python-igraph tutorial

- https://igraph.org/python/doc/tutorial/tutorial.html (https://igraph.org/python/doc/tutorial/tutorial.html)
- STATS701: Data Analysis in Python (https://pages.stat.wisc.edu/~kdlevin/teaching/Winter2018/STATS701/) - https://pages.stat.wisc.edu/~kdlevin/teaching/Winter2018/STATS701/ (https://pages.stat.wisc.edu/~kdlevin/teaching/Winter2018/STATS701/)
  - Lecture 27 - Graph processing - https://pages.stat.wisc.edu/~kdlevin/teaching/Winter2018/STATS701/slides/lecture27.pdf (https://pages.stat.wisc.edu/~kdlevin/teaching/Winter2018/STATS701/slides/lecture27.pdf)

1 How to create graph

1.1 Create a simple graph

In [1]:

```
1 import igraph as ig
2 print(ig.__version__)
```

```
0.9.6
```

In [188]:

```python
def default_visual_style(g, visual_style={}):
    if not "vertex_size" in visual_style:
        visual_style["vertex_size"] = 20

    if not "vertex_color" in visual_style:
        visual_style["vertex_color"] = "green"

    if not "bbox" in visual_style:
        visual_style["bbox"] = (300, 300)

    if not "margin" in visual_style:
        visual_style["margin"] = 20

    if not "vertex_label" in visual_style:
        if "name" in g.vs.attributes():
            g.vs["label"] = g.vs["name"]
        elif "id" in g.vs.attributes():
            g.vs["label"] = g.vs["id"]
        elif "label" in g.vs.attributes():
            pass
        else:
            g.vs["label"] = [str(i) for i in range(len(g.vs))]
        visual_style["vertex_label"] = g.vs["label"]
    return visual_style
```

In [29]:

```python
g = ig.Graph()
```

In [30]:

```python
g.add_vertices(3)

g.add_edges([(0,1), (1,2)])

print(g)
```

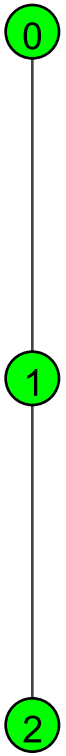
```
IGRAPH U--- 3 2 --
+ edges:
0--1 1--2
```

In [31]:
```
1 len(g.vs), len(g.es), list(g.vs), list(g.es)
```

Out[31]:
```
(3,
 2,
 [igraph.Vertex(<igraph.Graph object at 0x000001F3F760BB88>, 0, {}),
  igraph.Vertex(<igraph.Graph object at 0x000001F3F760BB88>, 1, {}),
  igraph.Vertex(<igraph.Graph object at 0x000001F3F760BB88>, 2, {})],
 [igraph.Edge(<igraph.Graph object at 0x000001F3F760BB88>, 0, {}),
  igraph.Edge(<igraph.Graph object at 0x000001F3F760BB88>, 1, {})])
```

In [32]:
```
1 ig.plot(g, layout=g.layout("tree", root=[0]), **default_visual_style(g))
```

Out[32]:



In [33]:
```
1 g.add_vertices(3)
2 g.add_edges([(2, 3), (3, 4), (4, 5), (5, 3)])
```

In [34]:
```
1 print(g)
```

```
IGRAPH U--- 6 6 --
+ attr: label (v)
+ edges:
0--1 1--2 2--3 3--4 4--5 3--5
```

In [35]:
```
1 g.summary()   # ignore edges
```

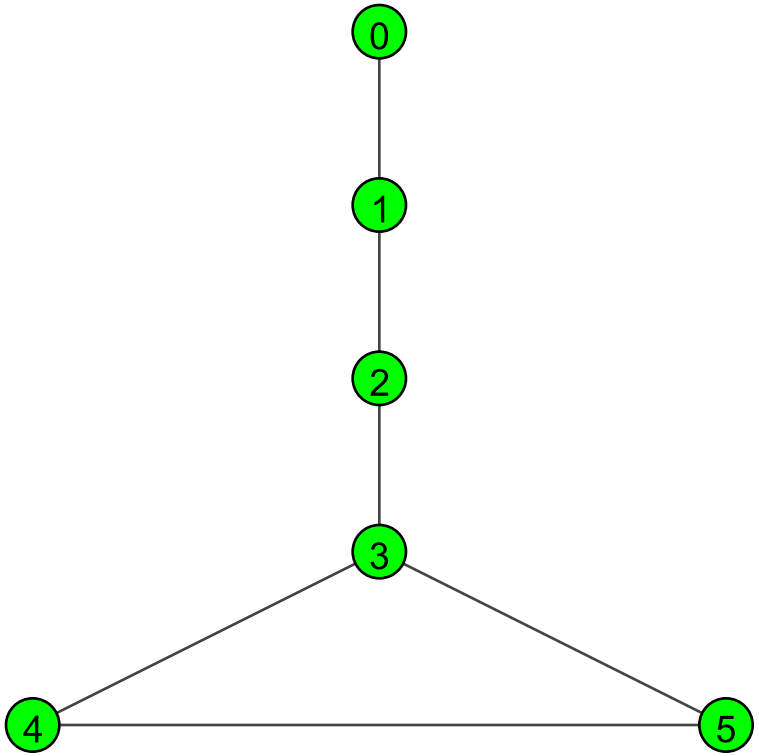
Out[35]: 'IGRAPH U--- 6 6 -- \n+ attr: label (v)'

In [36]:
```
1 g.get_eid(2, 3), g.get_eid(3, 5)
```

Out[36]: (2, 5)

In [37]:
```
1 ig.plot(g, layout=g.layout("tree", root=[0]), **default_visual_style(g))
```

Out[37]:



1.2 Create a Tree graph

In [38]:

