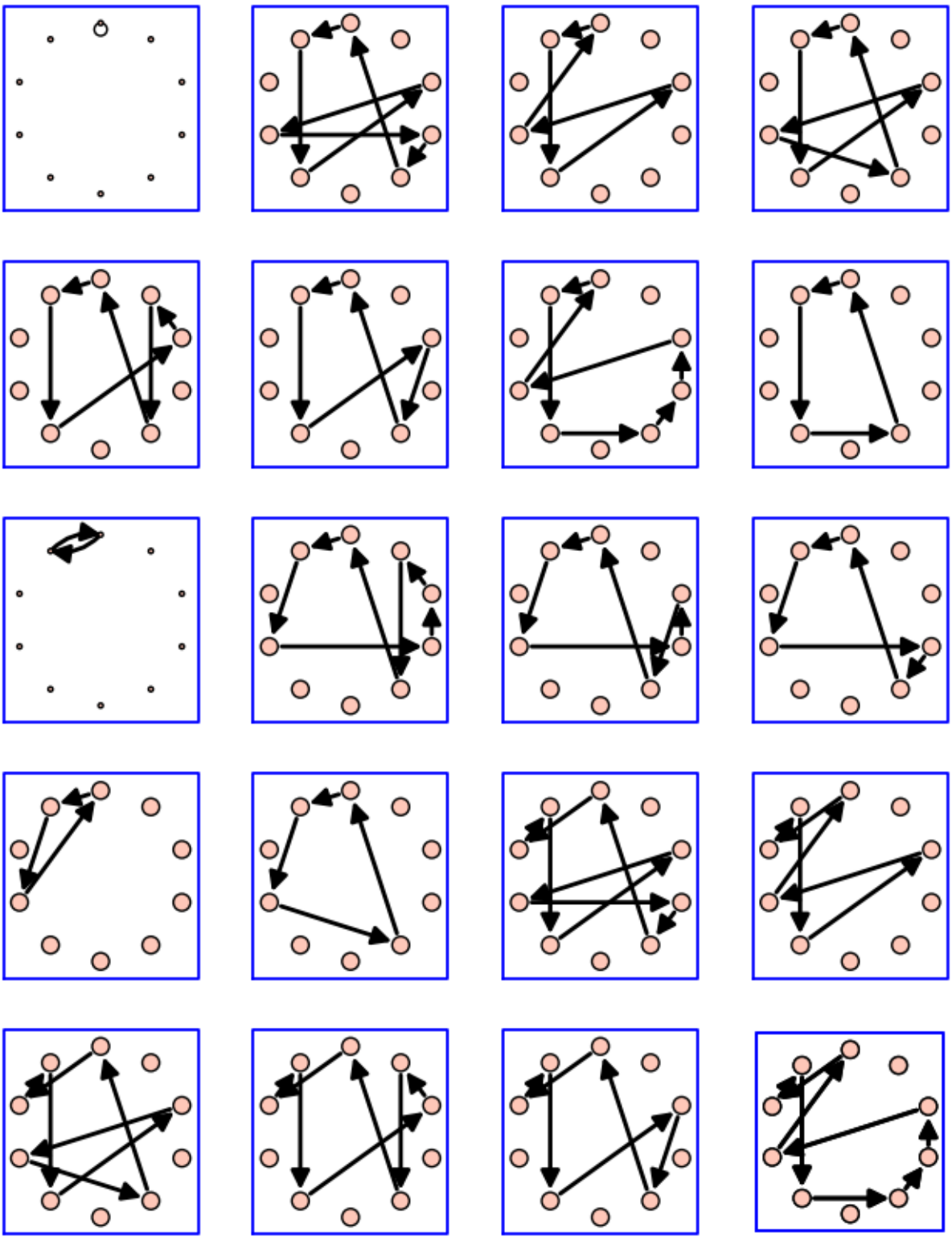


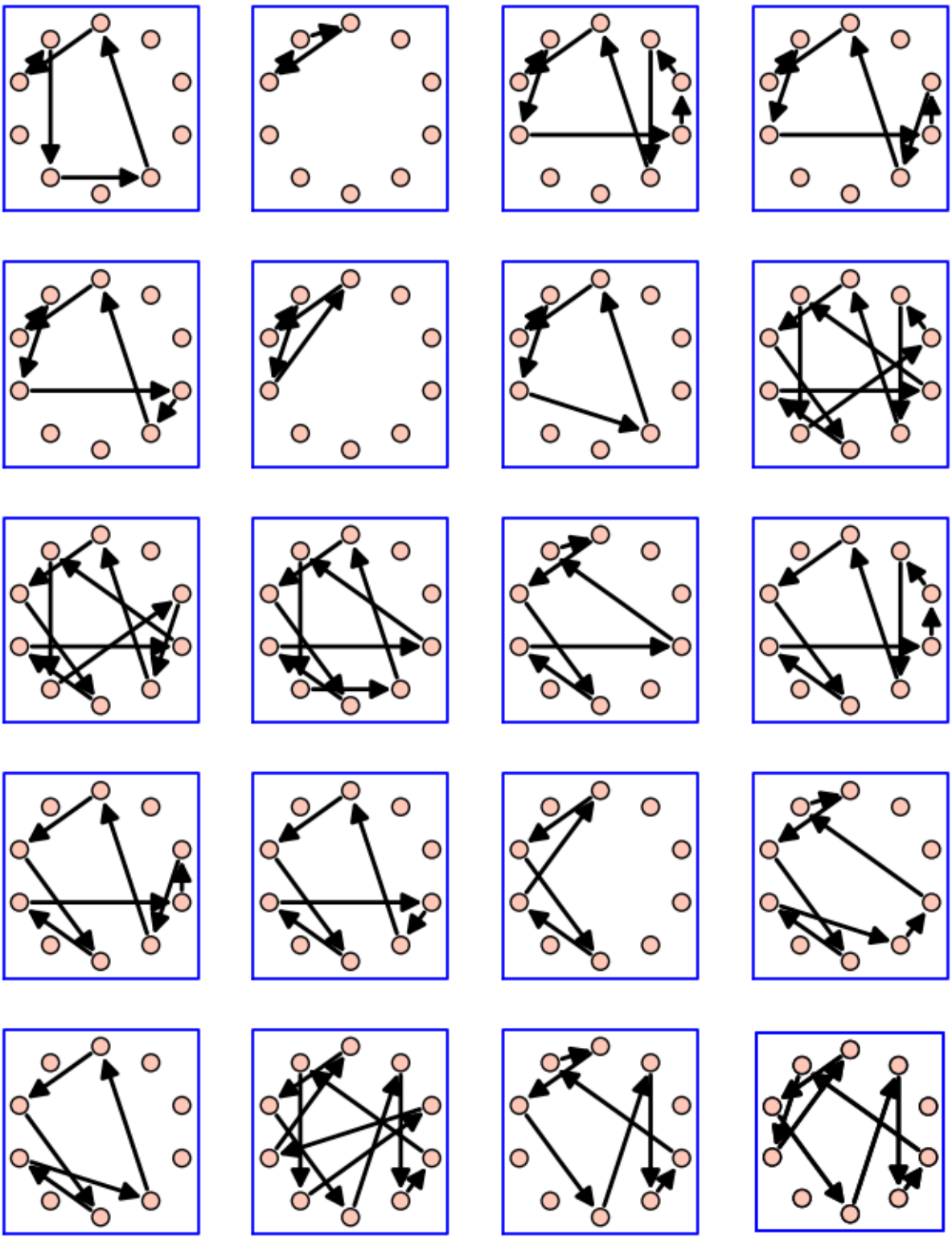
Finalmente, la misma información para el grafo de malabares con cabeza

