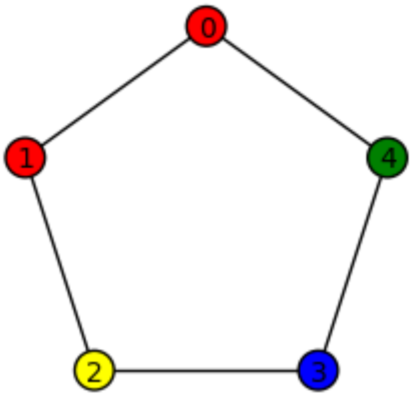
examples

```
attach "/Users/araichev/graph_dynamics/graph_dynamics.py"

# For a list of Sage's graph generators, see
http://wiki.sagemath.org/graph_generators.
```

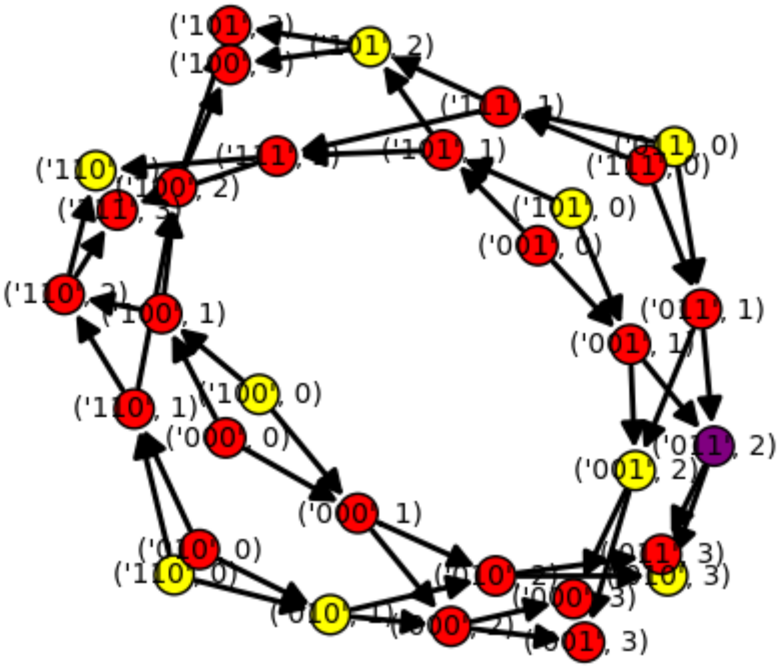
```
# Example: Use color()

G = graphs.CycleGraph(5)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green'])
print(coloring)
G.show(vertex_colors=invert_dict(coloring), figsize=3)
```

```
{0: 'red', 1: 'red', 2: 'yellow', 3: 'blue', 4: 'green'}
```

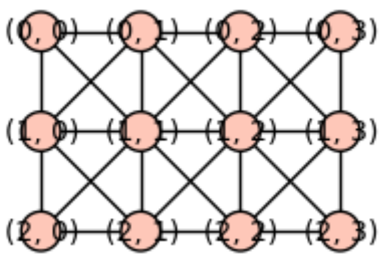



```
# Example: Use color_randomly() and color_count()

G = digraphs.ButterflyGraph(3)
coloring = color_randomly(G, {'red': 0.6, 'yellow': 0.3,
'purple': 1/10})
print(coloring)
print('Color count = {!s}'.format(color_count(coloring)))
G.show(vertex_colors=invert_dict(coloring), figsize=5)
```

```
{('101', 1): 'red', ('111', 3): 'red', ('001', 2): 'yellow',
0): 'yellow', ('110', 2): 'red', ('010', 2): 'red', ('000', 2
'red', ('011', 3): 'red', ('101', 0): 'yellow', ('001', 3): '
('100', 1): 'red', ('110', 1): 'red', ('010', 3): 'yellow', (
0): 'red', ('011', 2): 'purple', ('000', 1): 'red', ('101', 3
'red', ('001', 0): 'red', ('110', 0): 'yellow', ('111', 1): '
('011', 1): 'red', ('100', 2): 'red', ('010', 0): 'red', ('00
'red', ('101', 2): 'yellow', ('001', 1): 'red', ('111', 2): '
('011', 0): 'yellow', ('100', 3): 'red', ('110', 3): 'yellow'
('010', 1): 'yellow', ('000', 3): 'red'}
Color count = Counter({'red': 22, 'yellow': 9, 'purple': 1})
```

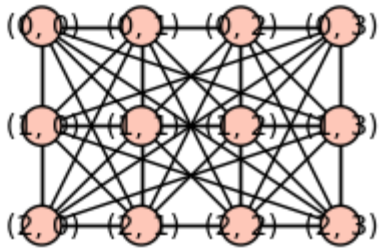


```
# Test the Moore lattice generator

G = moore_lattice(3, 4)
G.show(figsize=2)
print('num edges = {!s}'.format(G.num_edges()))
print('degrees = %s' % G.degree())
print(G.edges())


G = moore_lattice(3, 4, toroidal=True)
G.show(figsize=2)
print('num edges = %s' % G.num_edges())
print('degrees = %s' % G.degree())
print(G.edges())

G = moore_lattice(2, 4)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green',
'green', 'blue', 'blue'])
G.show(vertex_colors=invert_dict(coloring), figsize=3)
print(coloring)
```

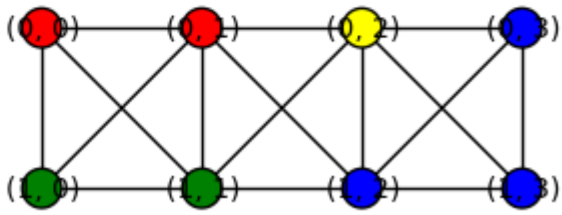


```
num edges = 29
degrees = [5, 8, 3, 5, 8, 3, 5, 3, 5, 5, 3, 5]
```

```
[((0, 0), (0, 1), None), ((0, 0), (1, 0), None), ((0, 0), (1,
None), ((0, 1), (0, 2), None), ((0, 1), (1, 0), None), ((0, 1
1), None), ((0, 1), (1, 2), None), ((0, 2), (0, 3), None), ((
(1, 1), None), ((0, 2), (1, 2), None), ((0, 2), (1, 3), None)
3), (1, 2), None), ((0, 3), (1, 3), None), ((1, 0), (1, 1), N
((1, 0), (2, 0), None), ((1, 0), (2, 1), None), ((1, 1), (1,
None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((1, 1
2), None), ((1, 2), (1, 3), None), ((1, 2), (2, 1), None), ((
(2, 2), None), ((1, 2), (2, 3), None), ((1, 3), (2, 2), None)
3), (2, 3), None), ((2, 0), (2, 1), None), ((2, 1), (2, 2), N
((2, 2), (2, 3), None)]
```