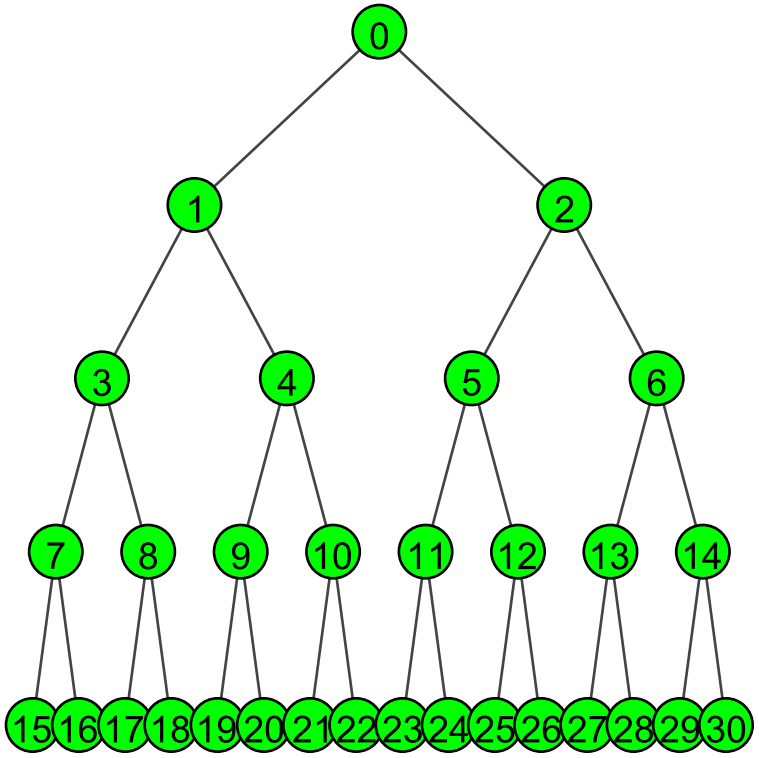
```
import math

# complete binary tree
depth = 5
breadth = 2
n_vs = 0
for d in range(depth):
    n_vs += math.pow(breadth, d)
n_vs = int(n_vs)
print(f"No. of nodes: {n_vs}")
```

```
No. of nodes: 31
```

In [39]:

```
gt = ig.Graph.Tree(n_vs,breadth)
ig.plot(gt, layout=gt.layout("tree", root=[0]), **default_visual_style(gt))
```

Out[39]:



In [40]:
```
1 print(gt)
```

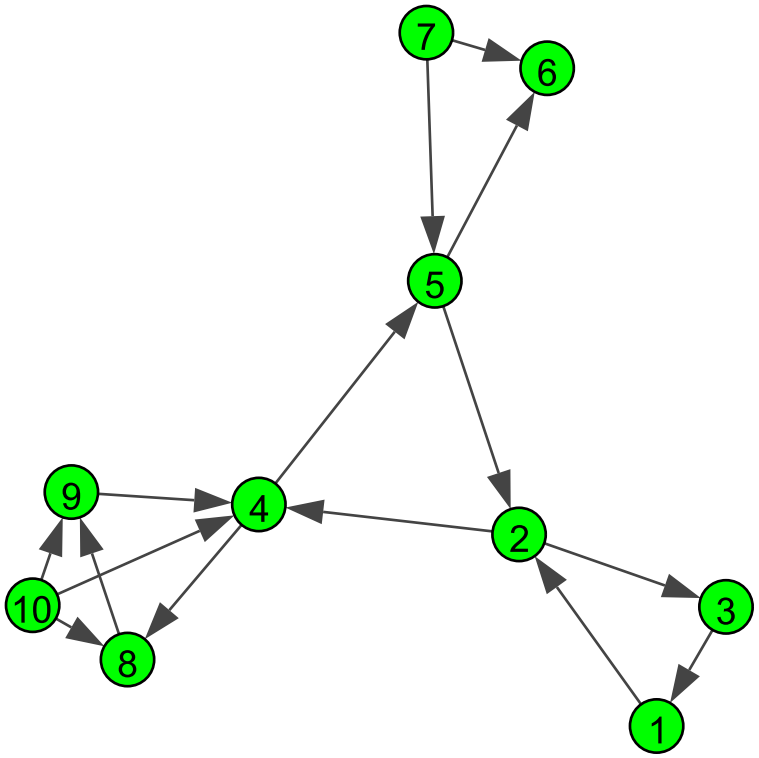
```
IGRAPH U--- 31 30 --
+ attr: label (v)
+ edges:
0--1 0--2 1--3 1--4 2--5 2--6 3--7 3--8 4--9 4--10 5--11 5--12 6--13 6--14
7--15 7--16 8--17 8--18 9--19 9--20 10--21 10--22 11--23 11--24 12--25 12--26
13--27 13--28 14--29 14--30
```

1.3 create graph from text file

In [42]:
```
1  small_net = """
2  1 2
3  2 3
4  2 7
5  3 1
6  4 2
7  4 6
8  5 4
9  5 6
10 7 4
11 7 8
12 8 9
13 9 7
14 10 7
15 10 8
16 10 9
17 """
18
19 with open("net.txt", "w") as f:
20     f.write(small_net)
```

In [45]:
```
g1 = ig.Graph.Read_Ncol("net.txt")
g1.vs["label"] = [str(i+1) for i in range(len(g1.vs))]
ig.plot(g1, **default_visual_style(g1))
```

Out[45]:



1.4 generates a geometric random graph

n points are chosen randomly and uniformly inside the unit square and pairs of points closer to each other than a predefined distance d are connected by an edge

