2014-05-21.sagews

May 21, 2014

Contents

1 Math 480b Sage Course

1.1 Today: Graph Theory

1.1.1 May 21, 2014

Screencast: `http://youtu.be/4a1QZJ8aKUM`
Plan

- Questions
- Homework:
  - all grading of hw6 should have been returned
- Presentations: update about final schedule
- Topic: Graph Theory

1.2 Graph Theory

- Vertices and edges: Networks
- Incredibly important in computer science, social sciences, etc.
- Also big in pure math research part of combinatorics. Very popular REU (=research experience for undergraduates) topic.

1.3 A quick basic demo

(so you can easily do that homework problem)

```
g = graphs.BarbellGraph(4, 4)
g
```

```
Barbell graph: Graph on 12 vertices
```

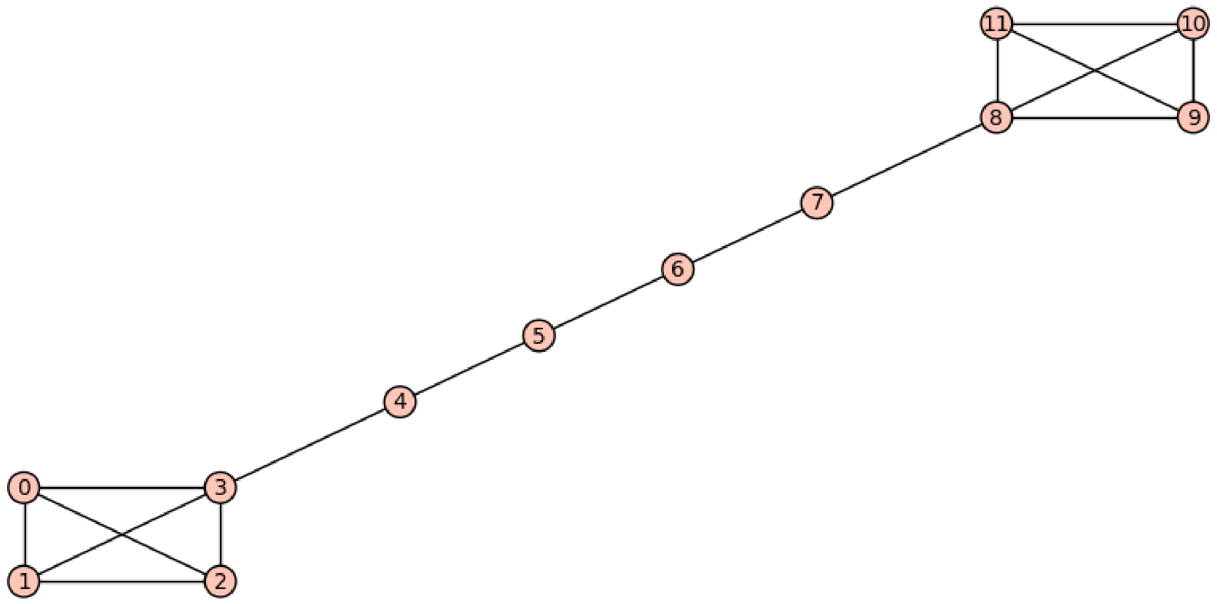
```
type(g)
```

```
<class 'sage.graphs.graph.Graph'>
```

```
(0,9)(1,10)(2,11)(3,8)(4,7)(5,6)
```

```
(0,9)(1,10)(2,11)(3,8)(4,7)(5,6)
```

```
plot(g) #+ plot(sin, (-5,5))
```



```
v = v = g.to_dictionary(); v
```

```
{0: [1, 2, 3], 1: [0, 2, 3], 2: [0, 1, 3], 3: [0, 1, 2, 4], 4: [3, 5], 5: [4, 6], 6: [5,
7], 7: [6, 8], 8: [7, 9, 10, 11], 9: [8, 10, 11], 10: [8, 9, 11], 11: [8, 9, 10]}
```

```
type(v)
```

```
<type 'dict'>
```

```
Graph(v)
```

```
Graph on 12 vertices
```

```
type(v)
```

```
<type 'dict'>
```

```
Graph(v)
```

```
Graph on 12 vertices
```

```
# graphs have a few methods...
len(dir(g))
```

```
377
```

A clique is a maximally completed subgraph. Heres a function to find them all

```
g.cliques_maximal()
```

```
[[0, 1, 2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9, 10, 11]]
```