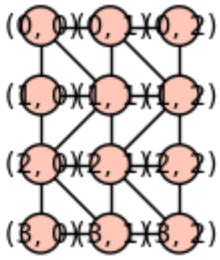
```
num edges = 48
degrees = [8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
[((0, 0), (0, 1), None), ((0, 0), (0, 3), None), ((0, 0), (1,
None), ((0, 0), (1, 1), None), ((0, 0), (1, 3), None), ((0, (
0), None), ((0, 0), (2, 1), None), ((0, 0), (2, 3), None), ((
(0, 2), None), ((0, 1), (1, 0), None), ((0, 1), (1, 1), None)
1), (1, 2), None), ((0, 1), (2, 0), None), ((0, 1), (2, 1), N
((0, 1), (2, 2), None), ((0, 2), (0, 3), None), ((0, 2), (1,
None), ((0, 2), (1, 2), None), ((0, 2), (1, 3), None), ((0, 2
1), None), ((0, 2), (2, 2), None), ((0, 2), (2, 3), None), ((
(1, 0), None), ((0, 3), (1, 2), None), ((0, 3), (1, 3), None)
3), (2, 0), None), ((0, 3), (2, 2), None), ((0, 3), (2, 3), N
((1, 0), (1, 1), None), ((1, 0), (1, 3), None), ((1, 0), (2,
None), ((1, 0), (2, 1), None), ((1, 0), (2, 3), None), ((1, 1
2), None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((
(2, 2), None), ((1, 2), (1, 3), None), ((1, 2), (2, 1), None)
2), (2, 2), None), ((1, 2), (2, 3), None), ((1, 3), (2, 0), N
((1, 3), (2, 2), None), ((1, 3), (2, 3), None), ((2, 0), (2,
None), ((2, 0), (2, 3), None), ((2, 1), (2, 2), None), ((2, 2
3), None)]
```



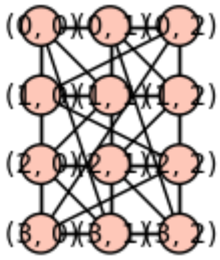
```
{(0, 1): 'red', (1, 2): 'blue', (0, 0): 'red', (0, 2): 'yello
3): 'blue', (1, 0): 'green', (0, 3): 'blue', (1, 1): 'green'}
```

```
# Test the triangular lattice generator

G = triangular_lattice(4, 3)
G.show(figsize=2)
print('num edges = {!s}'.format(G.num_edges()))
```

```
print('degrees = %s' % G.degree())
print(G.edges())


G = triangular_lattice(4, 3, toroidal=True)
G.show(figsize=2)
print('num edges = %s' % G.num_edges())
print('degrees = %s' % G.degree())
print(G.edges())

G = triangular_lattice(2, 4)
coloring = color(G, ['red', 'red', 'yellow', 'blue', 'green',
'green', 'blue', 'blue'])
G.show(vertex_colors=invert_dict(coloring), figsize=3)
print(coloring)
```

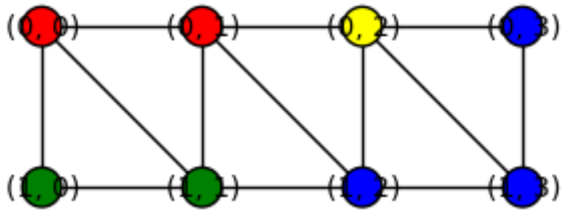


```
num edges = 23
degrees = [4, 5, 3, 3, 2, 4, 6, 6, 5, 3, 3, 2]
[((0, 0), (0, 1), None), ((0, 0), (1, 0), None), ((0, 0), (1,
None), ((0, 1), (0, 2), None), ((0, 1), (1, 1), None), ((0, 1
2), None), ((0, 2), (1, 2), None), ((1, 0), (1, 1), None), ((
(2, 0), None), ((1, 1), (1, 2), None), ((1, 1), (2, 0), None)
1), (2, 1), None), ((1, 2), (2, 1), None), ((1, 2), (2, 2), N
((2, 0), (2, 1), None), ((2, 0), (3, 0), None), ((2, 0), (3,
None), ((2, 1), (2, 2), None), ((2, 1), (3, 1), None), ((2, 1
2), None), ((2, 2), (3, 2), None), ((3, 0), (3, 1), None), ((
(3, 2), None)]
```



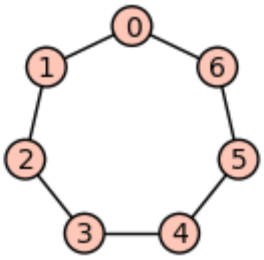
```
num edges = 36
degrees = [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]
[((0, 0), (0, 1), None), ((0, 0), (0, 2), None), ((0, 0), (1,
None), ((0, 0), (1, 1), None), ((0, 0), (3, 0), None), ((0, (
1), None), ((0, 1), (0, 2), None), ((0, 1), (1, 1), None), ((
(1, 2), None), ((0, 1), (3, 1), None), ((0, 1), (3, 2), None)
2), (1, 0), None), ((0, 2), (1, 2), None), ((0, 2), (3, 0), N
((0, 2), (3, 2), None), ((1, 0), (1, 1), None), ((1, 0), (1,
None), ((1, 0), (2, 0), None), ((1, 0), (2, 2), None), ((1, 1
```

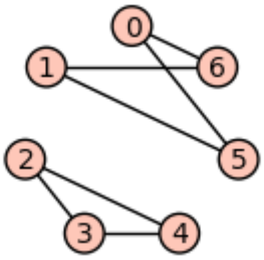
```
2), None), ((1, 1), (2, 0), None), ((1, 1), (2, 1), None), ((
(2, 1), None), ((1, 2), (2, 2), None), ((2, 0), (2, 1), None)
0), (2, 2), None), ((2, 0), (3, 0), None), ((2, 0), (3, 1), N
((2, 1), (2, 2), None), ((2, 1), (3, 1), None), ((2, 1), (3,
None), ((2, 2), (3, 0), None), ((2, 2), (3, 2), None), ((3, (
1), None), ((3, 0), (3, 2), None), ((3, 1), (3, 2), None)]
```

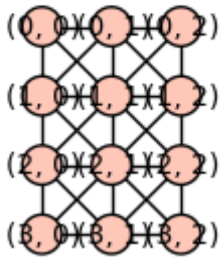


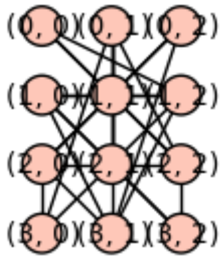
```
{(0, 1): 'red', (1, 2): 'blue', (0, 0): 'red', (0, 2): 'yellc
3): 'blue', (1, 0): 'green', (0, 3): 'blue', (1, 1): 'green'}
```

```
# Test the Maslov-Sneppen rewiring method

G = graphs.CycleGraph(7)
G.show(figsize=2)
H = maslov_sneppen(G)
H.show(figsize=2)

G = moore_lattice(4, 3)
G.show(figsize=2)
H = maslov_sneppen(G)
H.show(figsize=2)
print(G.num_verts() == H.num_verts())
print(G.num_edges() == H.num_edges())
print(G.degree() == H.degree())

G = graphs.RandomBarabasiAlbert(50, 5)
H = maslov_sneppen(G)
print(G.num_verts() == H.num_verts())
print(G.num_edges() == H.num_edges())
print(G.degree() == H.degree())

G = digraphs.ButterflyGraph(2)
G.show(figsize=5)
H = maslov_sneppen(G)
H.show(figsize=5)
print(G.num_verts() == H.num_verts())
print(G.num_edges() == H.num_edges())
print(G.degree() == H.degree())
```