In [153]:
```
gr = ig.Graph.GRG(50, 0.2)
```

In [154]:
```
gr.summary()
```

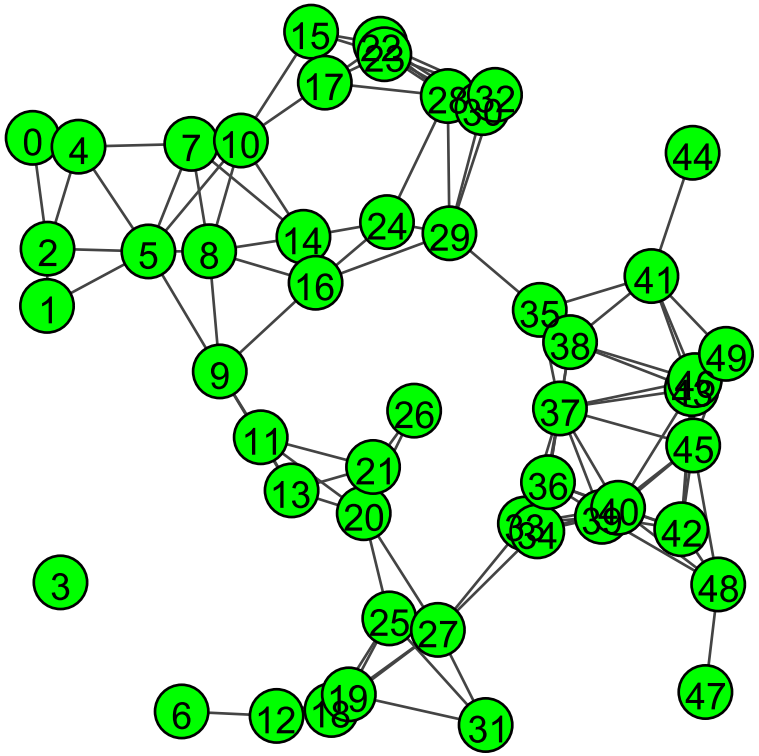
Out[154]:
```
'IGRAPH U--- 50 125 -- \n+ attr: x (v), y (v)'
```

In [155]:
```
1 print(gr)
```

```
IGRAPH U--- 50 125 --
+ attr: x (v), y (v)
+ edges:
 0 --  2  4                        24 -- 14 16 28 29                    48
-- 39 40 42 45 47
 1 --  2  5                        25 -- 18 19 20 27 31                 49
-- 41 43 45 46
 2 --  0  1  4  5                  26 -- 20 21
 3 --                              27 -- 18 19 20 25 31 33 34
 4 --  0  2  5  7                  28 -- 17 22 23 24 29 30 32
 5 --  1  2  4  7  8  9 10         29 -- 16 24 28 30 32 35
 6 -- 12                           30 -- 22 23 28 29 32
 7 --  4  5  8 10 14               31 -- 19 25 27
 8 --  5  7  9 10 14 16            32 -- 22 23 28 29 30
 9 --  5  8 11 13 16               33 -- 27 34 36 37 39 40
10 --  5  7  8 14 15 17            34 -- 27 33 36 37 39 40
11 --  9 13 20 21                  35 -- 29 37 38 41
12 --  6 18 19                     36 -- 33 34 37 38 39 40 42
13 --  9 11 20 21                  37 -- 33 34 35 36 38 39 40 43 45 46
14 --  7  8 10 16 24               38 -- 35 36 37 41 43 46
15 -- 10 17 22 23                  39 -- 33 34 36 37 40 42 45 48
16 --  8  9 14 24 29               40 -- 33 34 36 37 39 42 43 45 48
17 -- 10 15 22 23 28               41 -- 35 38 43 44 46 49
18 -- 12 19 25 27                  42 -- 36 39 40 43 45 46 48
19 -- 12 18 25 27 31               43 -- 37 38 40 41 42 45 46 49
20 -- 11 13 21 25 26 27            44 -- 41
21 -- 11 13 20 26                  45 -- 37 39 40 42 43 46 48 49
22 -- 15 17 23 28 30 32            46 -- 37 38 41 42 43 45 49
23 -- 15 17 22 28 30 32            47 -- 48
```

In [158]:
```
1 ig.plot(gr, **default_visual_style(gr, {}))
```

Out[158]:

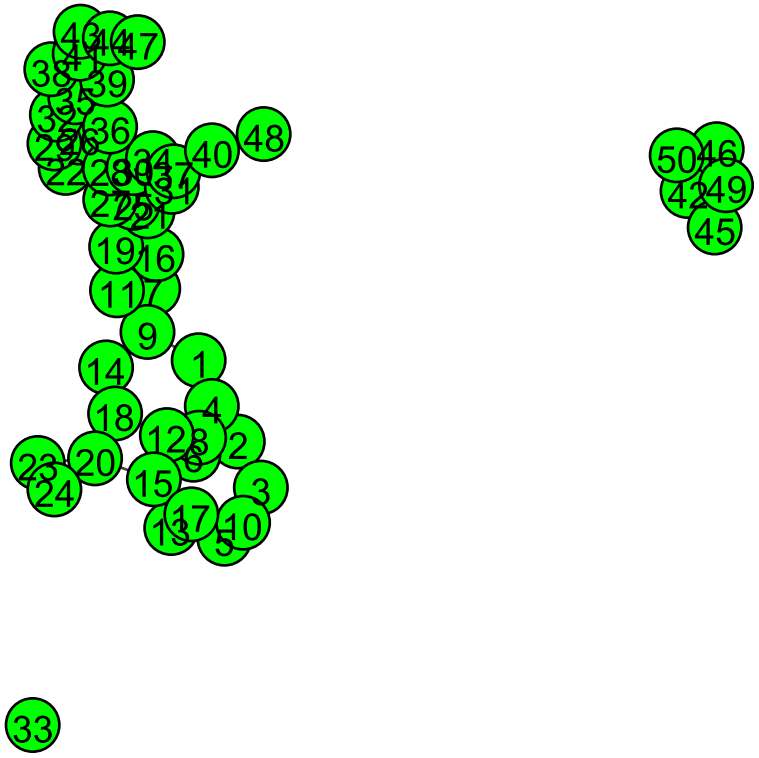


1.4.1 Various layout

In [127]:

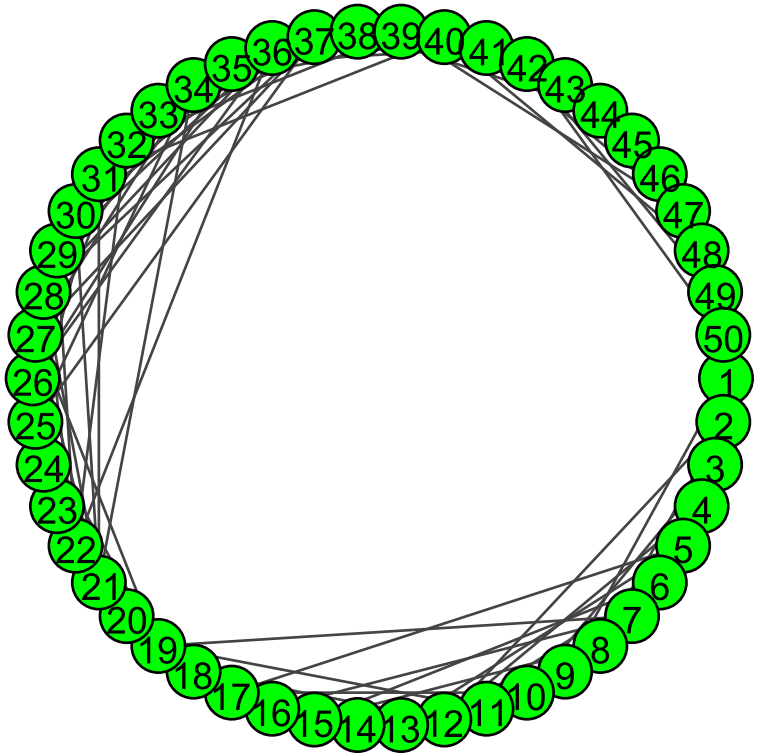
```
# Kamada-Kawai force-directed algorithm
ig.plot(gr, layout=gr.layout("kk"), **default_visual_style(gr, {}))
```

Out[127]:



In [128]:
```
# Deterministic layout that places the vertices on a circle
ig.plot(gr, layout=gr.layout("circle"), **default_visual_style(gr, {}))
```

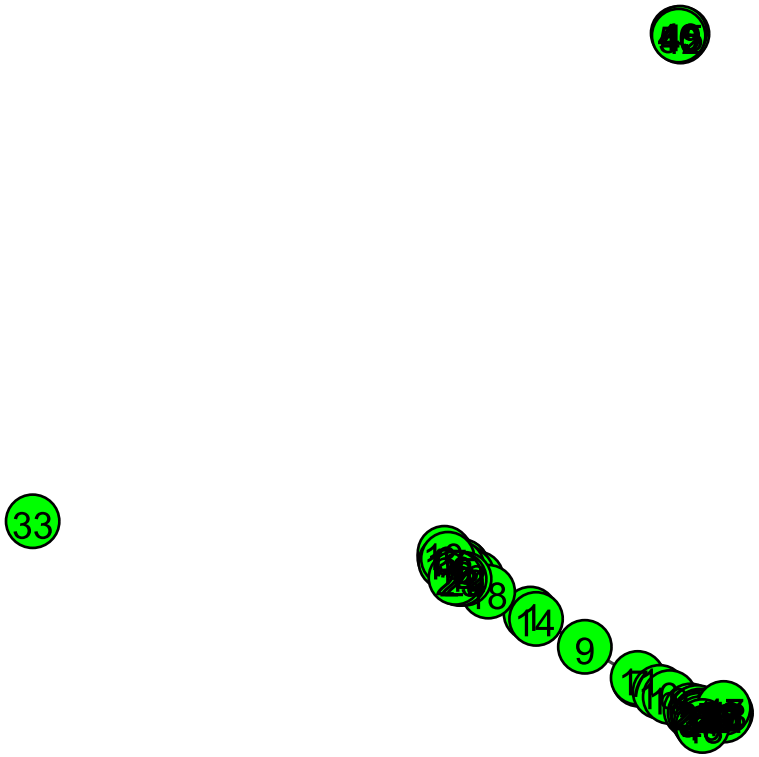
Out[128]:



In [129]:

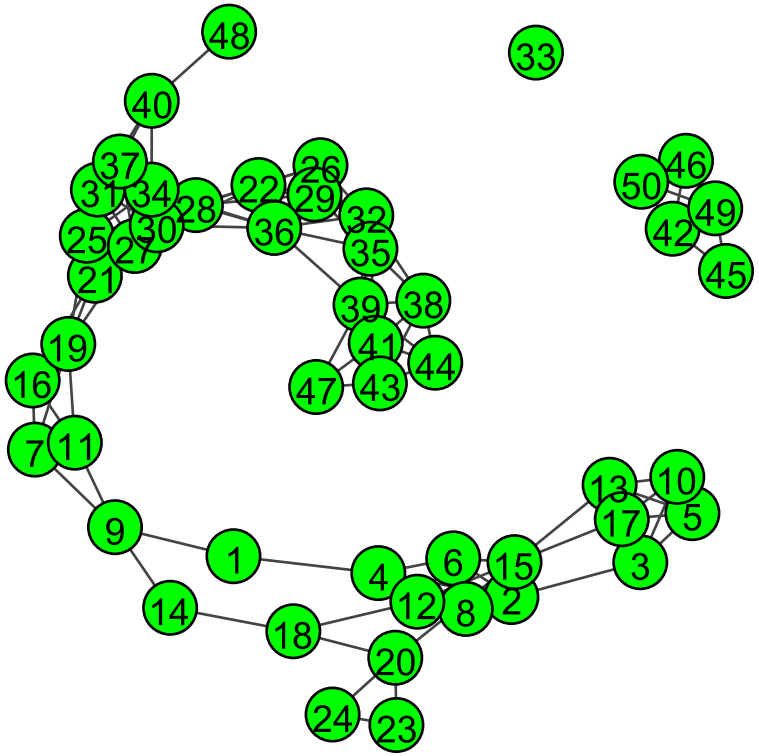
```
# The Distributed Recursive Layout algorithm for large graphs
ig.plot(gr, layout=gr.layout("drl"), **default_visual_style(gr, {}))
```