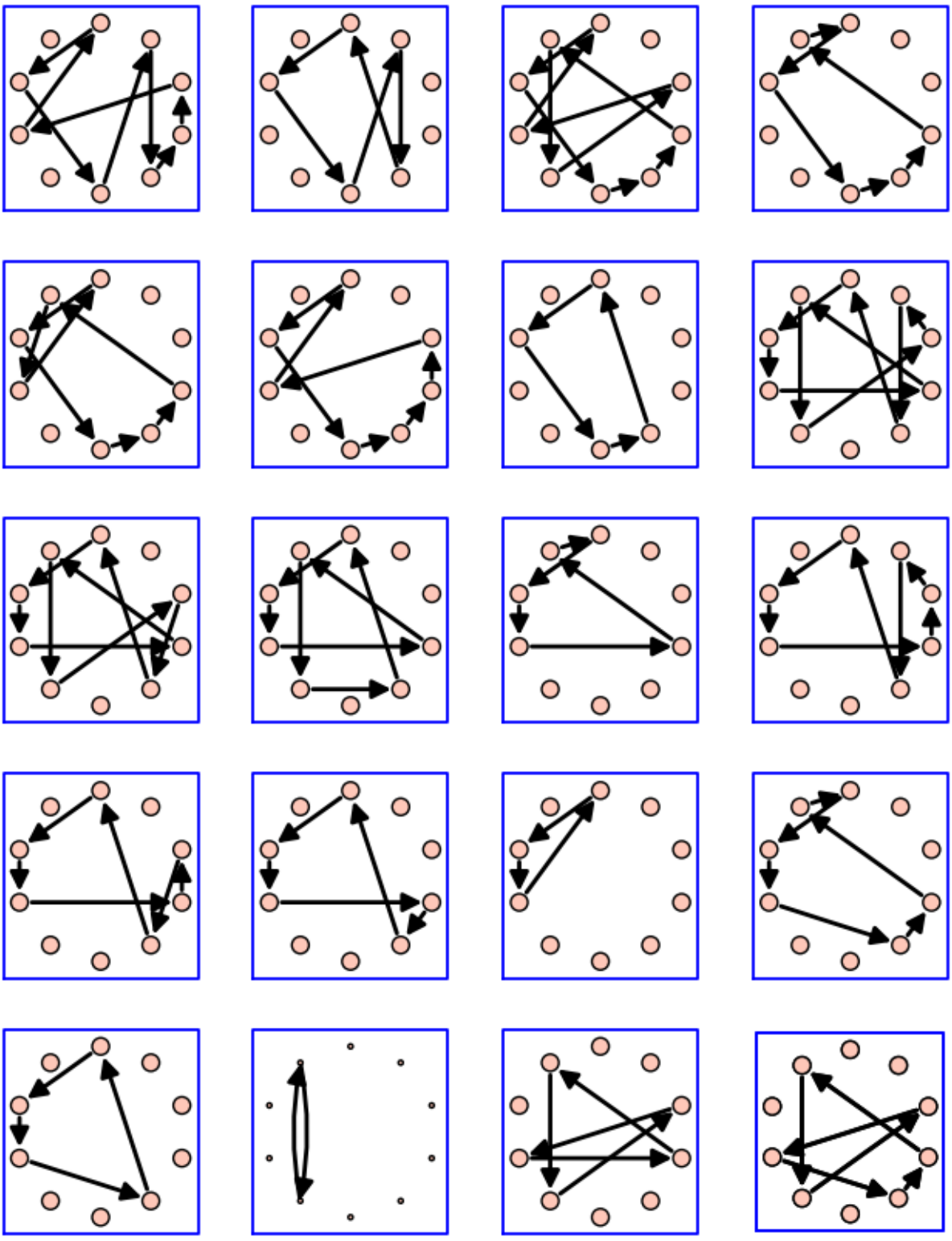
```
sage: g = grafo_malabar_con_cabeza(3,4)
sage: for ls in circuits(g):
...       aristas = [(ls[j],ls[j+1])
...                  for j in range(len(ls)-1)]
...       print ','.join([str(g.edge_label(v1,v2)) for v1,v2 in aristas])
...
sage: show(g, edge_labels=True, layout='circular',
...           figsize = 8, vertex_size=400)
sage: subgrafos = [g.subgraph(edges = [(ls[j],ls[j+1])
...                                    for j in range(len(ls)-1)])
...                for ls in circuits(g)]
sage: graphs_list.show_graphs(subgrafos)
2
3,4,o4,o0,o4,2,o0
3,4,o4,o0,0
3,4,o4,o0,o3,o0
3,4,o4,4,0,o0
3,4,o4,1,o0
3,4,o2,4,4,o0,0
3,4,o2,o0
3,1
3,3,o4,4,4,0,o0
3,3,o4,4,1,o0
3,3,o4,2,o0
3,3,0
3,3,o3,o0
4,1,4,o4,o0,o4,2,o0
4,1,4,o4,o0,0
4,1,4,o4,o0,o3,o0
4,1,4,o4,4,0,o0
4,1,4,o4,1,o0
4,1,4,o2,4,4,o0,0
4,1,4,o2,o0
```