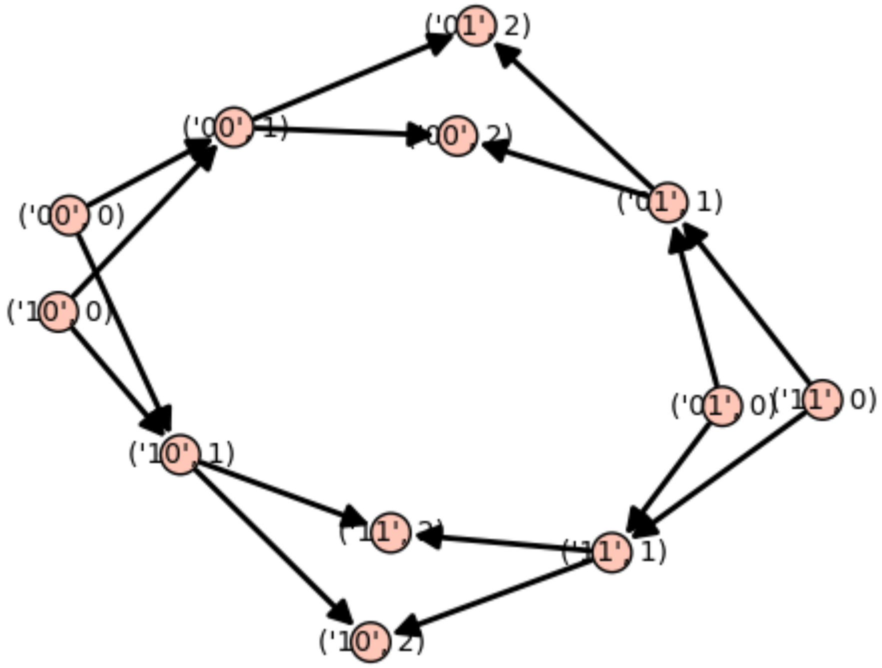


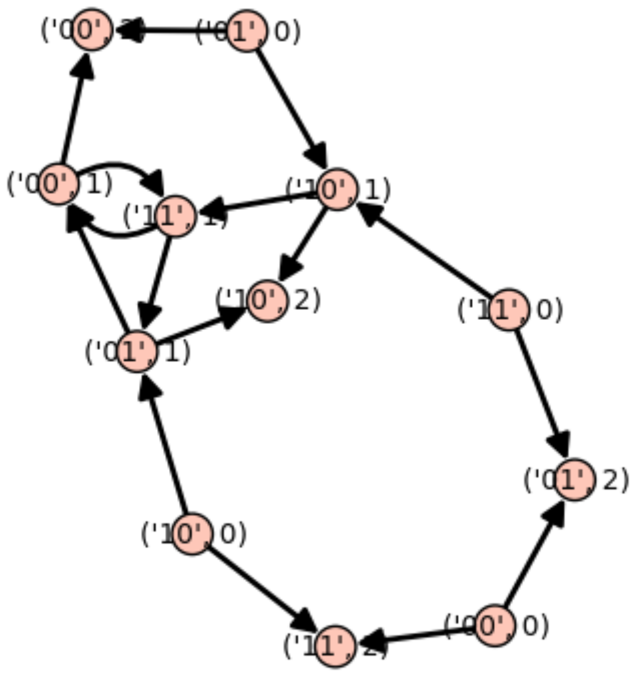






```
True
True
True
True
True
True
```



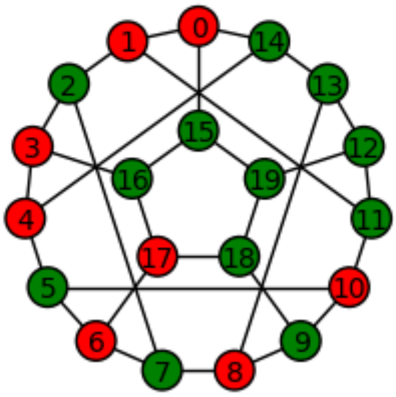


```
True
True
True
```

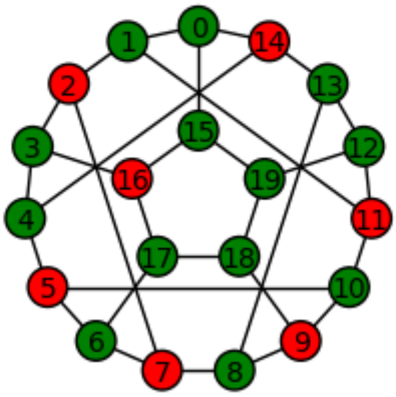
```
# Example: Run the majority rule

G = graphs.FlowerSnark()
color_bias = {'green': 0.7, 'red': 0.3}
ur = majority_rule
ur_kwargs = {}
initial_coloring = color_randomly(G, color_bias)
s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
```

```
print('Stabilized?\n    %s' % stabilized)
print(s)
show_colorings(G, s, vertex_labels=True)
```

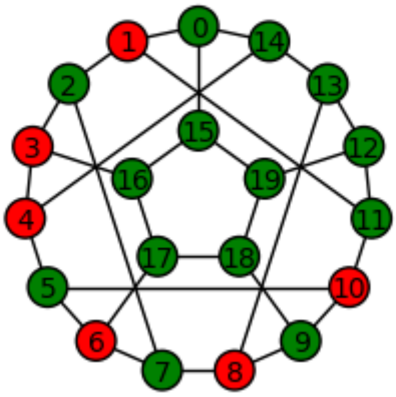
```
Stabilized?
    False
[{0: 'red', 1: 'red', 2: 'green', 3: 'red', 4: 'red', 5: 'gre
'red', 7: 'green', 8: 'red', 9: 'green', 10: 'red', 11: 'gree
'green', 13: 'green', 14: 'green', 15: 'green', 16: 'green',
'red', 18: 'green', 19: 'green'}, {0: 'green', 1: 'green', 2:
3: 'green', 4: 'green', 5: 'red', 6: 'green', 7: 'red', 8: 'g
9: 'red', 10: 'green', 11: 'red', 12: 'green', 13: 'green', 1
'red', 15: 'green', 16: 'red', 17: 'green', 18: 'green', 19:
'green'}, {0: 'green', 1: 'red', 2: 'green', 3: 'red', 4: 're
'green', 6: 'red', 7: 'green', 8: 'red', 9: 'green', 10: 'red
'green', 12: 'green', 13: 'green', 14: 'green', 15: 'green',
'green', 17: 'green', 18: 'green', 19: 'green'}, {0: 'green',
'green', 2: 'red', 3: 'green', 4: 'green', 5: 'red', 6: 'gree
'red', 8: 'green', 9: 'red', 10: 'green', 11: 'red', 12: 'gre
13: 'green', 14: 'green', 15: 'green', 16: 'green', 17: 'gree
'green', 19: 'green'}, {0: 'green', 1: 'red', 2: 'green', 3:
'green', 4: 'green', 5: 'green', 6: 'red', 7: 'green', 8: 're
'green', 10: 'red', 11: 'green', 12: 'green', 13: 'green', 14
'green', 15: 'green', 16: 'green', 17: 'green', 18: 'green',
'green'}, {0: 'green', 1: 'green', 2: 'green', 3: 'green', 4:
'green', 5: 'red', 6: 'green', 7: 'red', 8: 'green', 9: 'red'
'green', 11: 'red', 12: 'green', 13: 'green', 14: 'green', 15
'green', 16: 'green', 17: 'green', 18: 'green', 19: 'green'},
'green', 1: 'green', 2: 'green', 3: 'green', 4: 'green', 5: '
6: 'red', 7: 'green', 8: 'red', 9: 'green', 10: 'red', 11: 'g
12: 'green', 13: 'green', 14: 'green', 15: 'green', 16: 'gree
'green', 18: 'green', 19: 'green'}, {0: 'green', 1: 'green',
'green', 3: 'green', 4: 'green', 5: 'red', 6: 'green', 7: 're
'green', 9: 'red', 10: 'green', 11: 'green', 12: 'green', 13:
'green', 14: 'green', 15: 'green', 16: 'green', 17: 'green',
'green', 19: 'green'}, {0: 'green', 1: 'green', 2: 'green', 3
'green', 4: 'green', 5: 'green', 6: 'red', 7: 'green', 8: 're
'green', 10: 'red', 11: 'green', 12: 'green', 13: 'green', 14
'green', 15: 'green', 16: 'green', 17: 'green', 18: 'green',
'green'}, {0: 'green', 1: 'green', 2: 'green', 3: 'green', 4:
'green', 5: 'red', 6: 'green', 7: 'red', 8: 'green', 9: 'red'
'green', 11: 'green', 12: 'green', 13: 'green', 14: 'green',
'green', 16: 'green', 17: 'green', 18: 'green', 19: 'green'},
'green', 1: 'green', 2: 'green', 3: 'green', 4: 'green', 5: '
6: 'red', 7: 'green', 8: 'red', 9: 'green', 10: 'red', 11: 'g
12: 'green', 13: 'green', 14: 'green', 15: 'green', 16: 'gree
'green', 18: 'green', 19: 'green'}]
Step 0
```