Out[129]:



In [130]:
```
# Fruchterman-Reingold force-directed algorithm
ig.plot(gr, layout=gr.layout("fr"), **default_visual_style(gr, {}))
```

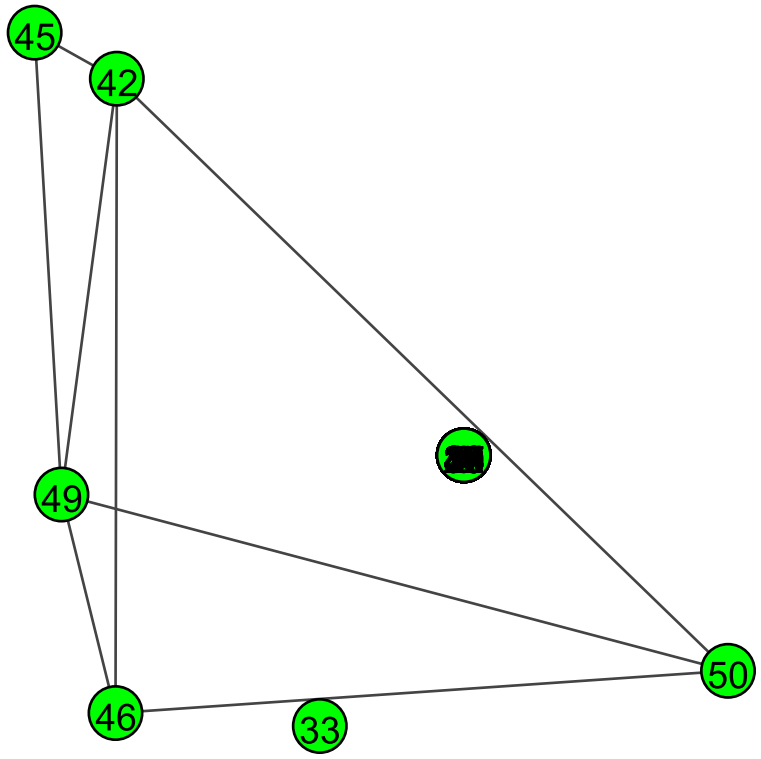
Out[130]:



In [131]:

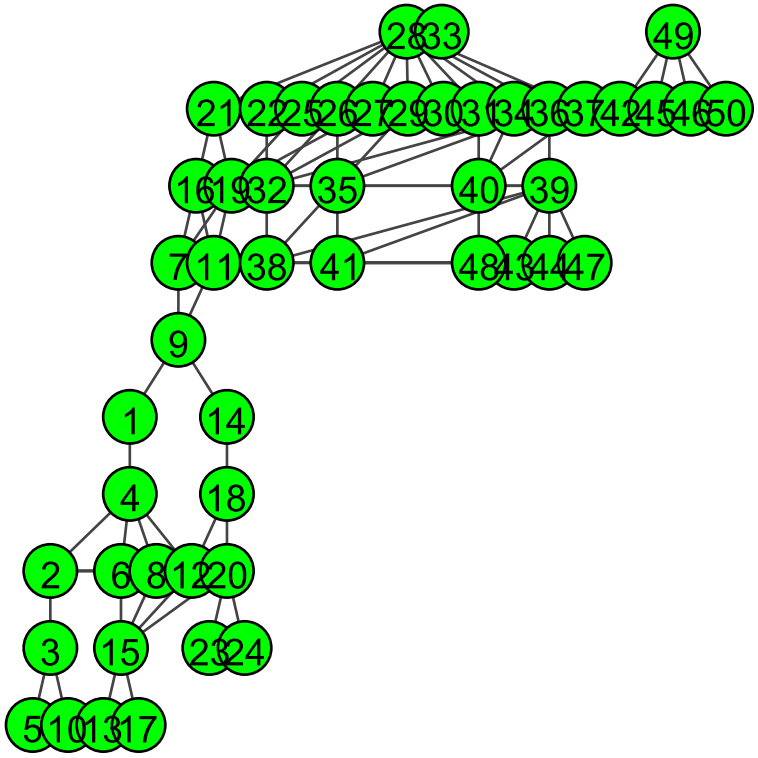
```
# The Large Graph Layout algorithm for large graphs
ig.plot(gr, layout=gr.layout("large"), **default_visual_style(gr, {}))
```

Out[131]:



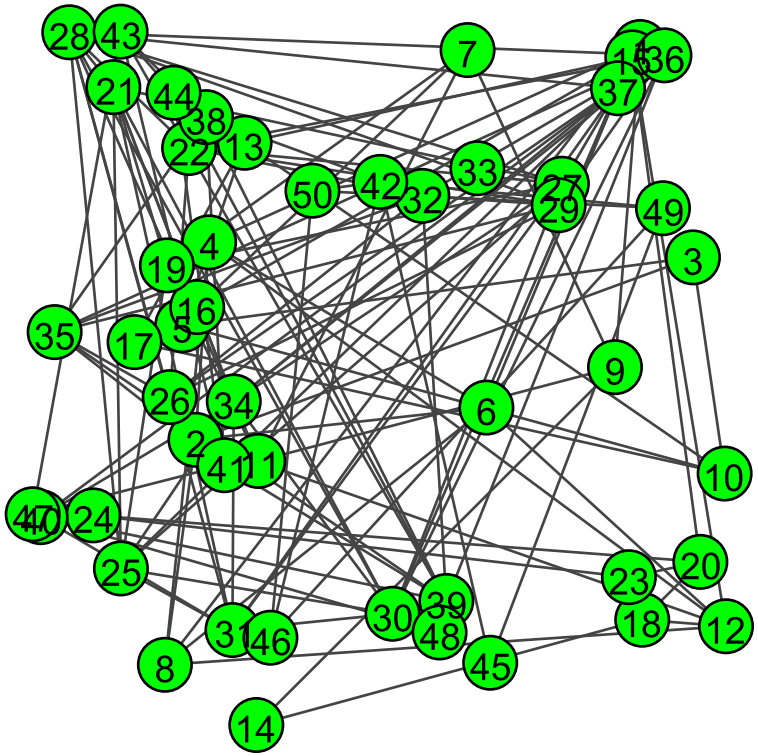
In [132]:
```
# Reingold-Tilford tree layout, useful for (almost) tree-like graphs
ig.plot(gr, layout=gr.layout("tree"), **default_visual_style(gr, {}))
```