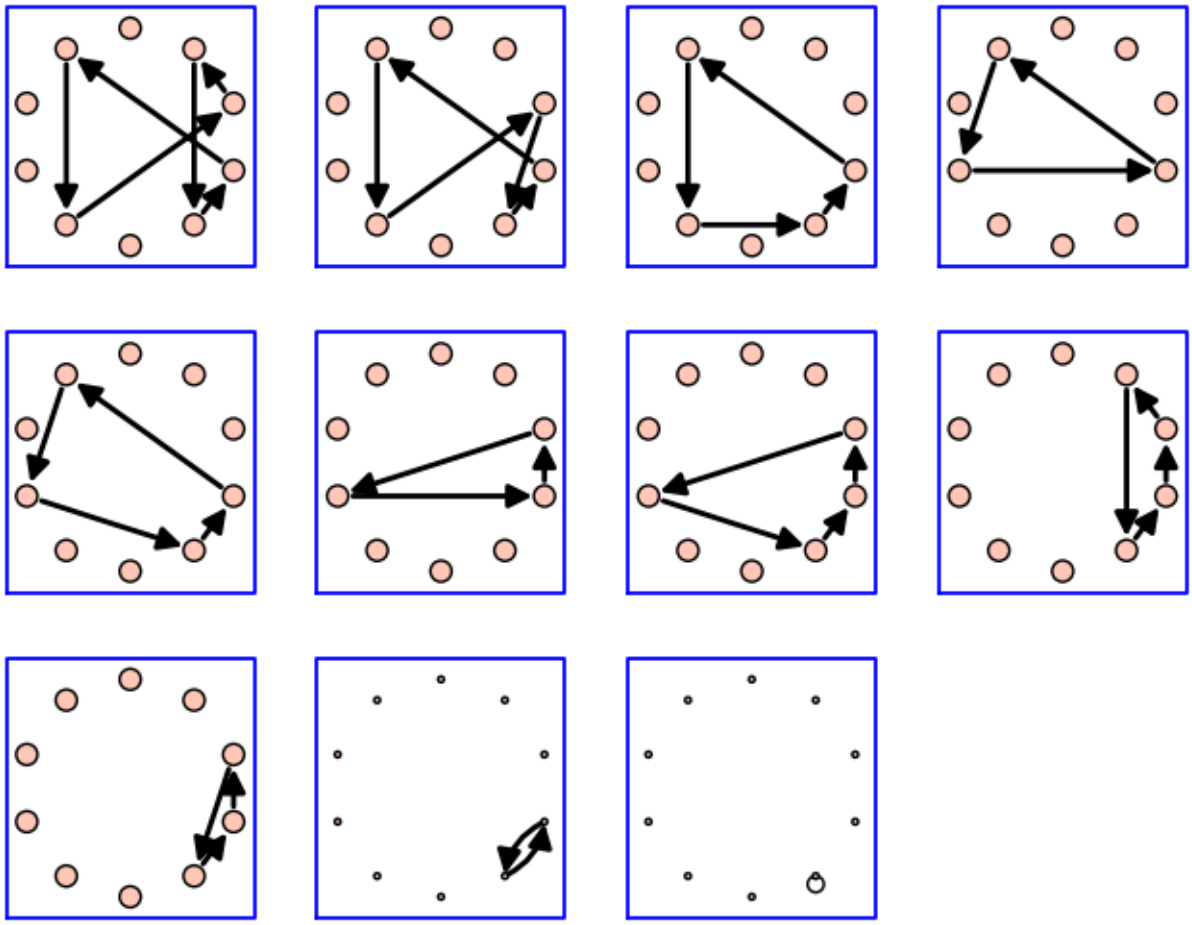
```
4,1,1
4,1,3,o4,4,4,0,o0
4,1,3,o4,4,1,o0
4,1,3,o4,2,o0
4,1,3,0
4,1,3,o3,o0
4,4,0,o4,o0,4,o4,4,0,o0
4,4,0,o4,o0,4,o4,1,o0
4,4,0,o4,o0,4,o2,o0
4,4,0,o4,o0,1
4,4,0,o4,4,4,0,o0
4,4,0,o4,4,1,o0
4,4,0,o4,2,o0
4,4,0,0
4,4,0,o3,4,o0,1
4,4,0,o3,o0
4,4,o4,0,4,o0,4,o4,o0,0
4,4,o4,0,4,o0,1
4,4,o4,0,4,o0,3,0
4,4,o4,0,4,4,o0,0
4,4,o4,0,o0
4,4,o1,4,o0,4,o4,o0,0
4,4,o1,4,o0,1
4,4,o1,4,o0,3,0
4,4,o1,4,4,o0,0
4,4,o1,o0
4,2,o4,o0,4,o4,4,0,o0
4,2,o4,o0,4,o4,1,o0
4,2,o4,o0,4,o2,o0
4,2,o4,o0,1
4,2,o4,4,4,0,o0
4,2,o4,4,1,o0
4,2,o4,2,o0
4,2,0
4,2,o3,4,o0,1
4,2,o3,o0
4,0
4,o4,o0,o4,o0
4,o4,o0,o3,4,o0
4,o4,4,0,4,o0
4,o4,1,4,o0
4,o2,4,o0
3,o4,o0
3,o3,4,o0
o4,4,o0
o3,4,4,o0
4,4,4,0
4,4,1
4,2
3
```


8**__
8*_*_
o_***
8*__*
o*_**
8_**_
o**_*
8_*_*
o***_
8__**









1.7 Ejercicios

- Escribe el grafo de malabares con dos pelotas de un color y una tercera de otro color y altura máxima 4. Si la tercera pelota es una manzana, identifica los estados en los que puedes pegarle un mordisco.
- Escribe el grafo de estados para dos malabaristas, con 5 pelotas y altura máxima 3. Los malabaristas pueden pasarle cada pelota al otro o lanzarla para recogerla ellos mismos.
- Pepito está aprendiendo a hacer malabares con cuchillos. Se ve capaz de lanzar y recibir los cuchillos a altura 5 pero no quiere lanzar dos cuchillos seguidos tan alto por si se hace un lío al recogerlos. Modifica el grafo de tres objetos y altura 5 para evitar lanzar dos objetos a altura 5 seguidos.
- Genera el grafo de dos malabaristas con altura máxima 3 y un total de 5 bolas trabajando en modo *canon* (los tiempos de uno y otro malabarista no están sincronzados, sino que tienen un retraso de medio tiempo). A modo de pista, tienes más abajo la construcción del grafo de n malabaristas, pero en modo síncrono.

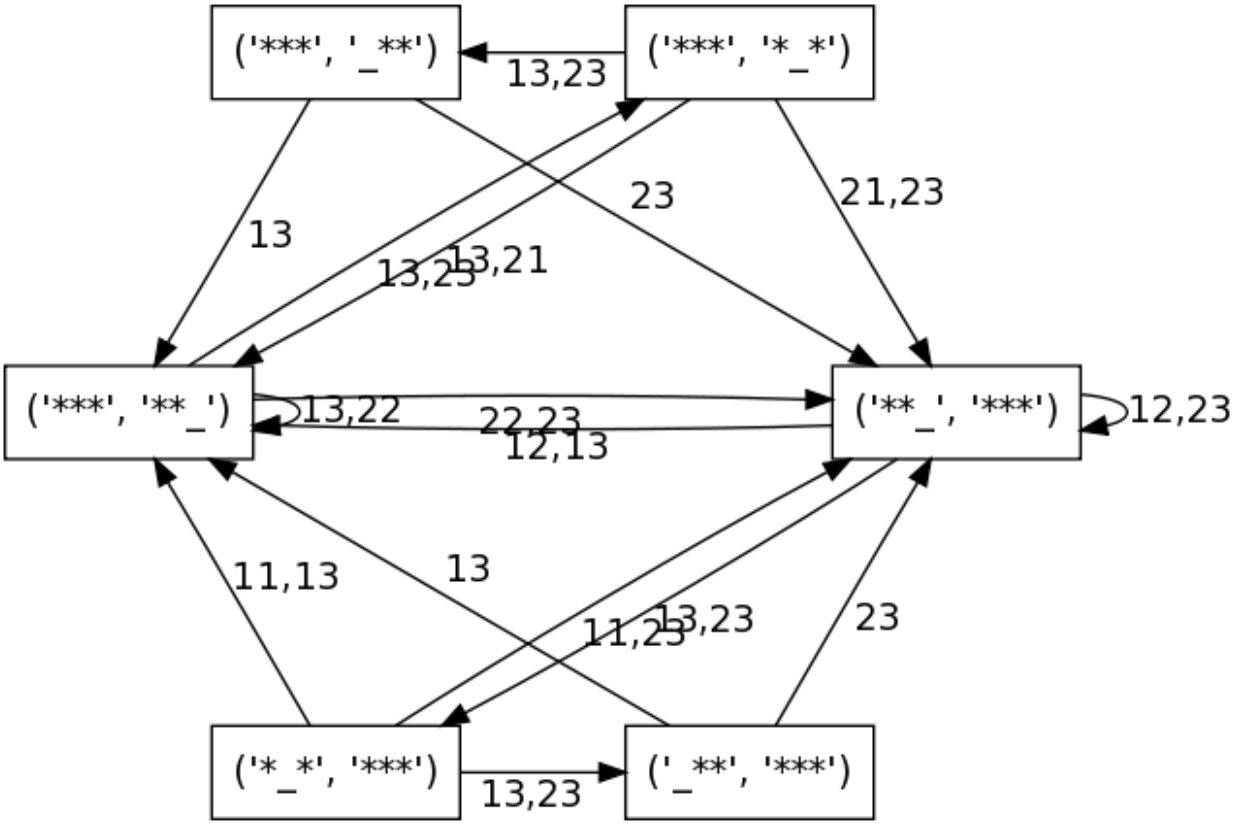
1.7.1 Cuestiones

- ¿Cuántos circuitos hamiltonianos tiene el grafo_malabar_con_cabeza(3,4)?
- Identifica en cada grafo el estado o estado "fundamentales", que debe verificar dos propiedades: que sea fácil alcanzar ese estado empezando con todas las bolas en la mano, y que se pueda mantener de forma indefinida.