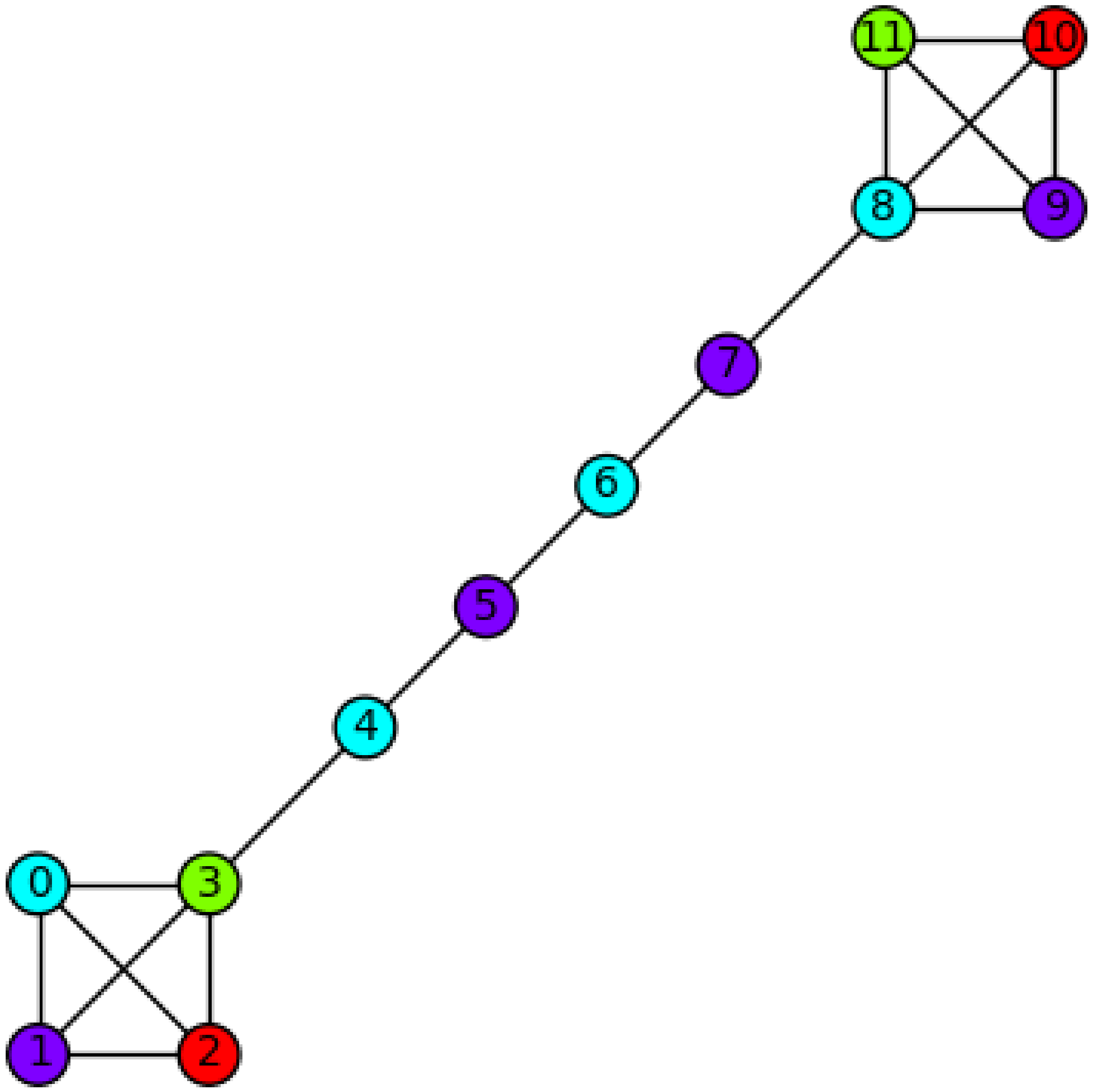
```
(0,9)(1,10)(2,11)(3,8)(4,7)(5,6)
```

```
P = g.coloring(); P
```

```
[[2, 10], [3, 11], [0, 4, 6, 8], [1, 5, 7, 9]]
```

```
(0,9)(1,10)(2,11)(3,8)(4,7)(5,6)
```

```
g.plot(partition=P)
```



```
g.adjacency_matrix()
[0 1 1 1 0 0 0 0 0 0 0 0]
[1 0 1 1 0 0 0 0 0 0 0 0]
[1 1 0 1 0 0 0 0 0 0 0 0]
[1 1 1 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 1 1 1]
[0 0 0 0 0 0 0 0 1 0 1 1]
```

```
[0 0 0 0 0 0 0 0 1 1 0 1]
[0 0 0 0 0 0 0 0 1 1 1 0]
```

```
0   1    2    3    4    5    6
3   2    5    1    4    6    0

3   2    5    1    4    6    0

# disjoint cycle notation
(0,3,1,2,5,6)
```

```
G = g.automorphism_group(); G
```

```
Permutation Group with generators [(10,11), (9,10), (1,2), (0,1),
(0,9)(1,10)(2,11)(3,8)(4,7)(5,6)]
```

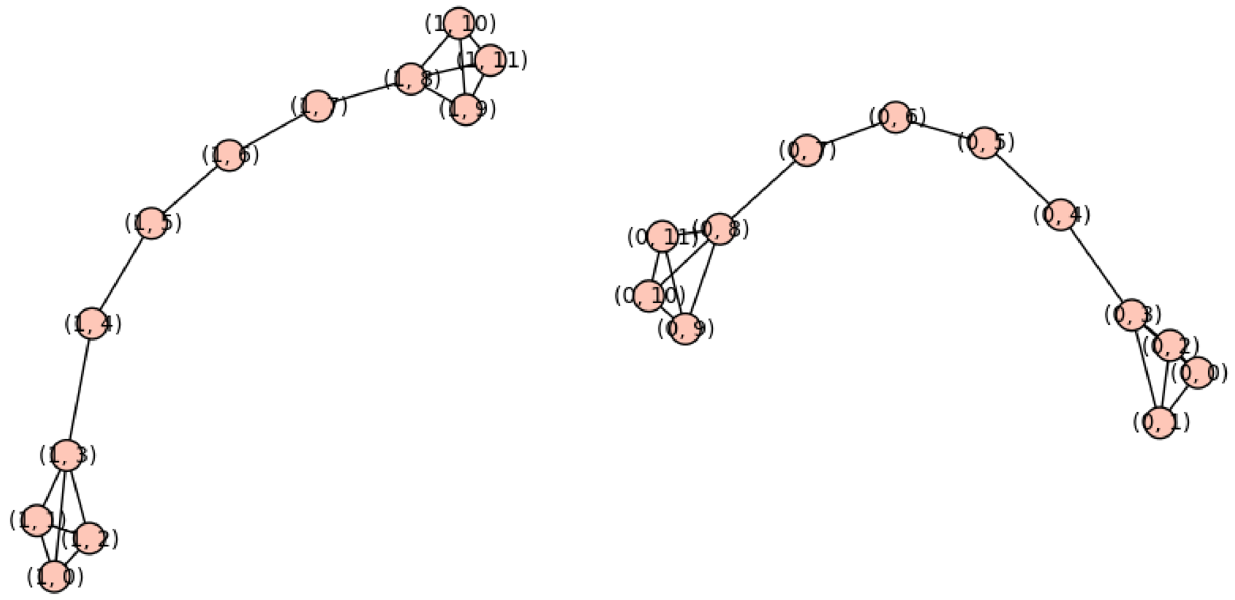
```
G.cardinality()
```

```
72
```

```
list(G)
```

```
[(), (10,11), (9,10), (9,10,11), (9,11,10), (9,11), (1,2), (1,2)(10,11), (1,2)(9,10),
(1,2)(9,10,11), (1,2)(9,11,10), (1,2)(9,11), (0,1), (0,1)(10,11), (0,1)(9,10),
(0,1)(9,10,11), (0,1)(9,11,10), (0,1)(9,11), (0,1,2), (0,1,2)(10,11), (0,1,2)(9,10),
(0,1,2)(9,10,11), (0,1,2)(9,11,10), (0,1,2)(9,11), (0,2,1), (0,2,1)(10,11), (0,2,1)(9,10),
(0,2,1)(9,10,11), (0,2,1)(9,11,10), (0,2,1)(9,11), (0,2), (0,2)(10,11), (0,2)(9,10),
(0,2)(9,10,11), (0,2)(9,11,10), (0,2)(9,11), (0,9)(1,10)(2,11)(3,8)(4,7)(5,6),
(0,9)(1,10,2,11)(3,8)(4,7)(5,6), (0,9,1,10)(2,11)(3,8)(4,7)(5,6),
(0,9,1,10,2,11)(3,8)(4,7)(5,6), (0,9,2,11,1,10)(3,8)(4,7)(5,6),
(0,9,2,11)(1,10)(3,8)(4,7)(5,6), (0,9)(1,11,2,10)(3,8)(4,7)(5,6),
(0,9)(1,11)(2,10)(3,8)(4,7)(5,6), (0,9,1,11,2,10)(3,8)(4,7)(5,6),
(0,9,1,11)(2,10)(3,8)(4,7)(5,6), (0,9,2,10)(1,11)(3,8)(4,7)(5,6),
(0,9,2,10,1,11)(3,8)(4,7)(5,6), (0,10,1,9)(2,11)(3,8)(4,7)(5,6),
(0,10,2,11,1,9)(3,8)(4,7)(5,6), (0,10)(1,9)(2,11)(3,8)(4,7)(5,6),
(0,10,2,11)(1,9)(3,8)(4,7)(5,6), (0,10)(1,9,2,11)(3,8)(4,7)(5,6),
(0,10,1,9,2,11)(3,8)(4,7)(5,6), (0,10,1,11,2,9)(3,8)(4,7)(5,6),
(0,10,2,9)(1,11)(3,8)(4,7)(5,6), (0,10)(1,11,2,9)(3,8)(4,7)(5,6),
(0,10,2,9,1,11)(3,8)(4,7)(5,6), (0,10)(1,11)(2,9)(3,8)(4,7)(5,6),
(0,10,1,11)(2,9)(3,8)(4,7)(5,6), (0,11,2,10,1,9)(3,8)(4,7)(5,6),
(0,11,1,9)(2,10)(3,8)(4,7)(5,6), (0,11,2,10)(1,9)(3,8)(4,7)(5,6),
(0,11)(1,9)(2,10)(3,8)(4,7)(5,6), (0,11,1,9,2,10)(3,8)(4,7)(5,6),
(0,11)(1,9,2,10)(3,8)(4,7)(5,6), (0,11,2,9)(1,10)(3,8)(4,7)(5,6),
(0,11,1,10,2,9)(3,8)(4,7)(5,6), (0,11,2,9,1,10)(3,8)(4,7)(5,6),
(0,11)(1,10,2,9)(3,8)(4,7)(5,6), (0,11,1,10)(2,9)(3,8)(4,7)(5,6),
(0,11)(1,10)(2,9)(3,8)(4,7)(5,6)]
```

```
plot(g.disjoint_union(g))
```



```
g.plot3d()
```