Out[132]:



In [133]:
```
# Places the vertices completely randomly
ig.plot(gr, layout=gr.layout("random"), **default_visual_style(gr, {}))
```

Out[133]:



2 Setting and retrieving attributes

igraph uses vertex and edge IDs in its core. These IDs are integers, starting from zero, and they are always continuous at any given time instance during the lifetime of the graph. This means that whenever vertices and edges are deleted, a large set of edge and possibly vertex IDs will be renumbered to ensure the continuity. Now, let us assume that our graph is a social network where vertices represent people and edges represent social connections between them. One way to maintain the association between vertex IDs and say, the corresponding names is to have an additional Python list that maps from vertex IDs to names. The drawback of this approach is that this additional list must be maintained in parallel to the modifications of the original graph.

Luckily, igraph knows the concept of attributes, i.e., auxiliary objects associated to a given vertex or edge of a graph, or even to the graph as a whole.

Every igraph Graph, vertex and edge behaves as a standard Python dictionary in some sense: you can add key-value pairs to any of them, with the key representing the name of your attribute (the only restriction is that it must be a string) and the value representing the attribute itself.

In [51]:
```
g3 = ig.Graph([(0,1), (0,2), (2,3), (3,4), (4,2), (2,5), (5,0), (6,3), (5,6)])
```

In [52]:
```
print(g3)
```

```
IGRAPH U--- 7 9 --
+ edges:
0 -- 1 2 5     2 -- 0 3 4 5   4 -- 2 3       6 -- 3 5
1 -- 0         3 -- 2 4 6     5 -- 0 2 6
```

Every Graph object contains two special members called **vs** (vertex sequence) and **es** (edge sequence)

In [73]:
```
g3.vs["name"] = ["Alice", "Bob", "Claire", "Dennis", "Esther", "Frank", "George"]
g3.vs["age"] = [25, 31, 18, 47, 22, 23, 50]
g3.vs["gender"] = ["f", "m", "f", "m", "f", "m", "m"]
g3.es["is_formal"] = [False, False, True, True, True, False, True, False, False]
```

In [74]:
```
print(g3)
```

```
IGRAPH UN-- 7 9 --
+ attr: age (v), gender (v), label (v), name (v), is_formal (e)
+ edges (vertex names):
 Alice -- Bob, Claire, Frank             Esther -- Claire, Dennis
   Bob -- Alice                           Frank -- Alice, Claire, George
Claire -- Alice, Dennis, Esther, Frank   George -- Dennis, Frank
Dennis -- Claire, Esther, George
```

In [75]:
```
g3.summary()
```

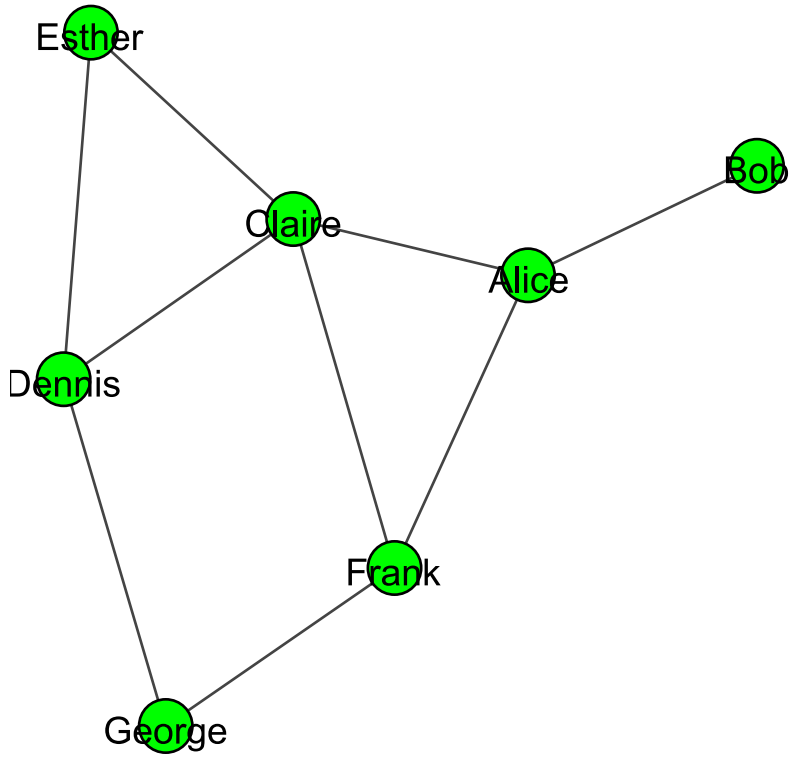
Out[75]:
```
'IGRAPH UN-- 7 9 -- \n+ attr: age (v), gender (v), label (v), name (v), is_formal (e)'
```