1.8 Apéndice: varios malabaristas

Grafo de varios malabaristas que tienen todos la misma altura como cota y con una cantidad dada de bolas en total. Observa que si en un instante varios malabaristas deben lanzar una bola, sólo anotamos los destinos de esas bolas, sin distinguir quién hace qué lanzamiento. Es decir, que no distinguimos si cada malabarista se lanza a sí mismo o si cada uno le lanza al otro. Representamos cada estado mediante la unión de las listas de espera de todos los malabaristas, y cada arista mediante pares que dicen el malabarista que debe recibir y la altura del lanzamiento.

```
sage: def opcionesm(estado):
...       estado1 = tuple(malabarista[1:] + '_' for malabarista in estado)
...       bolas = sum(1 if malabarista[0]=='*' else 0
...                   for malabarista in estado)
...       if not bolas:
...           return {estado1:'0'}
...       huecos = [(i, j)
...                 for i,malabarista in enumerate(estado1)
...                 for j,s in enumerate(malabarista)
...                 if s=='_']
...       opciones = {}
...       for objetivos in Combinations(huecos, bolas):
...           nuevo_estado = list(estado1)
...           for i, j in objetivos:
...               cadena = nuevo_estado[i]
...               nuevo_estado[i] = cadena[:j] + '*'+ cadena[j+1:]
...           opciones[tuple(nuevo_estado)] = ','.join('%d%d'%(i+1,j+1)
...                                               for i, j in objetivos)
...       return opciones
sage: def grafo_malabarm(bolas, altura, malabaristas):
...       '''Crea el grafo de malabares para varios malabaristas
...
...       Acepta como argumento el numero de bolas y la lista de alturas
...       máximas de cada malabarista'''
...       total = altura*malabaristas
...       cadenas =  [ ''.join(('*' if j in ss else '_') for j in range(total))
...                    for ss in Subsets(range(total), bolas) ]
...       parte =lambda c: tuple(c[j:j+altura]
...                              for j in range(0, total, altura))
...       estadosm = [parte(c) for c in cadenas]
...       transiciones = dict((estado,opcionesm(estado)) for estado in estadosm)
...       return DiGraph(transiciones)
...

sage: #5 bolas, dos malabaristas a altura 3 cada uno
sage: g = grafo_malabarm(5, 3, 2)
sage: graphviz(g)
```