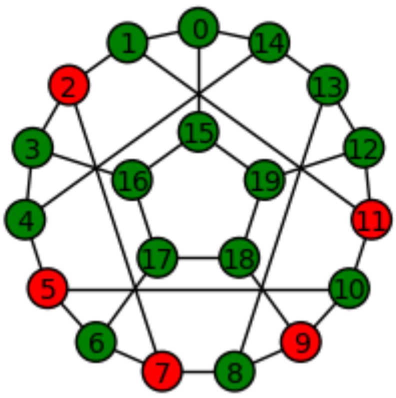
Step 1



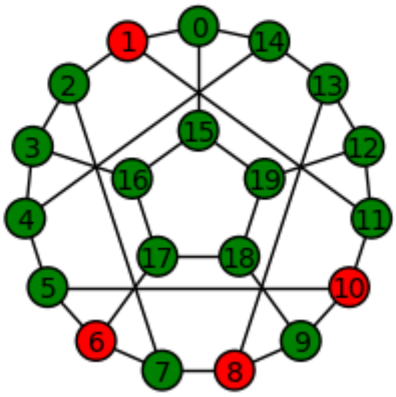
Step 2



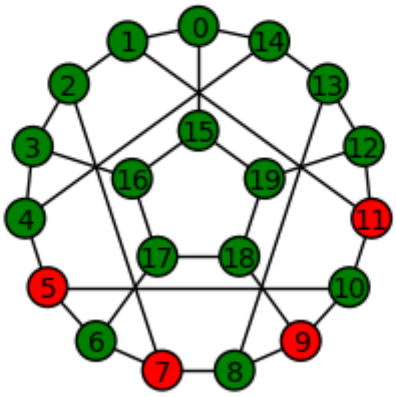
Step 3



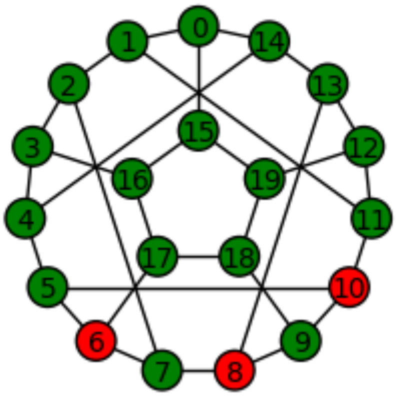
Step 4



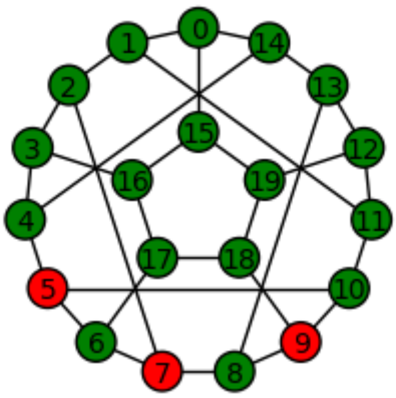
Step 5



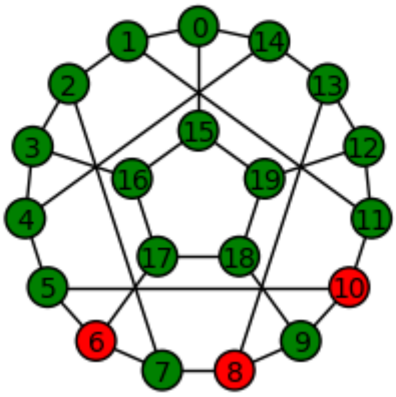
Step 6



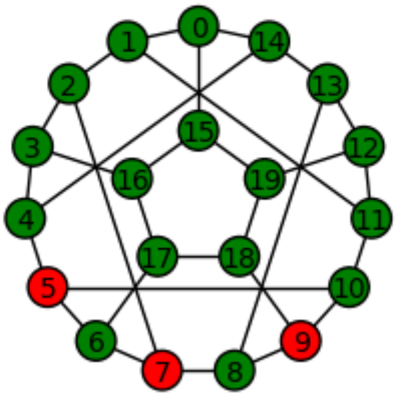
Step 7



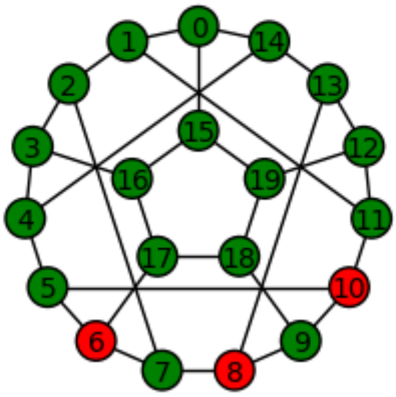
Step 8



Step 9



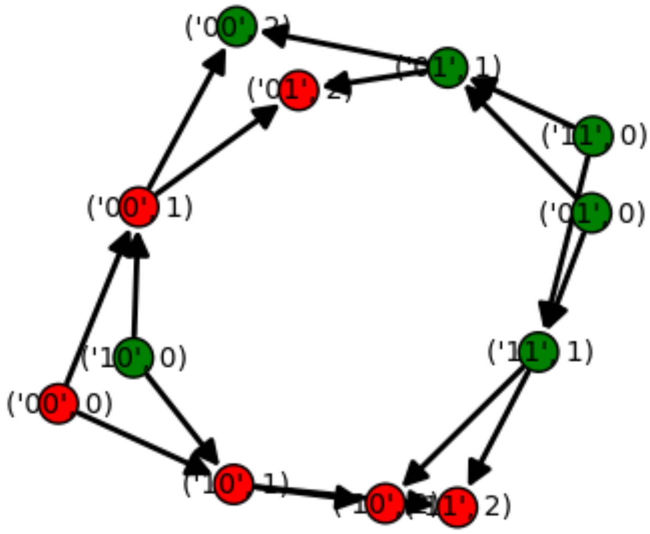
Step 10




```
# Example: Run the majority rule on a directed graph

G = digraphs.ButterflyGraph(2)
color_bias = {'green': 0.7, 'red': 0.3}
ur = majority_rule
ur_kwargs = {}
initial_coloring = color_randomly(G, color_bias)
s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
print(s)
show_colorings(G, s, vertex_labels=True, figsize=4)
```