1.4 Historical Interlude

- Robert Miller and Emily Kirkman
- Nathann Cohen

Started with the graph survey": `http://wiki.sagemath.org/graph_survey`
Conclusion:

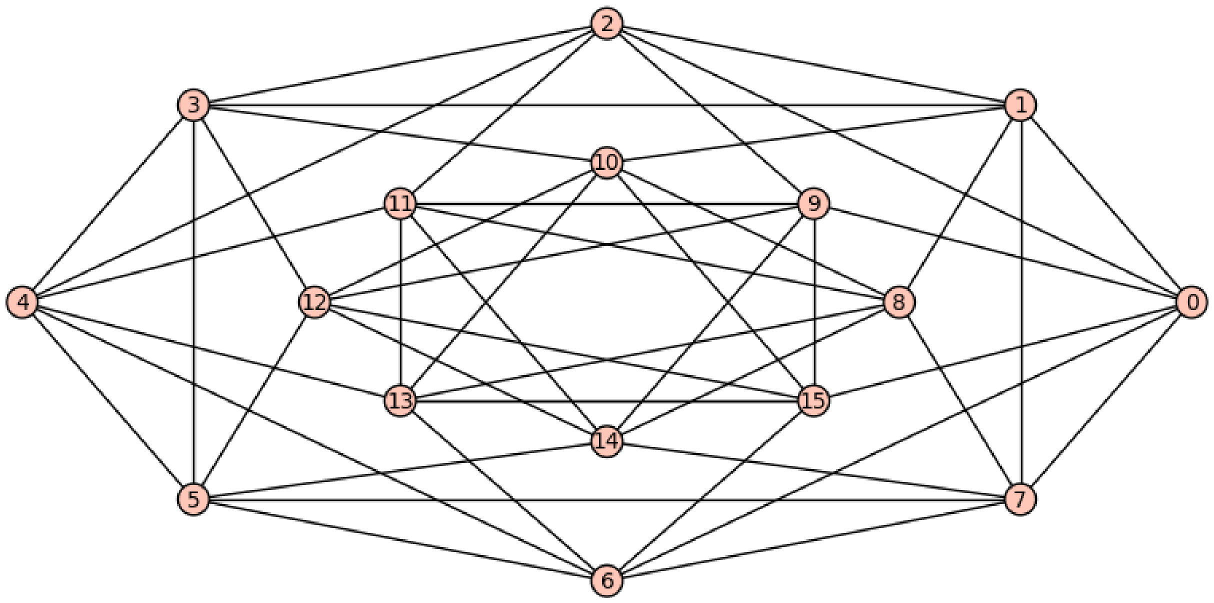
- overall NetworkX looks more or less best, of all software they could evaluate back then.
- pleasant fact: NetworkX is an open source Python library :-)
- NetworkX is from Los Alamos, and targets applications (not pure math)

Included NetworkX in Sage, built our own graph data types, added functionality, etc., to make it more useful for pure math as well. Tons of work over time with many other contributors.
See the overview of functionality at `http://www.sagemath.org/doc/reference/graphs/`

1.5 The Graph Catalog

(mainly Emily Kirkman)