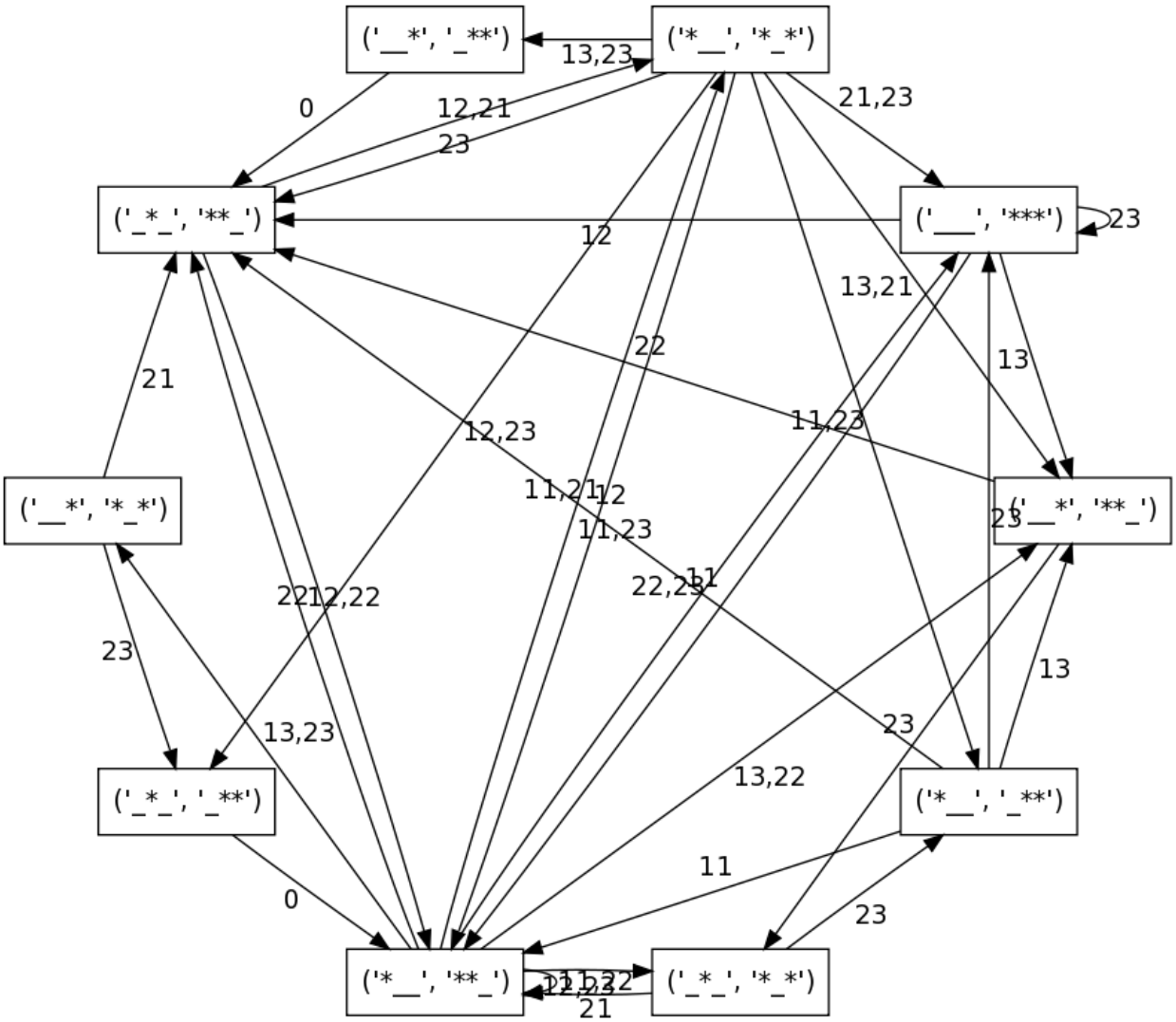


1.8.1 Coda: un malabarista y un inútil

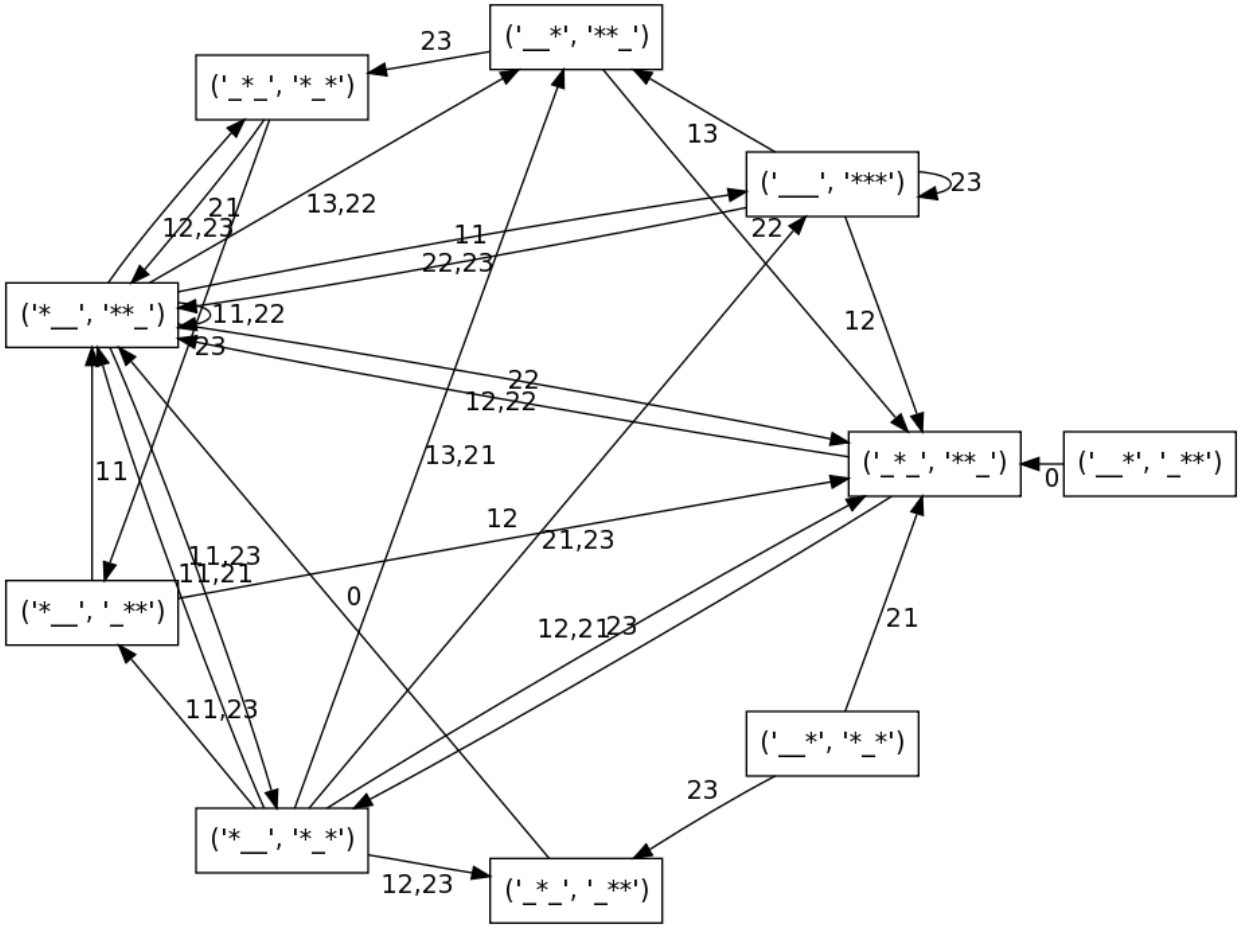
¿Podemos encontrar un truco en el que un inútil hace malabares con un malabarista?

- el inútil no puede tener más de una bola en su lista de caída
- el inútil no puede lanzar a altura mayor que 2

```
sage: #3 bolas, el malabarista trabaja a altura 3, el inutil a altura 2
sage: g = grafo_malabarm(3,3,2)
sage: #Añadimos la restricción de que el inútil
sage: #sólo puede tener una bola en la mano
sage: cota_bolas = 1
sage: vs = [v for v in g.vertices()
...           if sum(1 for c in v[0] if c=='*')<=cota_bolas ]
sage: sg = g.subgraph(vertices = vs)
sage: graphviz(sg)
```