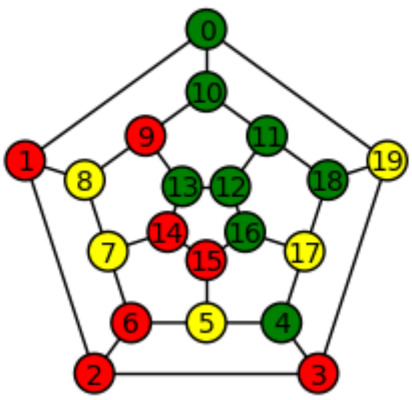
```
Stabilized?
    True
```

```
[{('11', 1): 'green', ('10', 2): 'red', ('01', 2): 'red', ('1
'green', ('01', 0): 'green', ('01', 1): 'green', ('00', 1): '
('11', 2): 'red', ('00', 0): 'red', ('00', 2): 'green', ('10'
'green', ('10', 1): 'red'}]
Step 0
```

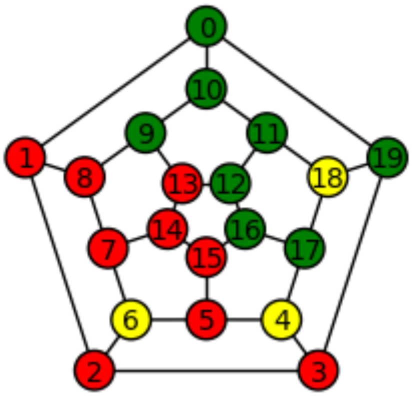



```
# Example: Run the plurality rule.

G = graphs.DodecahedralGraph()
ur = plurality_rule
ur_kwargs = {}
color_bias = {'green': 0.6, 'red': 0.3, 'yellow': 0.1}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s, vertex_labels=True)
```