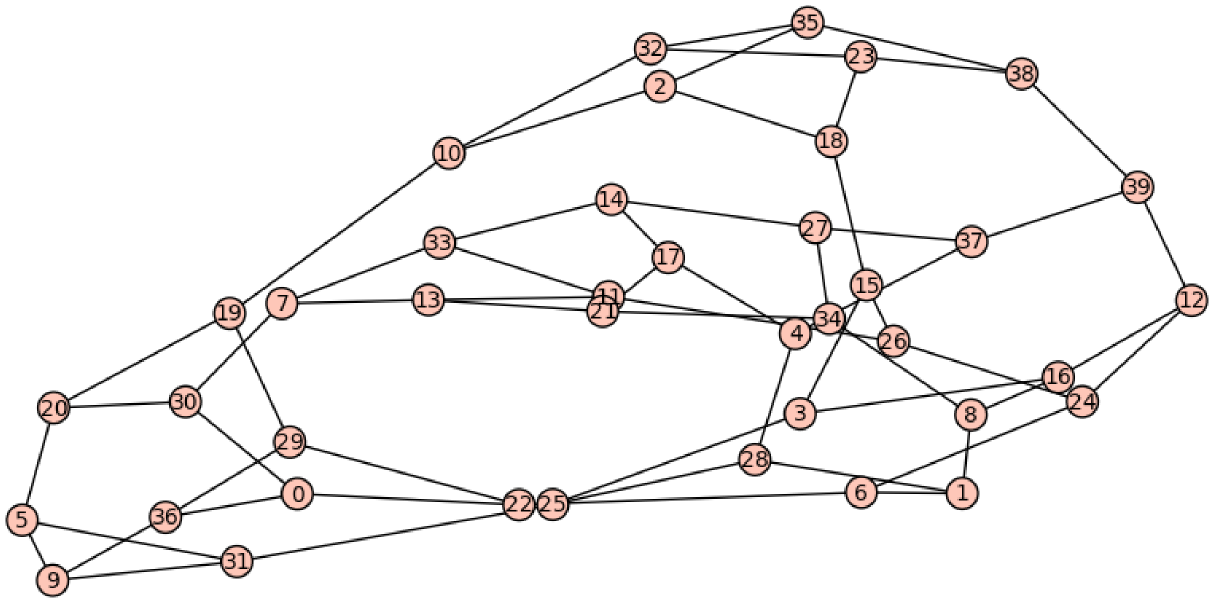
```
plot(graphs.ShrikhandeGraph())
```



```
dir(graphs)
```

```
['Balaban10Cage', 'Balaban11Cage', 'BalancedTree', 'BarbellGraph', 'BidiakisCube',
'BiggsSmithGraph', 'BishopGraph', 'BlanusaFirstSnarkGraph', 'BlanusaSecondSnarkGraph',
'BrinkmannGraph', 'BrouwerHaemersGraph', 'BubbleSortGraph', 'BuckyBall', 'BullGraph',
'ButterflyGraph', 'CameronGraph', 'ChessboardGraphGenerator', 'ChvatalGraph',
'CirculantGraph', 'CircularLadderGraph', 'ClawGraph', 'ClebschGraph',
'CompleteBipartiteGraph', 'CompleteGraph', 'CompleteMultipartiteGraph', 'CoxeterGraph',
'CubeGraph', 'CycleGraph', 'DegreeSequence', 'DegreeSequenceBipartite',
'DegreeSequenceConfigurationModel', 'DegreeSequenceExpected', 'DegreeSequenceTree',
'DesarguesGraph', 'DiamondGraph', 'DodecahedralGraph', 'DorogovtsevGoltsevMendesGraph',
'DoubleStarSnark', 'DurerGraph', 'DyckGraph', 'EllinghamHorton54Graph',
'EllinghamHorton78Graph', 'EmptyGraph', 'ErreraGraph', 'FibonacciTree', 'FlowerSnark',
'FoldedCubeGraph', 'FolkmanGraph', 'FosterGraph', 'FranklinGraph', 'FriendshipGraph',
'FruchtGraph', 'FuzzyBallGraph', 'GeneralizedPetersenGraph', 'GoldnerHararyGraph',
'GrayGraph', 'Grid2dGraph', 'GridGraph', 'GrotzschGraph', 'HallJankoGraph',
'HanoiTowerGraph', 'HararyGraph', 'HarriesGraph', 'HarriesWongGraph', 'HeawoodGraph',
'HerschelGraph', 'HexahedralGraph', 'HigmanSimsGraph', 'HoffmanGraph',
'HoffmanSingletonGraph', 'HoltGraph', 'HortonGraph', 'HouseGraph', 'HouseXGraph',
'HyperStarGraph', 'IcosahedralGraph', 'IntervalGraph', 'JohnsonGraph', 'KingGraph',
'KittellGraph', 'KneserGraph', 'KnightGraph', 'KrackhardtKiteGraph', 'LCFGraph',
'LadderGraph', 'LjubljanaGraph', 'LollipopGraph', 'M22Graph', 'MarkstroemGraph',
'McGeeGraph', 'McLaughlinGraph', 'MeredithGraph', 'MoebiusKantorGraph', 'MoserSpindle',
'MycielskiGraph', 'MycielskiStep', 'NKStarGraph', 'NStarGraph', 'NauruGraph',
'OctahedralGraph', 'OddGraph', 'PaleyGraph', 'PappusGraph', 'PathGraph',
'PermutationGraph', 'PetersenGraph', 'PoussinGraph', 'QueenGraph', 'RandomBarabasiAlbert',
'RandomBipartite', 'RandomBoundedToleranceGraph', 'RandomGNM', 'RandomGNP',
'RandomHolmeKim', 'RandomInterval', 'RandomIntervalGraph', 'RandomLobster',
'RandomNewmanWattsStrogatz', 'RandomRegular', 'RandomShell', 'RandomToleranceGraph',
'RandomTree', 'RandomTreePowerlaw', 'RingedTree', 'RobertsonGraph', 'RookGraph',
'SchlaefliGraph', 'ShrikhandeGraph', 'SimsGewirtzGraph', 'SousselierGraph', 'StarGraph',
'SylvesterGraph', 'SymplecticGraph', 'SzekeresSnarkGraph', 'TetrahedralGraph',
'ThomsenGraph', 'TietzeGraph', 'ToleranceGraph', 'Toroidal6RegularGrid2dGraph',
'ToroidalGrid2dGraph', 'Tutte12Cage', 'TutteCoxeterGraph', 'TutteGraph', 'WagnerGraph',
'WatkinsSnarkGraph', 'WellsGraph', 'WheelGraph', 'WienerArayaGraph', 'WorldMap',
'__call__', '__doc__', '__module__', 'cospectral_graphs', 'fullerenes', 'fusenes',
'line_graph_forbidden_subgraphs', 'nauty_geng', 'petersen_family', 'sage', 'trees']
```

```
# peer grading graph :-)
g = graphs.RandomRegular(3,40,seed=8)
plot(g)
```



```
g.is_planar()
False
```

```
# I tried this and it "took forever", illustrating how combinatorial \
    algorithm often have
# exponential complexity...
g.genus()
Error in lines 3-3
Traceback (most recent call last):
  File "/projects/74af30b7-ad25-4308-a02e-c71fcd84de6e/.sagemathcloud/sage_server.py",
line 733, in execute
    exec compile(block+'\n', '', 'single') in namespace, locals
  File "", line 1, in <module>
  File "/usr/local/sage/sage-6.2.rc0/local/lib/python2.7/site-
packages/sage/graphs/generic_graph.py", line 4052, in genus
    g += genus.simple_connected_graph_genus(H, set_embedding = True, check = False,
minimal = True)
  File "genus.pyx", line 743, in sage.graphs.genus.simple_connected_graph_genus
(sage/graphs/genus.c:4508)
  File "genus.pyx", line 570, in
sage.graphs.genus.simple_connected_genus_backtracker.genus (sage/graphs/genus.c:3242)
  File "c_lib.pyx", line 73, in sage.ext.c_lib.sig_raise_exception (sage/ext/c_lib.c:872)
```

```
KeyboardInterrupt
```