In [76]:
```
g3.vs["name"]
```

Out[76]:
```
['Alice', 'Bob', 'Claire', 'Dennis', 'Esther', 'Frank', 'George']
```

In [79]:
```
1 ig.plot(g3, **default_visual_style(g3))
```

Out[79]:



In [56]:
```
1 g3.es[0].attributes()
```

Out[56]: {'is_formal': False}

In [57]:
```
1 g3.es[0].attributes()["is_formal"]
```

Out[57]: False

In [58]:
```
1 g3.vs[3].attributes()
```

Out[58]: {'name': 'Dennis', 'age': 47, 'gender': 'm'}

In [59]:
```
1 g3.vs[3]
```

Out[59]:
```
igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 3, {'name': 'Dennis', 'age': 47, 'gender': 'm'})
```

In [60]:
```
1 list(g3.vs)
```

Out[60]:
```
[igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 0, {'name': 'Alice', 'age': 25, 'gender':
'f'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 1, {'name': 'Bob', 'age': 31, 'gender': 'm'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 2, {'name': 'Claire', 'age': 18, 'gender':
'f'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 3, {'name': 'Dennis', 'age': 47, 'gender':
'm'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 4, {'name': 'Esther', 'age': 22, 'gender':
'f'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 5, {'name': 'Frank', 'age': 23, 'gender':
'm'}),
 igraph.Vertex(<igraph.Graph object at 0x000001F3F7853408>, 6, {'name': 'George', 'age': 50, 'gender':
'm'})]
```

In [61]:
```
1 list(g3.es)
```

Out[61]:
```
[igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 0, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 1, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 2, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 3, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 4, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 5, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 6, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 7, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 8, {'is_formal': False})]
```

2.1 graph properties

In [128]:
```
1 g3.degree()
```

Out[128]:
```
[3, 1, 4, 3, 2, 3, 2]
```

In [129]:
```
1 g3.degree(6)   # degree for vortexID=6
```

Out[129]: 2

In [130]:
```
1 g3.degree([2,3])
```

Out[130]: [4, 3]

In [131]:
```
1 g3.vs.select(_degree=g3.maxdegree())["name"]
```

Out[131]: ['Claire']

In [83]:
```
1 list(g3.es.select(_source=2))
```

Out[83]:
```
[igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 1, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 2, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 4, {'is_formal': True}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 5, {'is_formal': False})]
```

In [82]:
```
1 list(g3.es.select(_target=0))
```

Out[82]:
```
[igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 0, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 1, {'is_formal': False}),
 igraph.Edge(<igraph.Graph object at 0x000001F3F7853408>, 6, {'is_formal': True})]
```

In [133]:
```
1 for e in g3.es.select(_within=g3.vs[2:5]):
2     print(e)
```

```
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 2, {'is_formal': True})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 3, {'is_formal': True})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 4, {'is_formal': True})
```

In [134]:
```
for v in g3.vs.select(gender="m"):
    print(v)
```

```
igraph.Vertex(<igraph.Graph object at 0x00000144B4EDF318>, 1, {'name': 'Bob', 'age': 31, 'gender': 'm'})
igraph.Vertex(<igraph.Graph object at 0x00000144B4EDF318>, 3, {'name': 'Dennis', 'age': 47, 'gender': 'm'})
igraph.Vertex(<igraph.Graph object at 0x00000144B4EDF318>, 5, {'name': 'Frank', 'age': 23, 'gender': 'm'})
igraph.Vertex(<igraph.Graph object at 0x00000144B4EDF318>, 6, {'name': 'George', 'age': 50, 'gender': 'm'})
```

In [135]:
```
men = g3.vs.select(gender="m")
women = g3.vs.select(gender="f")
for e in g3.es.select(_between=(men, women)):
    print(e)
```

```
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 0, {'is_formal': False})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 2, {'is_formal': True})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 3, {'is_formal': True})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 5, {'is_formal': False})
igraph.Edge(<igraph.Graph object at 0x00000144B4EDF318>, 6, {'is_formal': True})
```

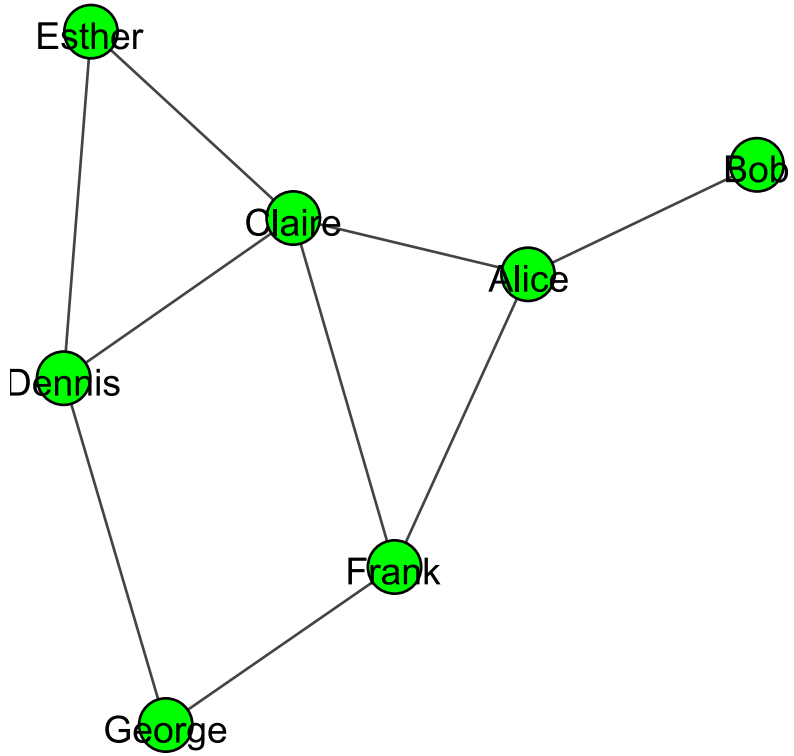
In [136]:
```
g3.get_adjacency()
```

Out[136]:
```
Matrix([[0, 1, 1, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 0], [1, 0, 0, 1, 1, 1, 0], [0, 0, 1, 0, 1, 0, 1], [0, 0,
1, 1, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1], [0, 0, 0, 1, 0, 1, 0]])
```