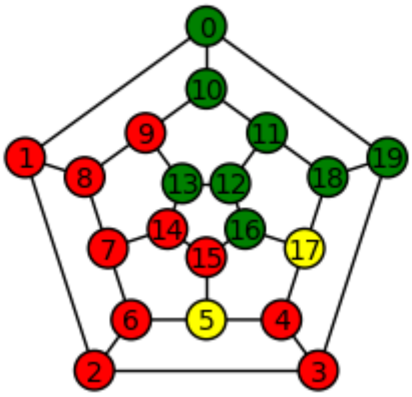
```
Stabilized?
    False
Step 0
```



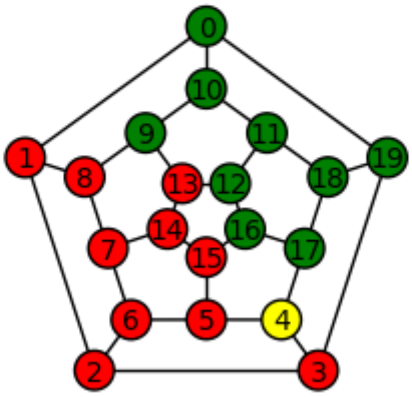

```
Step 1
```



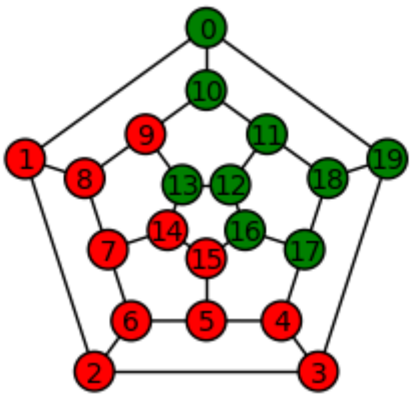
Step 2



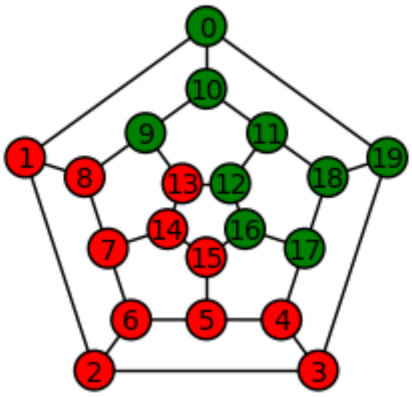
Step 3



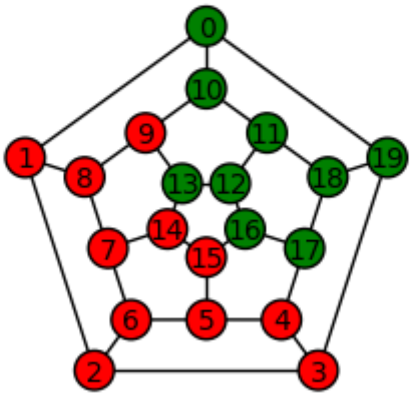
Step 4



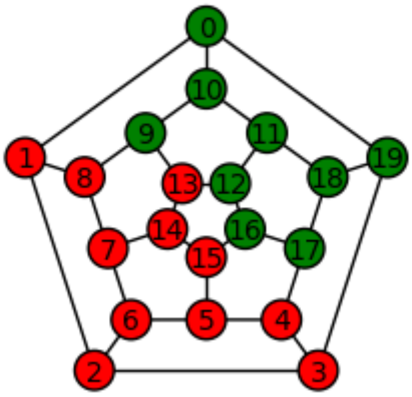
Step 5



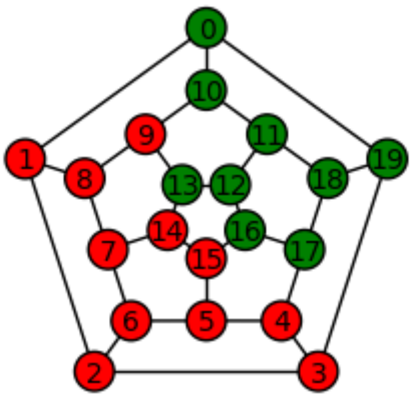
Step 6



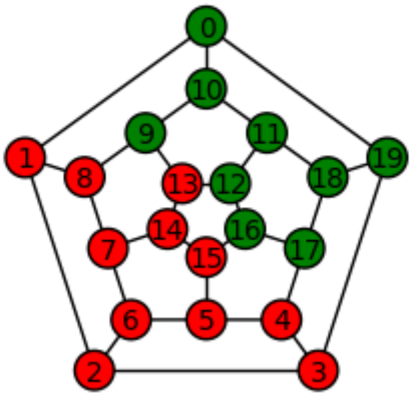
Step 7



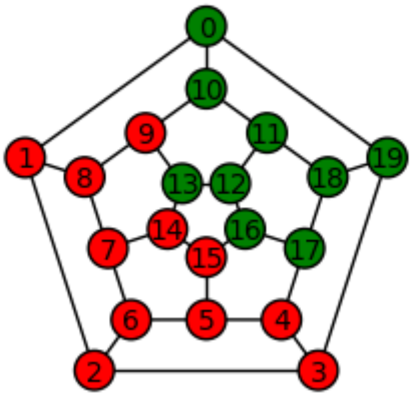
Step 8



Step 9

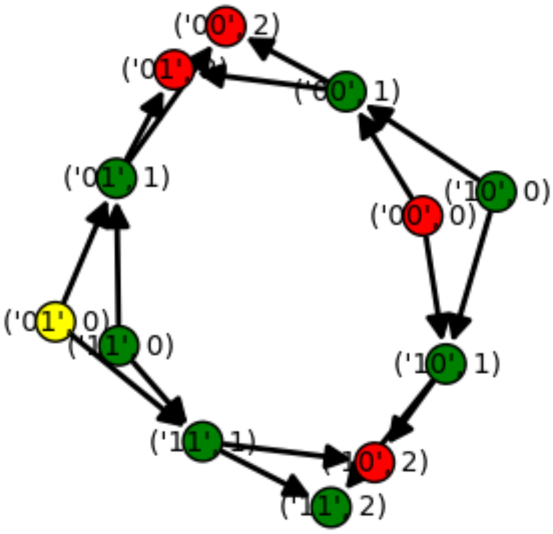



```
Step 10
```

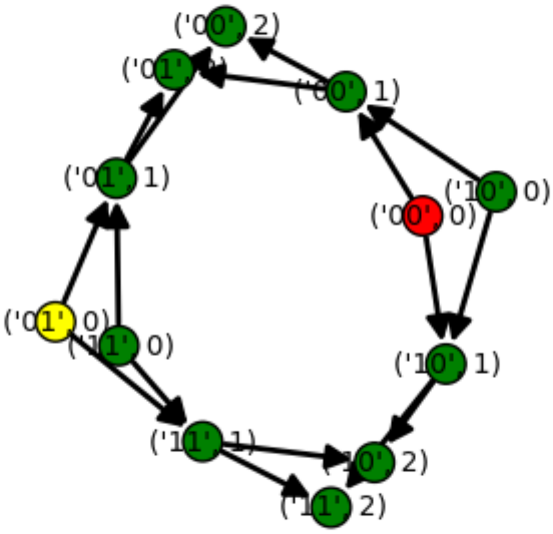



```
# Example: Run the plurality rule.

G = digraphs.ButterflyGraph(2)
ur = plurality_rule
ur_kwargs = {}
color_bias = {'green': 0.6, 'red': 0.3, 'yellow': 0.1}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s, vertex_labels=True, figsize=4)
```

```
Stabilized?
    True
Step 0
```

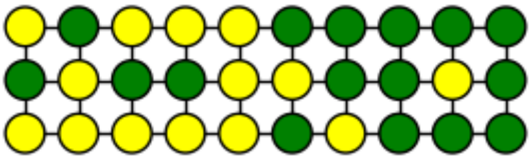


Step 1

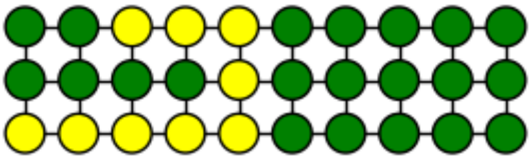


```
# Example: Run the GSL2 rule.

G = graphs.Grid2dGraph(3, 10)
color_bias = {'green': 0.6, 'yellow': 0.4}
ur = gsl2_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.7}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

```
Stabilized?
    True
Step 0
```

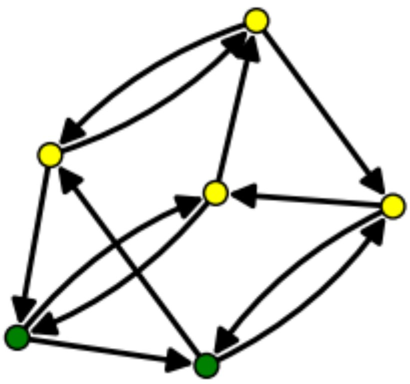


Step 1



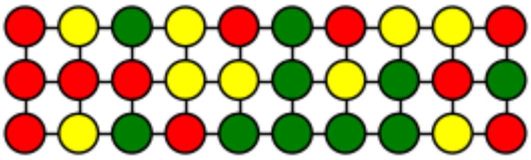
```
# Example: Run the GSL2 rule.

G = digraphs.Kautz(2, 2)
color_bias = {'green': 0.6, 'yellow': 0.4}
ur = gsl2_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.7}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

Stabilized?
    True
Step 0

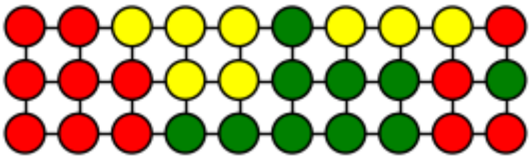


```
# Example: Run the GSL3 rule.

G = graphs.Grid2dGraph(3, 10)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

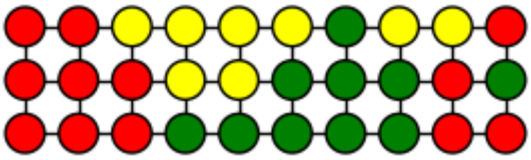
Stabilized?
    True
Step 0



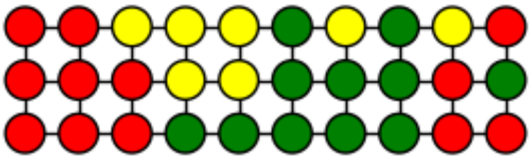
Step 1



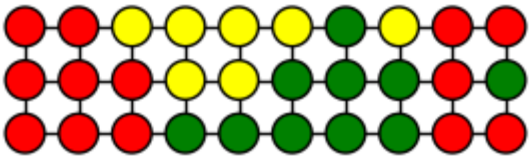
Step 2



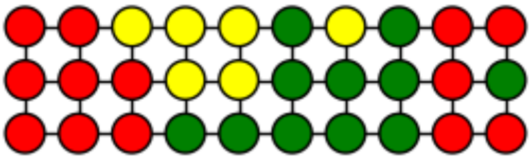
Step 3



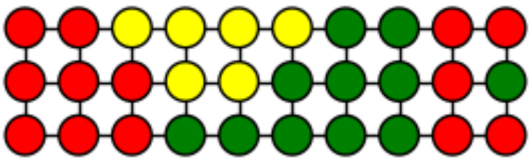
Step 4



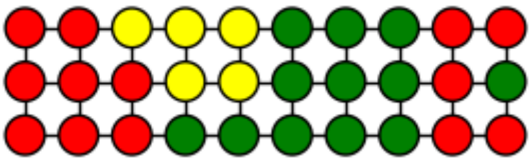
Step 5



Step 6

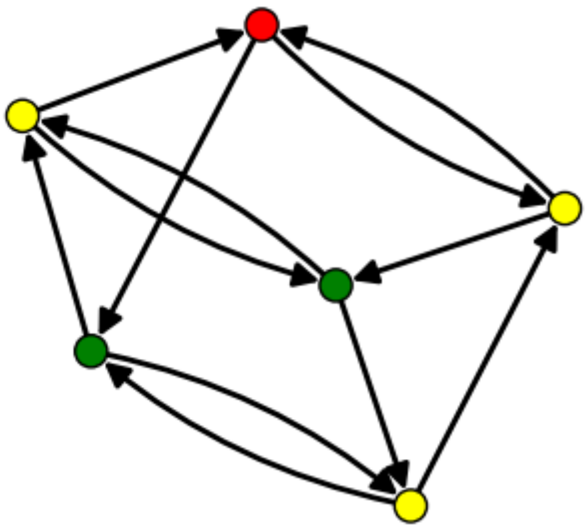


Step 7

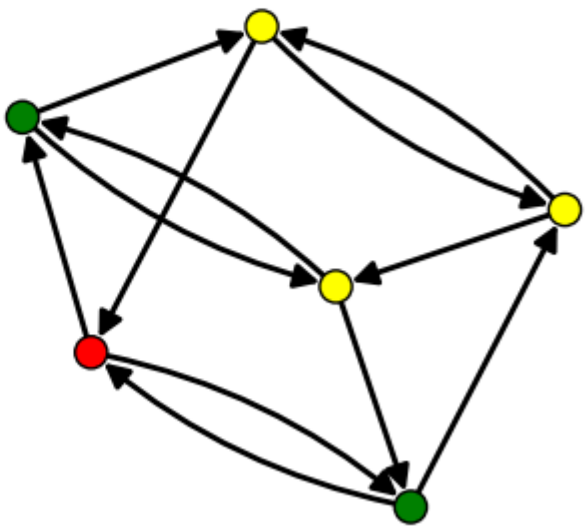