1.6 The Graph Database

(mainly Jason Grout and Emily Kirkman)

See http://www.sagemath.org/doc/reference/graphs/sage/graphs/graph_database.html

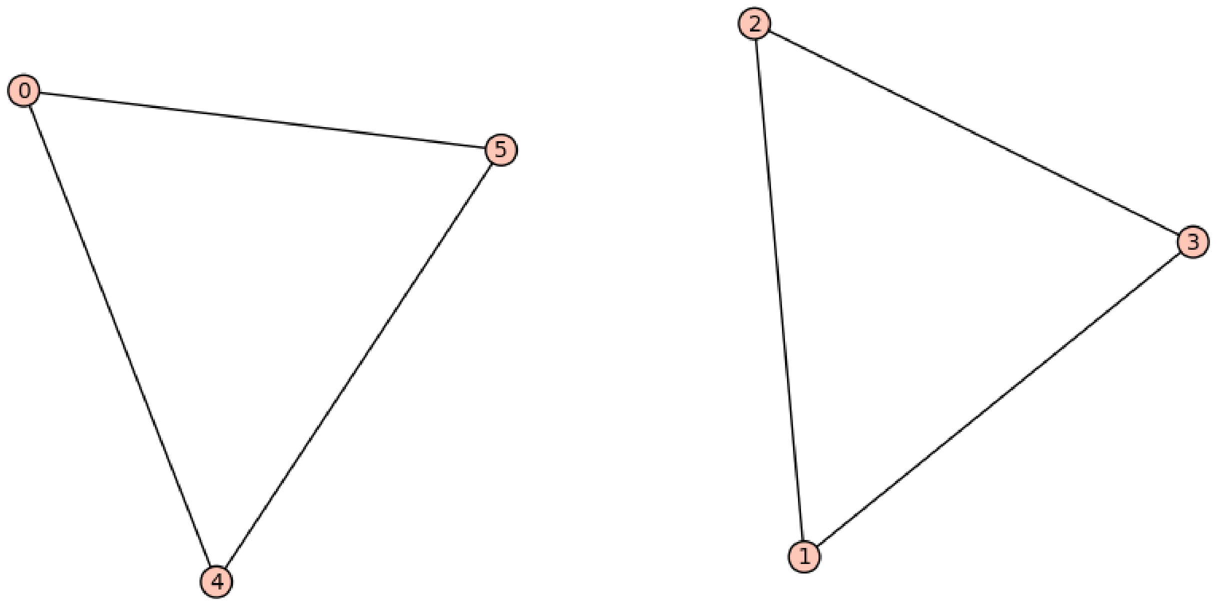
```
Q = GraphQuery(display_cols=['graph6','num_vertices','degree_sequence'],
               num_edges=['<=',6],
               min_degree=2)
Q.number_of()
```

```
9
```

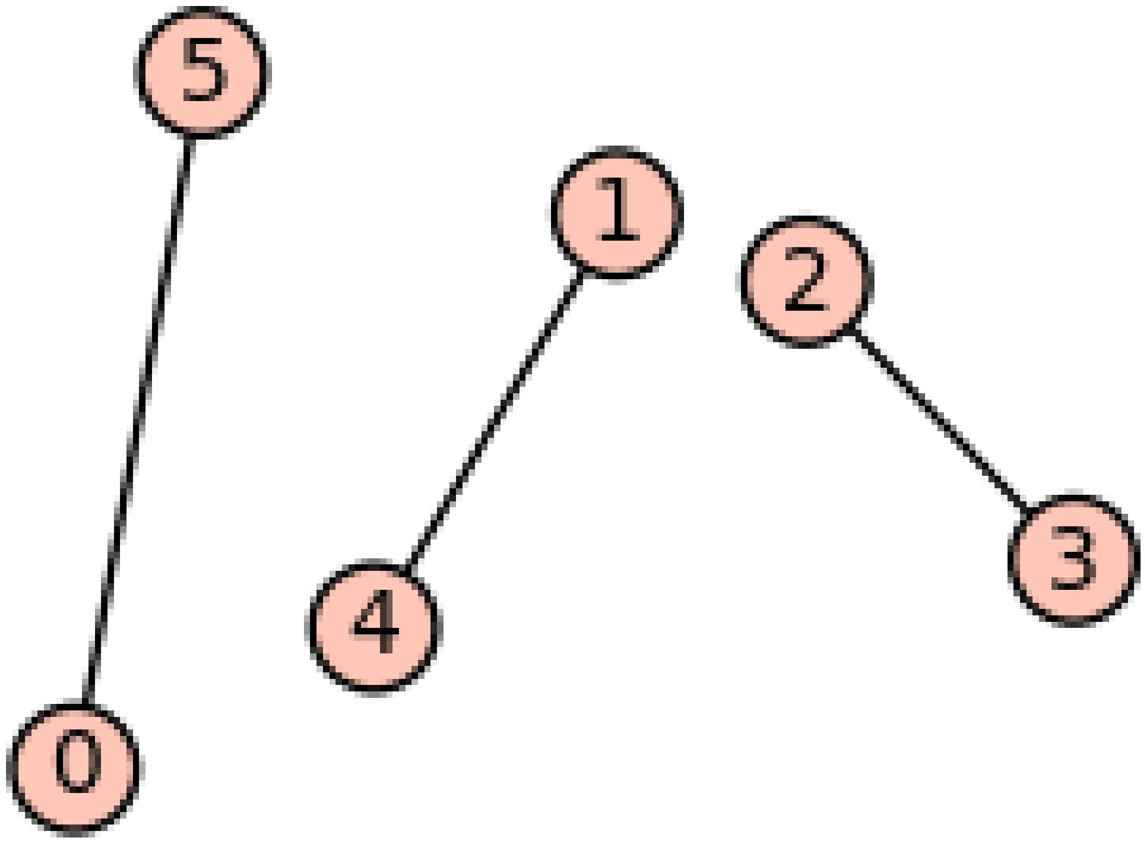
```
Q.show()
```

```
Graph6               Num Vertices         Degree Sequence
------------------------------------------------------------
Bw                   3                    [2, 2, 2]
C]                   4                    [2, 2, 2, 2]
C^                   4                    [2, 2, 3, 3]
DFw                  5                    [2, 2, 2, 3, 3]
DK{                  5                    [2, 2, 2, 2, 4]
DLo                  5                    [2, 2, 2, 2, 2]
Dbk                  5                    [2, 2, 2, 3, 3]
EIe_                 6                    [2, 2, 2, 2, 2, 2]
EJaG                 6                    [2, 2, 2, 2, 2, 2]
```

```
g = Graph('EJaG'); plot(g)
```