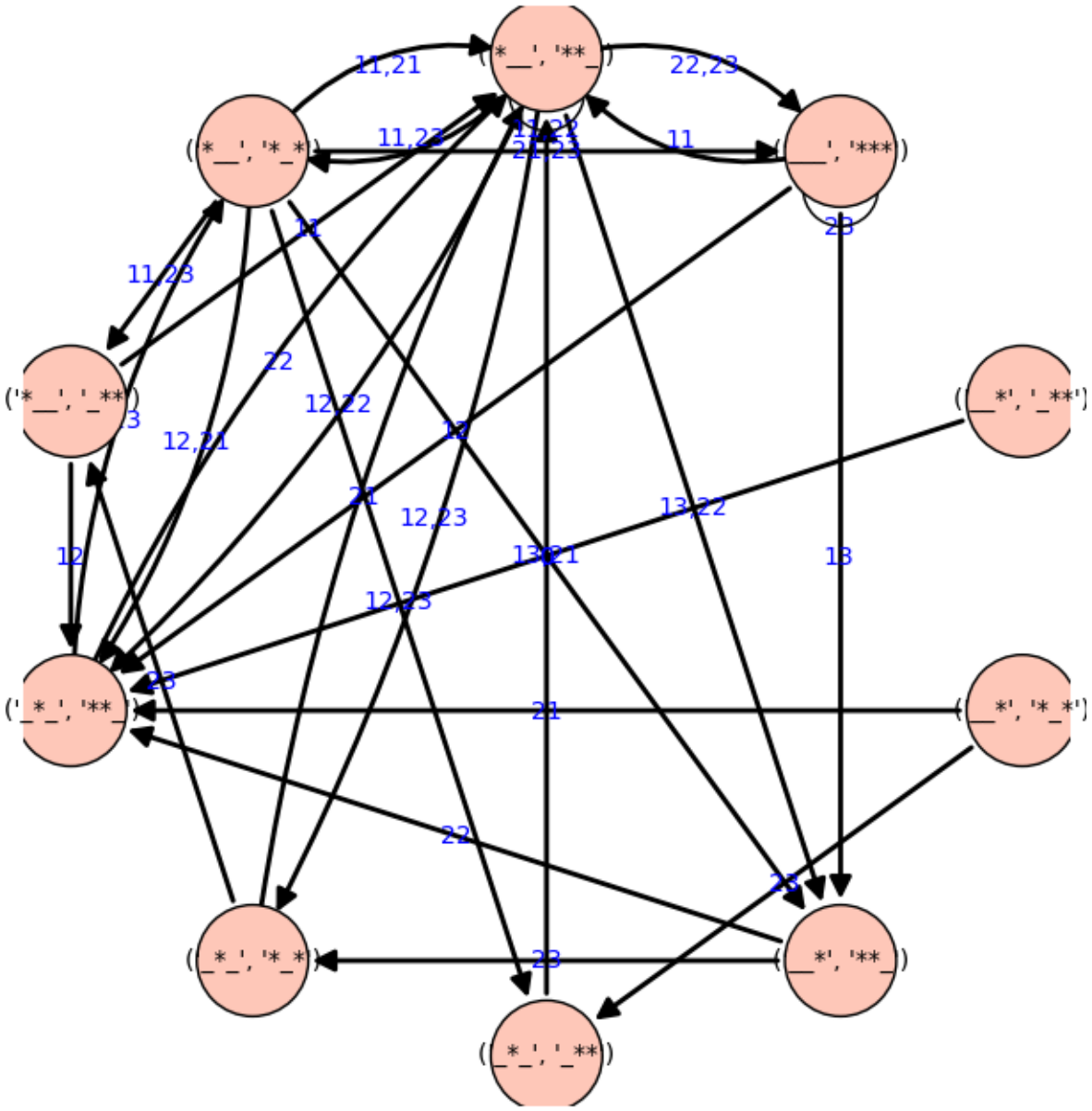


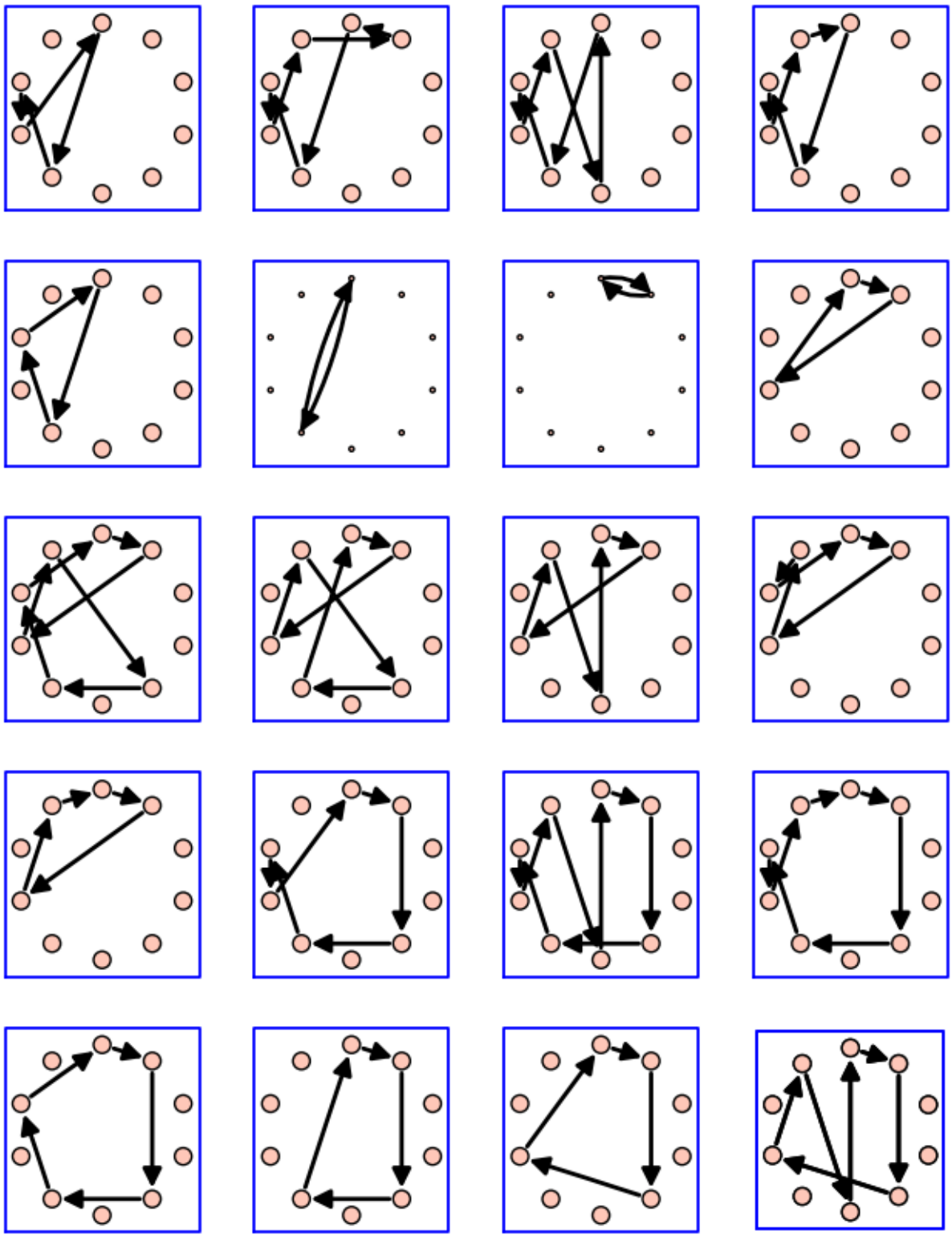

```
sage: #Añadimos la restricción de que el inútil
sage: #sólo puede lanzar a altura menor o igual que dos
sage: cota_altura = 2
sage: def criterio(arista):
...       '''Decide si una arista exige que el primer malabarista
...       tenga que lanzar mas alto que su cota
...       '''
...       e1,e2,lanzamientos = arista
...       m1, m2 = e1
...       if m1[0] == '_':
...           return True
...       if m2[0] == '_':
...           return int(lanzamientos[1]) <= cota_altura
...       return (int(lanzamientos[1]) <= cota_altura or
...               int(lanzamientos[4]) <= cota_altura)
sage: aristas = [arista for arista in g.edges()
...                     if criterio(arista) ]
sage: sg2 = sg.subgraph(edges = aristas)
sage: graphviz(sg2)
```