```
G = digraphs.Kautz(2, 2)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s, figsize=4)
```

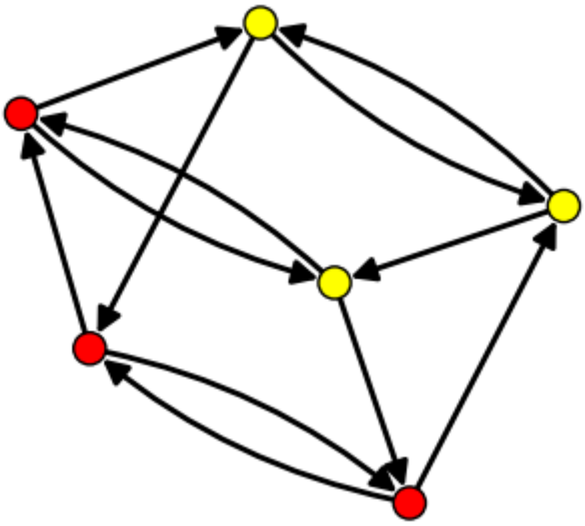
```
Stabilized?
    True
Step 0
```



```
Step 1
```

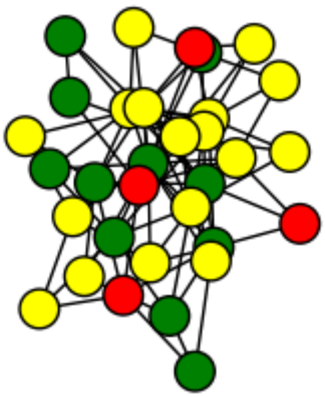


```
Step 2
```

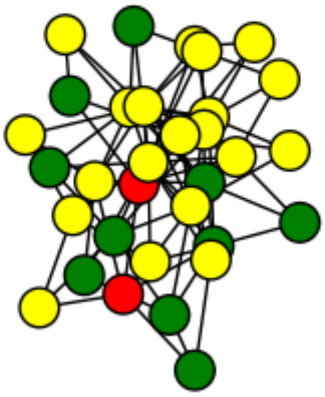


```
# Example: Run the GSL3 rule on a random graph

gg = graphs.RandomBarabasiAlbert
print(gg)
G = gg(32, 3)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s)
```

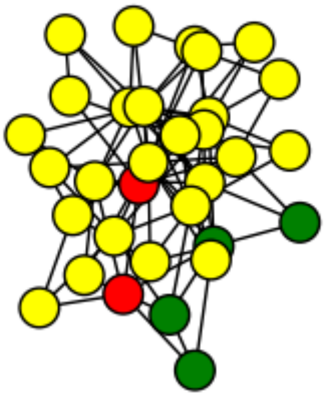
```
<function RandomBarabasiAlbert at 0x10c58e1b8>
Stabilized?
    True
Step 0
```



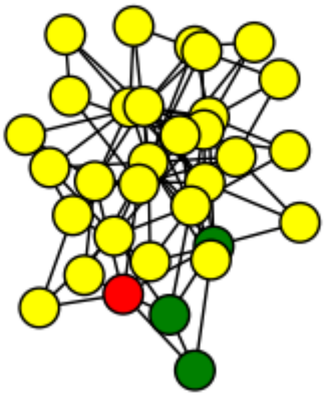
```
Step 1
```



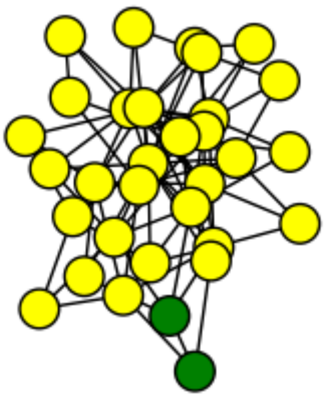
Step 2



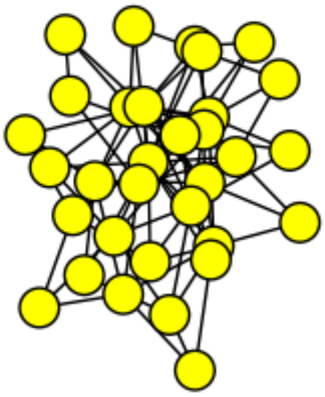
Step 3



Step 4

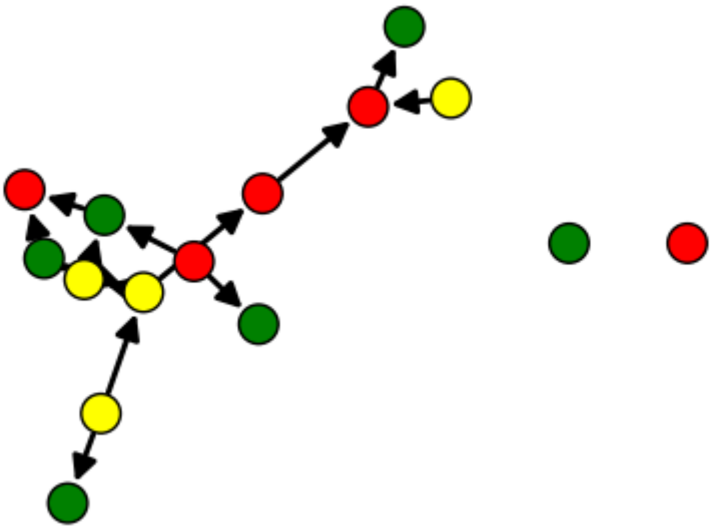


Step 5

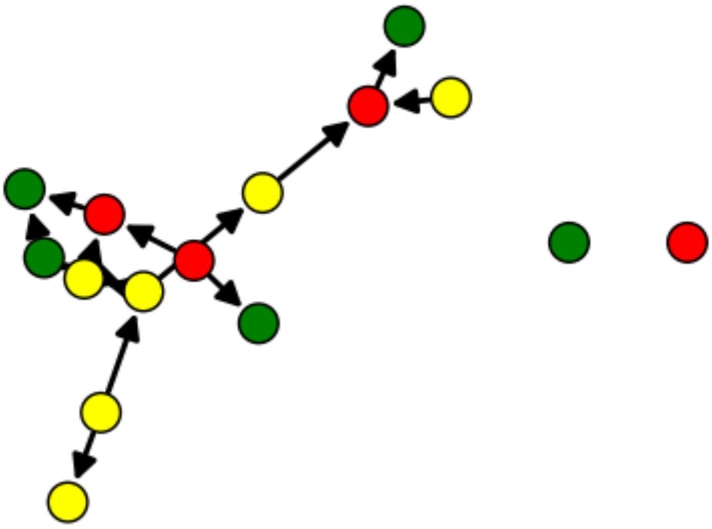