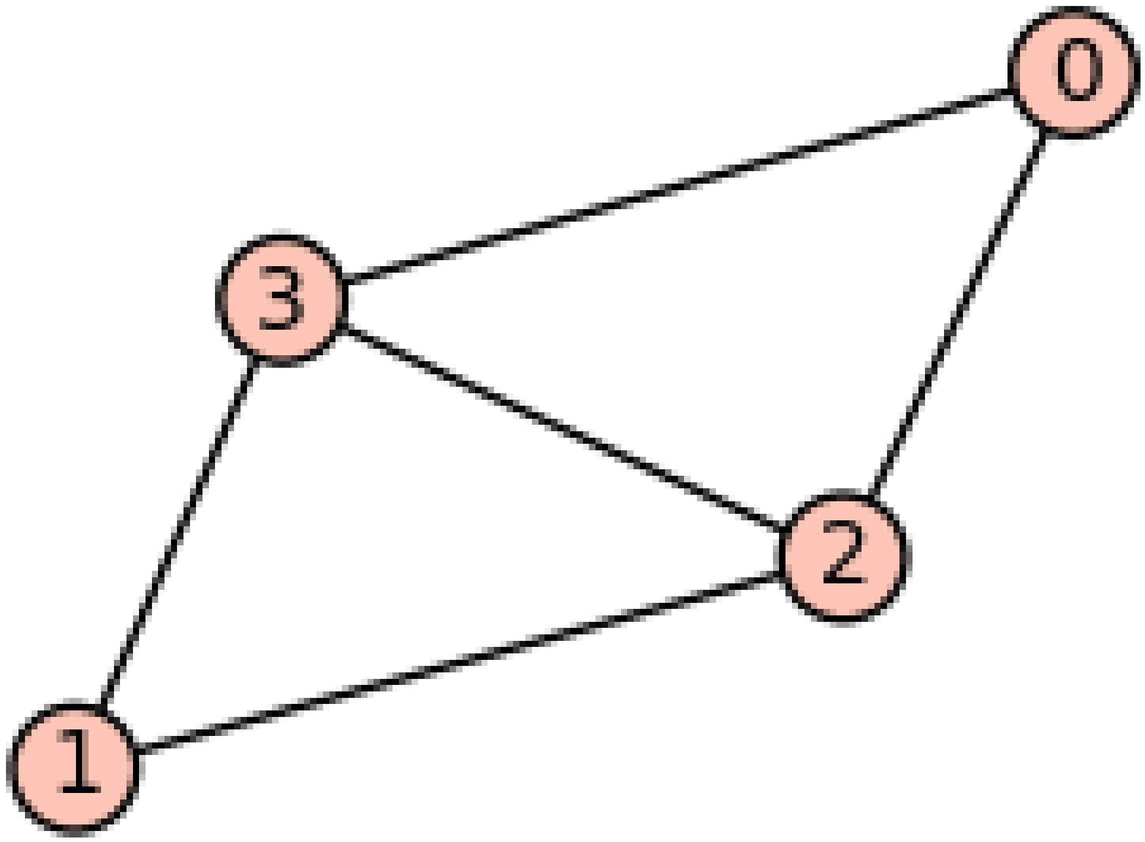
```
plot(g, figsize=2)
```



```
Q = GraphQuery(display_cols=['graph6','num_vertices','degree_sequence'],\
    num_edges=['<=',5],min_degree=2)
Q.number_of()
4
```

```
Q.show()
Graph6                  Num Vertices            Degree Sequence
------------------------------------------------------------------
Bw                      3                       [2, 2, 2]
C]                      4                       [2, 2, 2, 2]
C^                      4                       [2, 2, 3, 3]
DLo                     5                       [2, 2, 2, 2, 2]
```

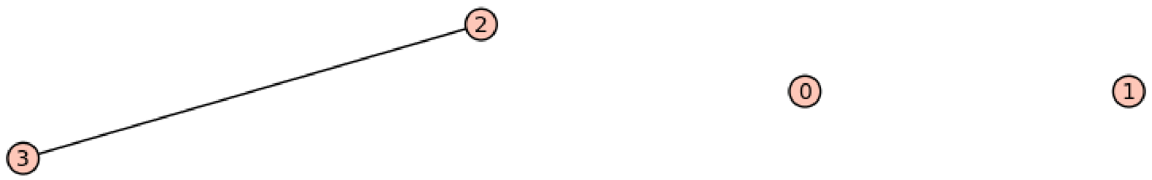
```
g = Graph('C^'); plot(g, figsize=2)
```