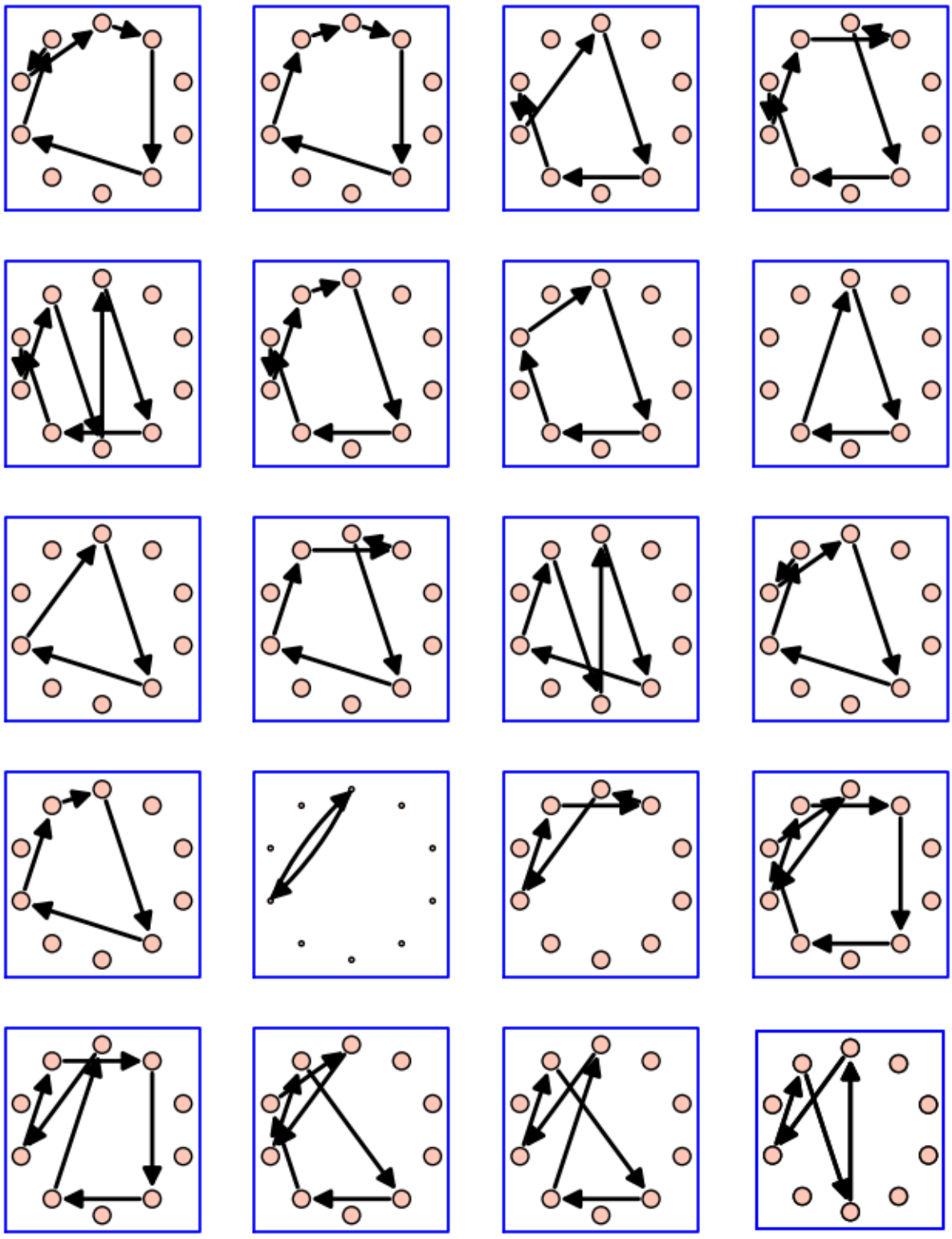


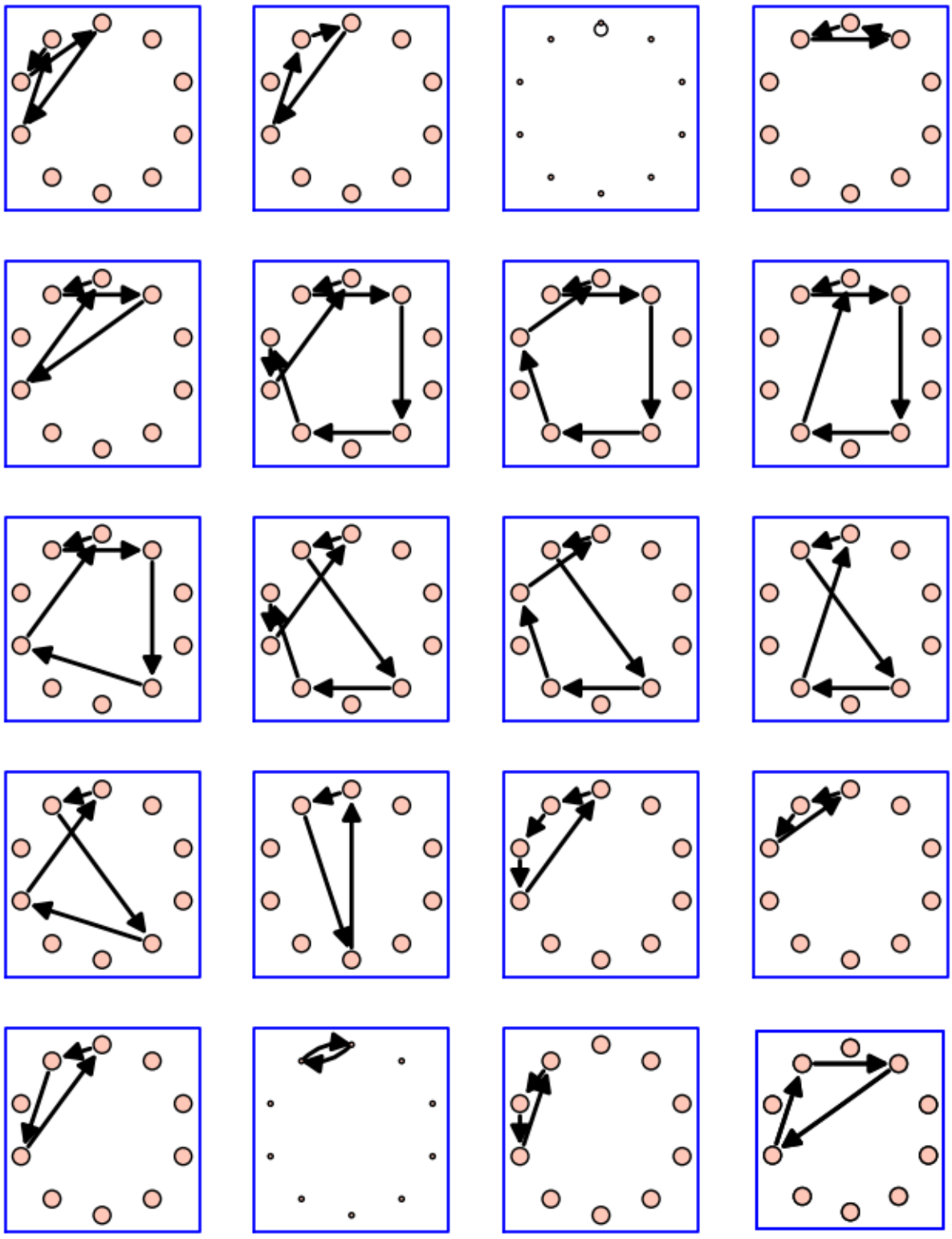

```
sage: #Todos los circuitos
sage: g = sg2
sage: for ls in circuits(g):
...       aristas = [(ls[j],ls[j+1])
...                  for j in range(len(ls)-1)]
...       print ';'.join([str(g.edge_label(v1,v2)) for v1,v2 in aristas])
...
sage: show(g, edge_labels=True, layout='circular',
...           figsize = 8, vertex_size=800)
sage: subgrafos = [g.subgraph(edges = [(ls[j],ls[j+1])
...                                    for j in range(len(ls)-1)])
...                for ls in circuits(g)]
sage: graphs_list.show_graphs(subgrafos)
12,23;23;12;22
12,23;23;12;23;21,23;11
12,23;23;12;23;12,23;0
12,23;23;12;23;11,21
12,23;23;11
12,23;21
22,23;11
22,23;12;22
22,23;12;23;13,21;23;23;11
22,23;12;23;13,21;23;21
22,23;12;23;12,23;0
22,23;12;23;11,23;11
```