```
# Example: Run the GSL3 rule on a random graph

gg = digraphs.RandomDirectedGNP
print(gg)
G = gg(15, .1)
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
ur_kwargs = {'palette': color_bias.keys(), 'T': 0.6}
initial_coloring = color_randomly(G, color_bias)

s, stabilized = run_rule(ur, ur_kwargs, G, initial_coloring)
print('Stabilized?\n    %s' % stabilized)
show_colorings(G, s, figsize=4)
```

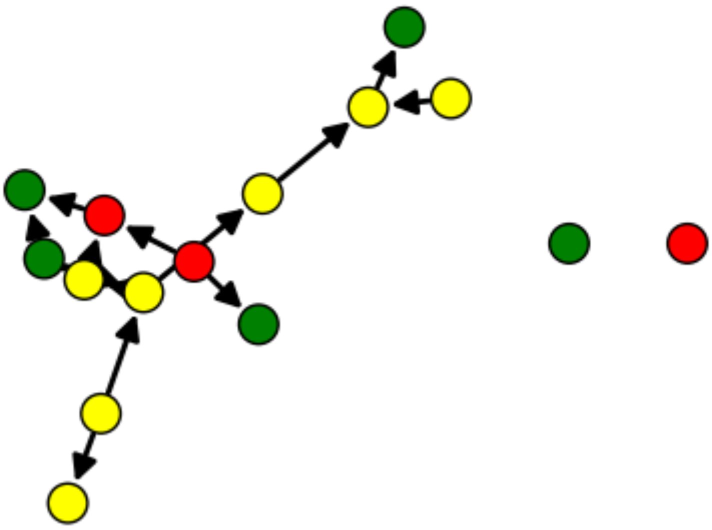
```
<bound method DiGraphGenerators.RandomDirectedGNP of
<sage.graphs.digraph_generators.DiGraphGenerators instance at
0x10b1cd4d0>>
Stabilized?
    True
Step 0
```



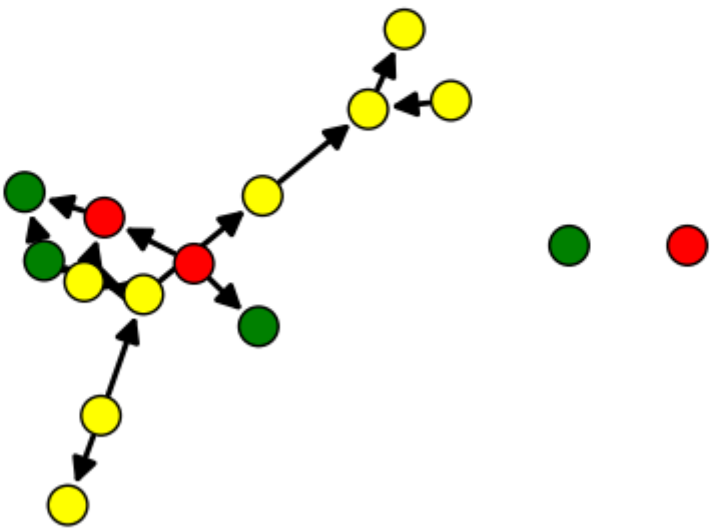
```
Step 1
```



Step 2



Step 3



```
# Example: Use run_rule_many_times() on one graph
```

```
color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys()}
gg = moore_lattice
ggk = {'r': 8, 'c': 16, 'toroidal': True}
cf = color_randomly
cfk = {'bias': color_bias}

num_runs = 1000
num_stabilized, mean_steps, mean_initial, mean_final =
run_rule_many_times(ur, urk, gg, ggk, cf, cfk,
num_runs=num_runs)
```

```
    <function gsl3_rule at 0x10c4c3848>
    <function moore_lattice at 0x10c4c38c0>
    <function color_randomly at 0x10c4c3668>
    --------------------
    Number of runs: 1000
    Number of runs that stabilized: 736
    Mean number of steps required to stabilize: 6.53
    Mean initial color counts:
        green: 42.5
        red: 43
        yellow: 42.6
    Mean final color counts:
        green: 43.7
        red: 39.2
        yellow: 45
```

```
# Example: Use run_rule_many_times() on many instances of a
random graph

color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys(), 'T': 0.5, 't': 0.25, 's':
0.25}
gg = maslov_sneppen
ml = moore_lattice(r=8, c=16, toroidal=True)
ggk = {'graph': ml}  # Warning: Maslov-Sneppon does 4096 (=
4*num_edges) steps on this graph. Slow!
cf = color_randomly
cfk = {'bias': color_bias}

num_runs = 10
num_stabilized, mean_steps, mean_initial, mean_final =
run_rule_many_times(ur, urk, gg, ggk, cf, cfk,
num_runs=num_runs)
```

```
    <function gsl3_rule at 0x10c4c3848>
    <function maslov_sneppen at 0x10c4c39b0>
    <function color_randomly at 0x10c4c3668>
    --------------------
    Number of runs: 10
```

```
    Number of runs that stabilized: 9
    Mean number of steps required to stabilize: 4
    Mean initial color counts:
        green: 46.2
        red: 39.7
        yellow: 42.1
    Mean final color counts:
        green: 50.1
        red: 41.9
        yellow: 36
```

```
# Exploring the random Barabasi Albert graph

G = graphs.RandomBarabasiAlbert(128, 4)
n = G.num_verts()
degrees = G.degree()
ave_degree = sum(degrees)/n

print('nvertices = {!s}'.format(n))
print('degrees = {!s}'.format(degrees))
print('ave degree = {:.3f}'.format(ave_degree))
```

```
    nvertices = 128
    degrees = [5, 7, 23, 34, 46, 26, 21, 20, 27, 29, 11, 14, 12,
    16, 8, 12, 9, 11, 11, 10, 12, 4, 7, 11, 11, 10, 14, 7, 7, 9,
    8, 8, 11, 4, 13, 5, 4, 7, 9, 7, 7, 4, 5, 7, 7, 14, 6, 4, 7, 4
    7, 4, 5, 4, 8, 6, 6, 5, 6, 4, 7, 4, 5, 5, 4, 4, 6, 4, 4, 5, 7
    5, 5, 6, 5, 5, 4, 6, 4, 6, 4, 5, 4, 5, 6, 6, 6, 4, 5, 6, 4, 5
    4, 4, 4, 4, 5, 4, 4, 4, 5, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4
    4, 4, 4]
    ave degree = 7.000
```

```
# Example: Use run_rule_many_times() on many instances of a
random graph

color_bias = {'green': 1/3, 'red': 1/3, 'yellow': 1/3}
ur = gsl3_rule
urk = {'palette': color_bias.keys(), 'T': 0.5, 't': 0.25, 's':
0.25}
gg = graphs.RandomBarabasiAlbert
ggk = {'n': 128, 'm': 4}
cf = color_randomly
cfk = {'bias': color_bias}

num_runs = 1000
num_stabilized, mean_steps, mean_initial, mean_final =
run_rule_many_times(ur, urk, gg, ggk, cf, cfk,
num_runs=num_runs)
```

```
    <function gsl3_rule at 0x10c4c3848>
    <function RandomBarabasiAlbert at 0x10c58e1b8>
    <function color_randomly at 0x10c4c3668>
    --------------------
```

```
Number of runs: 1000
Number of runs that stabilized: 911
Mean number of steps required to stabilize: 4.96
Mean initial color counts:
    green: 42.5
    red: 42.6
    yellow: 42.9
Mean final color counts:
    green: 43.1
    red: 38.7
    yellow: 46.2
```