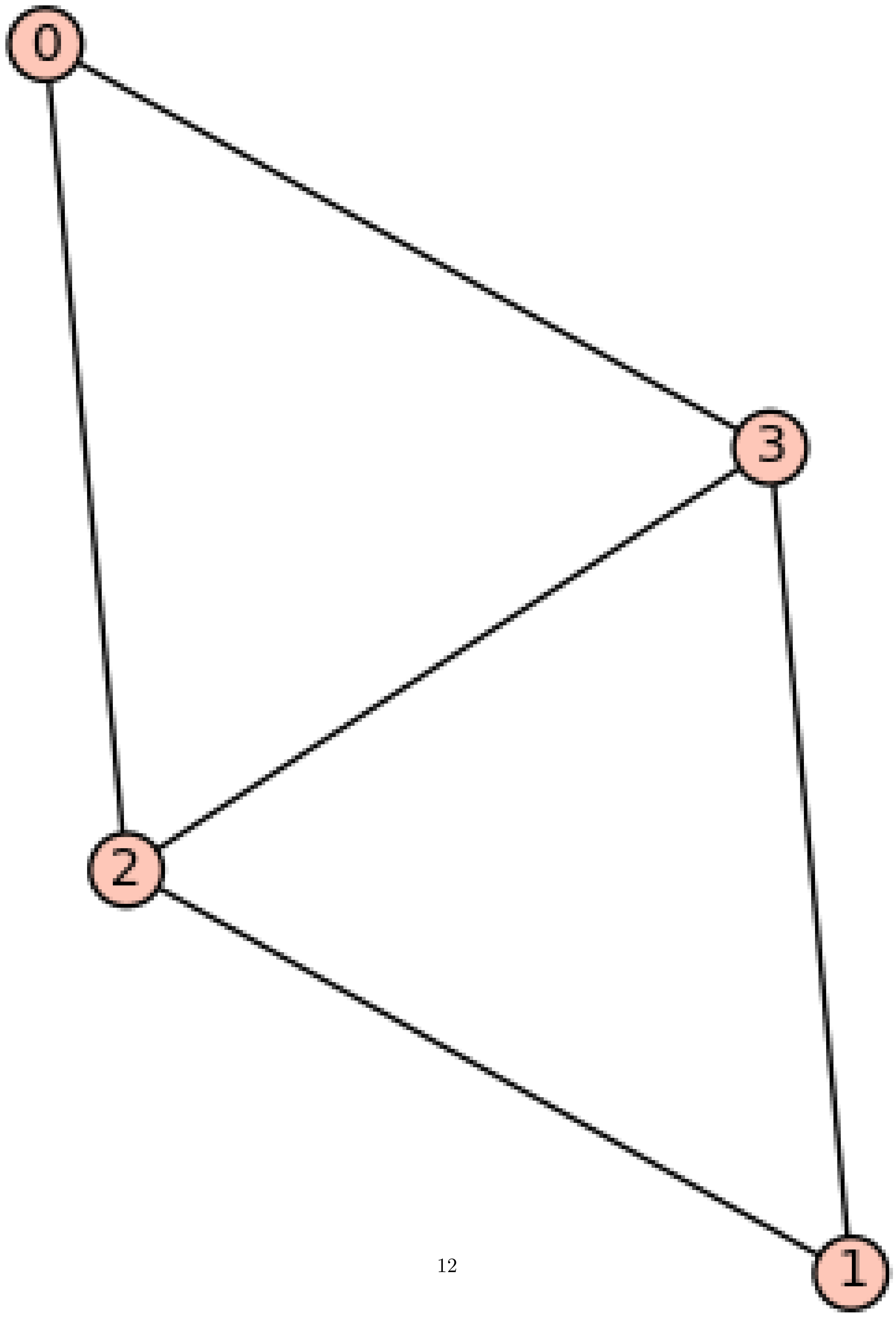
```
Q = GraphQuery(display_cols=['graph6','num_vertices','aut_grp_size'], \
   num_vertices=4, aut_grp_size=4)
Q.show()
```

```
Graph6               Num Vertices         Aut Grp Size
--------------------------------------------------------------
C@                   4                    4
C^                   4                    4
```

```
Graph('C@').plot()
```



```
Graph('C^').plot()
```


0
3
2
1

```
for k in [1..9]:
    print k, GraphQuery(display_cols=['num_vertices'], num_vertices=k).\
       number_of()
```

```
1 1
2 2
3 4
4 11
5 34
6 156
7 1044
8 0
9 0
```

1.7 Automorphisms, Subgraphs, Isomorphism

(Robert Miller)

Nauty and NICE.

Nauty: http://pallini.di.uniroma1.it/

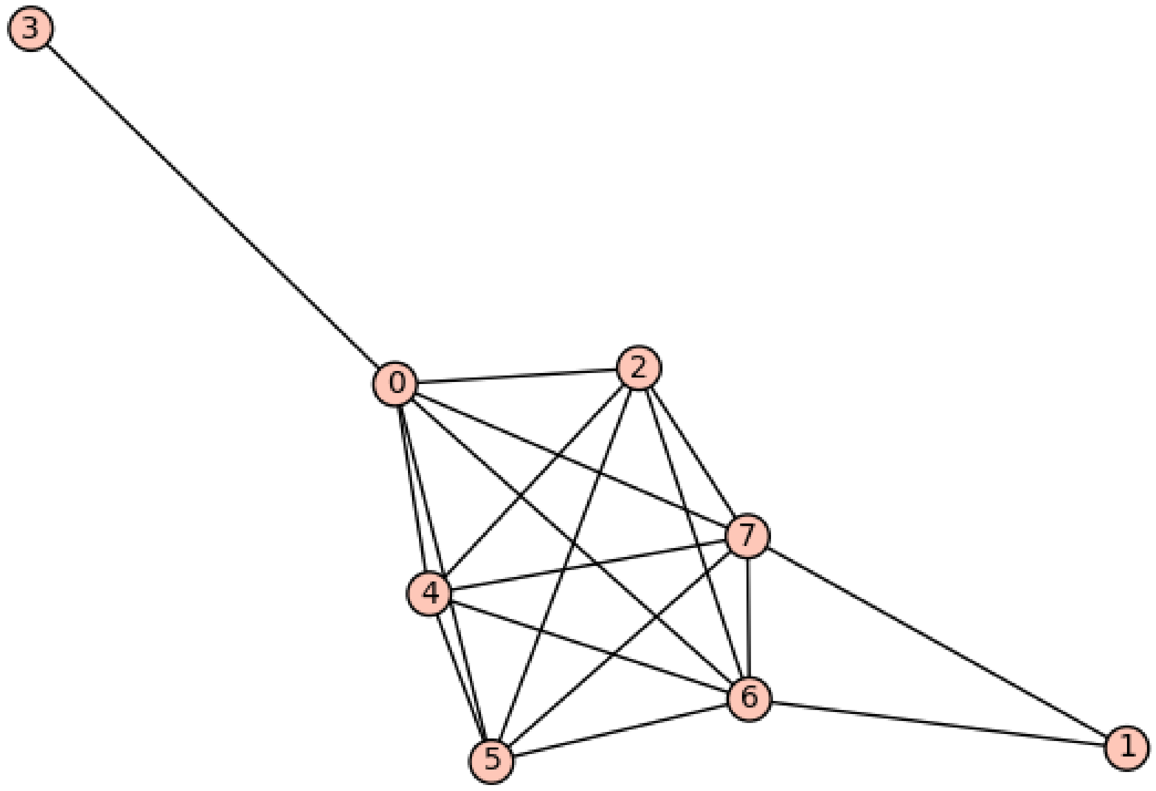
The license is open source by GPL-incompatible: nauty is copyright (1984-2014) Brendan McKay. All rights reserved. Traces is copyright (2008-2014) Adolfo Piperno. All rights reserved. Permission is hereby given for use and/or distribution with the exception of sale for profit or application with nontrivial military significance.