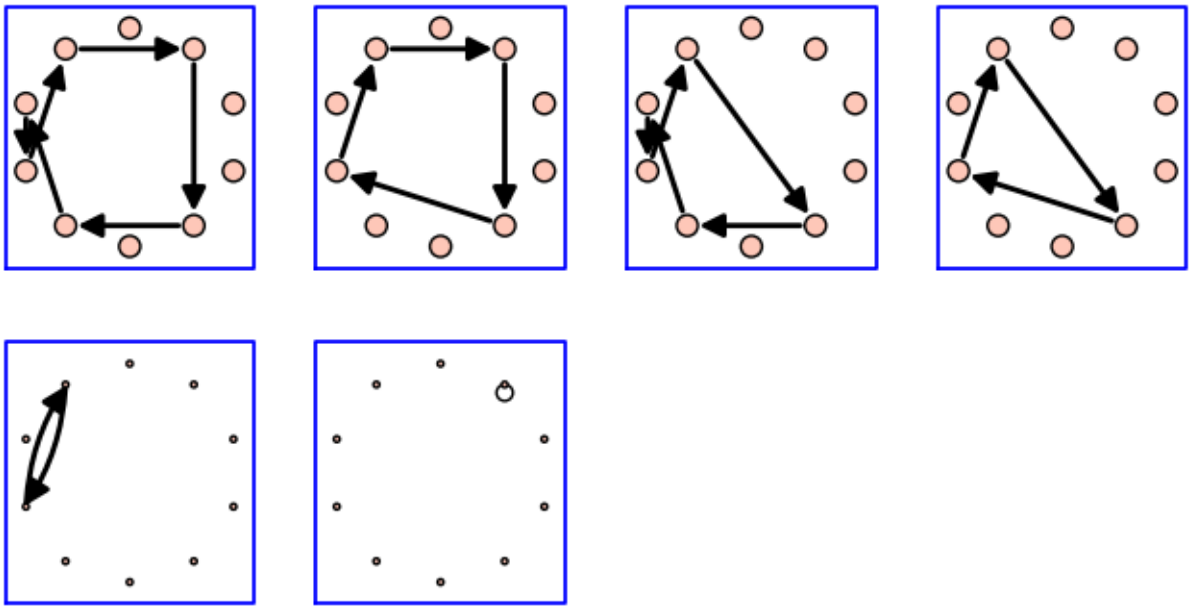
```
22,23;12;23;11,21
22,23;13;23;23;12;22
22,23;13;23;23;12;23;12,23;0
22,23;13;23;23;12;23;11,21
22,23;13;23;23;11
22,23;13;23;21
22,23;13;22;22
22,23;13;22;23;12,23;0
22,23;13;22;23;11,23;11
22,23;13;22;23;11,21
13,22;23;23;12;22
13,22;23;23;12;23;21,23;11
13,22;23;23;12;23;12,23;0
13,22;23;23;12;23;11,21
13,22;23;23;11
13,22;23;21
13,22;22;22
13,22;22;23;21,23;11
13,22;22;23;12,23;0
13,22;22;23;11,23;11
13,22;22;23;11,21
12,22;22
12,22;23;21,23;11
12,22;23;21,23;13;23;23;11
12,22;23;21,23;13;23;21
12,22;23;13,21;23;23;11
12,22;23;13,21;23;21
12,22;23;12,23;0
12,22;23;11,23;11
12,22;23;11,21
11,22
11,23;21,23;11
11,23;21,23;12;22
11,23;21,23;13;23;23;12;22
11,23;21,23;13;23;23;11
11,23;21,23;13;23;21
11,23;21,23;13;22;22
11,23;13,21;23;23;12;22
11,23;13,21;23;23;11
11,23;13,21;23;21
11,23;13,21;22;22
11,23;12,23;0
11,23;11,23;12;22
11,23;11,23;11
11,23;12,21;22
11,23;11,21
11,23;12;23
21,23;12;23
21,23;13;23;23;12;23
21,23;13;22;23
13,21;23;23;12;23
13,21;22;23
12,21;23
23
```


('*__', '**_')
('*__', '*_*')
('___', '***')
('*__', '_**')
('___*', '_**')
('_*_', '**_')
('___*', '*_*')
('_*_', '*_*')
('___*', '**_')
('_*_', '_**')
11,21
22,23
11,23
11
11,23
22
12,22
12,21
12,23
13,22
13,21
12,23
23
21
22
23
23

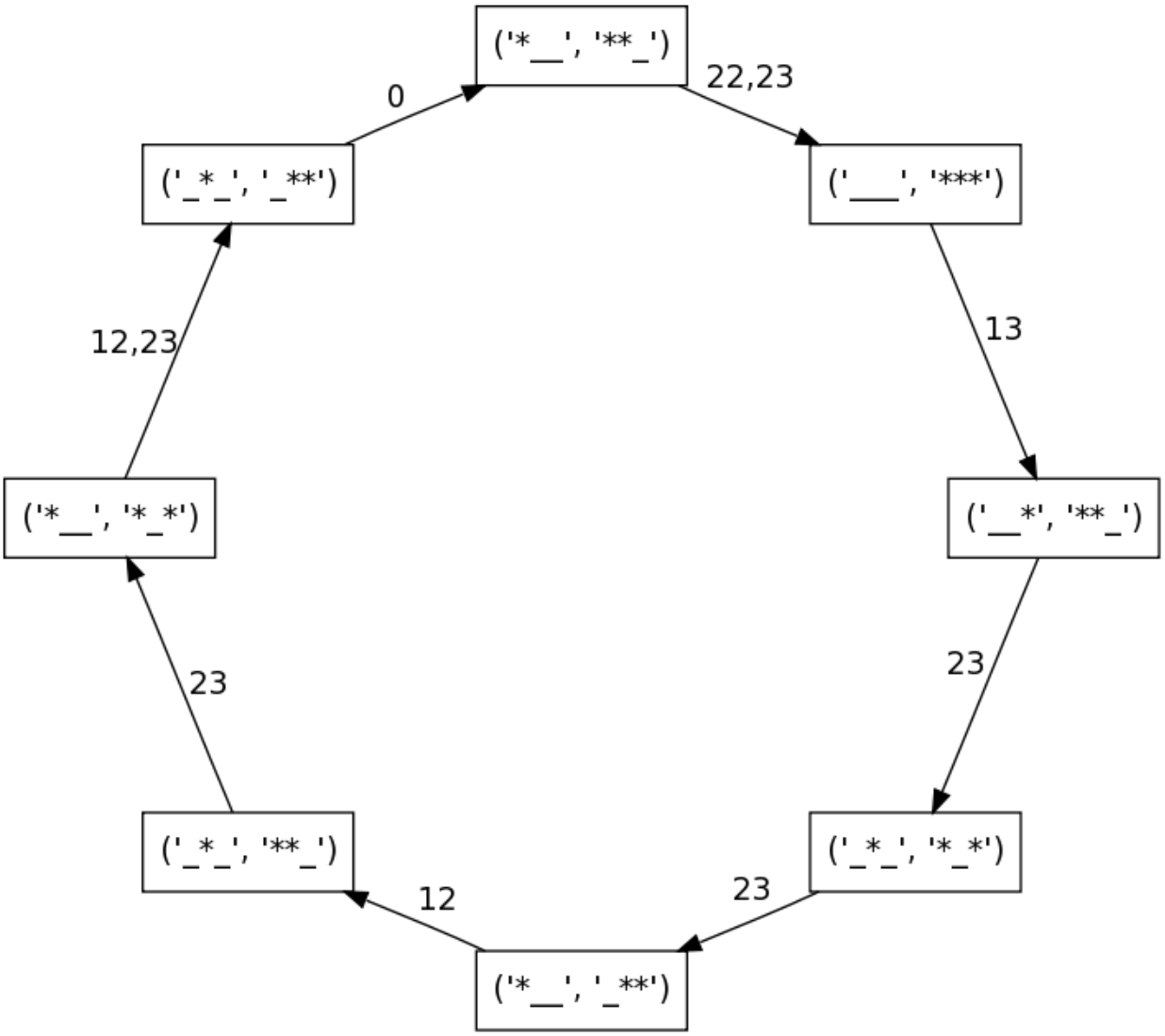








```
sage: #El circuito mas largo
sage: g = sg2
sage: L, circuito = max((len(ls), ls) for ls in circuits(g))
sage: subgrafo = g.subgraph(edges = [(circuito[j],circuito[j+1])
...                                  for j in range(len(circuito)-1)])
sage: graphviz(subgrafo)
```



1.9 Créditos

Esta presentación usa el programa Sage para hacer los cálculos sobre grafos y la librería graphviz para dibujar los grafos.

Muchas gracias a Daniel Sánchez , y a Nicolas M. Thiéry y Florent Hivert por sus comentarios (y a Clara, por supuesto).

1.10 Licencia

Este documento se distribuye con una licencia GFDL , ó cc-by-sa, a tu elección.