So no Nauty in Sage (or any other GPLd software).

Hence: Robert Miller (spent years) writing NICE, which is a Cython implementation from scratch of a similar algorithm.

```
set_random_seed(2)
g = graphs.RandomBoundedToleranceGraph(8)
```

```
g.plot()
```



```
S = g.automorphism_group(); S
Permutation Group with generators [(6,7), (4,5), (2,4)]
```

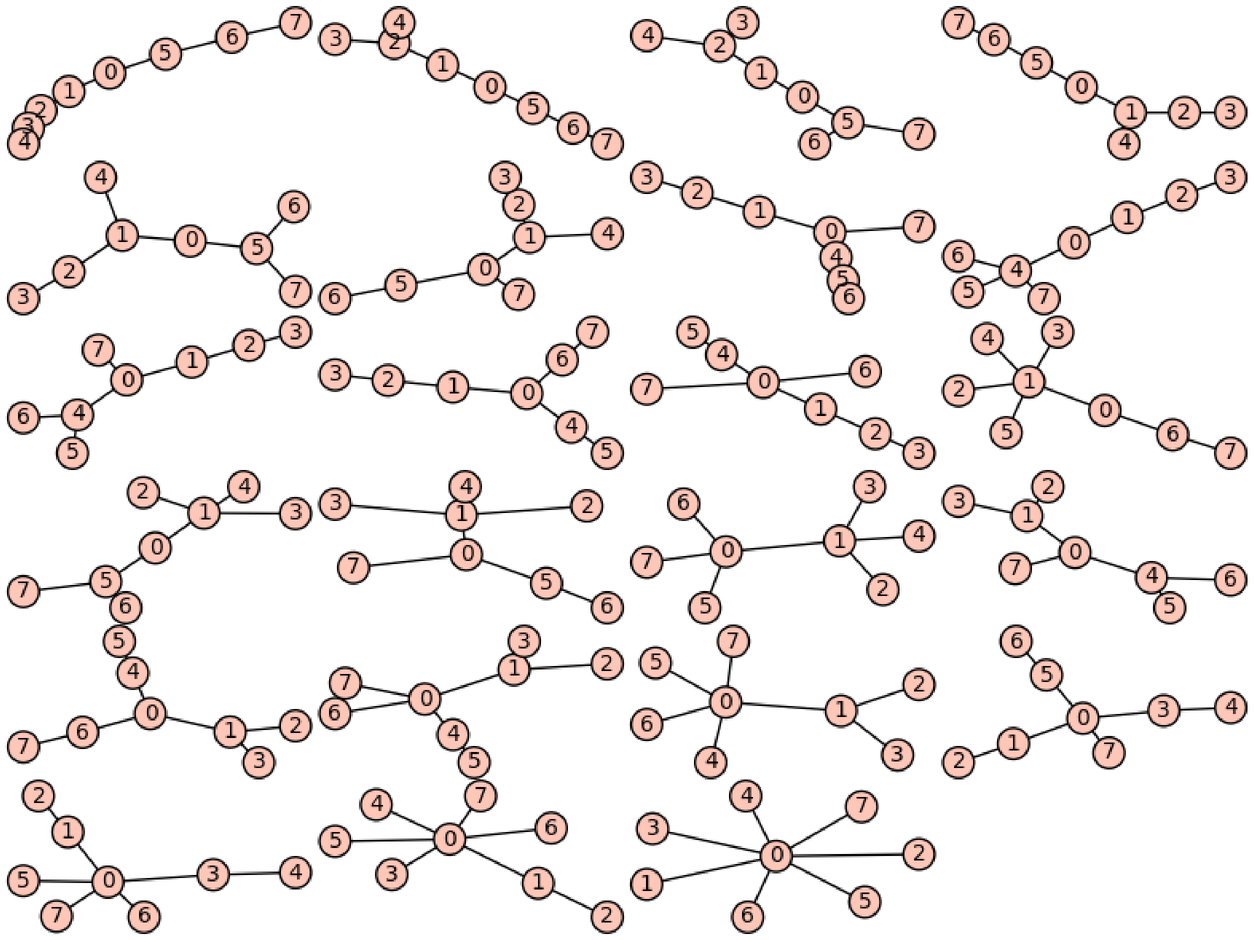
```
S.cardinality()
12
```

```
g.is_planar()
False
```

1.8 Enumerating Graphs

For example, a tree is a connected graph with no cycles.

```
from sage.graphs.trees import TreeIterator
v = [plot(t) for t in TreeIterator(8)]
len(v)
graphics_array(v, 6,4)
23
```



```
%time
# How many trees with 17 vertices?  Let's enumerate them *all*...
count = 0
for t in TreeIterator(17):
   count += 1
print count
48629
CPU time: 1.95 s, Wall time: 1.95 s
```