2.1.1 Plot layout

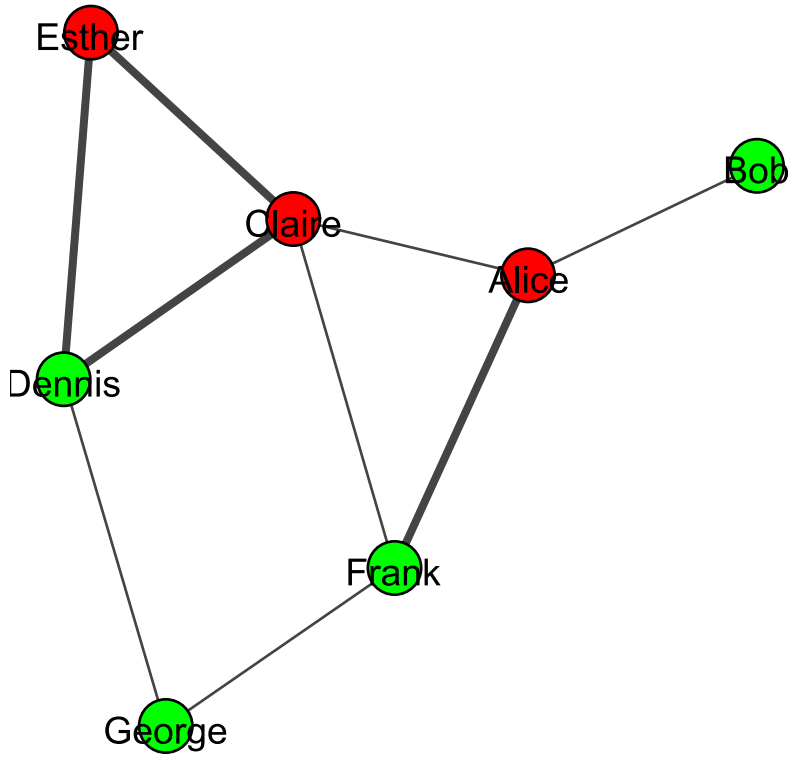
In [93]:
```
1 ig.plot(g3, layout=g3.layout("kk"), **default_visual_style(g3))
```

Out[93]:



In [95]:
```
# customize vertex_color, edge_width

# g3.vs["label"] = g3.vs["name"]
color_dict = {"m": "green", "f": "red"}
visual_style["edge_width"] = [1 + 2 * int(is_formal) for is_formal in g3.es["is_formal"]]
g3.vs["color"] = [color_dict[gender] for gender in g3.vs["gender"]]
visual_style["vertex_color"] = g3.vs["color"]
ig.plot(g3, layout=g3.layout("kk"), **default_visual_style(g3, visual_style))
```

Out[95]:



In [ ]:
```

```

2.2 persist graph

In [96]:
```
1 g3.save("soc_net.gml", format="gml")
```

```
c:\work\github\nlp\textnets\lib\site-packages\igraph\__init__.py:2984: RuntimeWarning: A boolean edge attri
bute was converted to numeric at src/io/gml.c:763
  return writer(f, *args, **kwds)
```

In [97]:
```
1 !dir soc_net.gml
```

```
 Volume in drive C is CDRIVE
 Volume Serial Number is F0EF-6714

 Directory of C:\work\github\NLP\textnets-tutorial

09/28/2021  10:12 AM             1,312 soc_net.gml
               1 File(s)          1,312 bytes
               0 Dir(s)  82,459,770,880 bytes free
```

In [98]:
```
1 !type soc_net.gml
```

```
    source 2
    target 0
    isformal 0
  ]
  edge
  [
    source 3
    target 2
    isformal 1
  ]
  edge
  [
    source 4
    target 3
    isformal 1
  ]
  edge
  [
    source 4
    target 2
```

In [99]:
```
1 g3_2 = ig.load("soc_net.gml")
```

In [100]:
```
1 print(g3_2)
```

```
IGRAPH UN-- 7 9 --
+ attr: age (v), color (v), gender (v), id (v), label (v), name (v), isformal
  (e)
+ edges (vertex names):
 Alice -- Bob, Claire, Frank             Esther -- Claire, Dennis
   Bob -- Alice                           Frank -- Alice, Claire, George
Claire -- Alice, Dennis, Esther, Frank   George -- Dennis, Frank
Dennis -- Claire, Esther, George
```

In [101]:
```
1 print(g3)
```

```
IGRAPH UN-- 7 9 --
+ attr: age (v), color (v), gender (v), label (v), name (v), is_formal (e)
+ edges (vertex names):
 Alice -- Bob, Claire, Frank             Esther -- Claire, Dennis
   Bob -- Alice                           Frank -- Alice, Claire, George
Claire -- Alice, Dennis, Esther, Frank   George -- Dennis, Frank
Dennis -- Claire, Esther, George
```

In [102]:
```
1 g3_2.isomorphic(g3)
```

Out[102]: True

2.3 Integration

In [152]:
```
1 karate = ig.Graph.Read_GraphML("zachary.graphml")
```

In [153]:
```
1 karate.summary()
```

Out[153]: 'IGRAPH U--- 34 78 -- \n+ attr: id (v)'

In [154]:
```
layout = karate.layout("kk")
visual_style = {}
visual_style["vertex_label"] = karate.vs["id"]
ig.plot(karate, layout=layout, **visual_style)
```

Out[154]: <igraph.drawing.Plot at 0x144b49d85f8>

In [175]:
```
layout = karate.layout("fr")
visual_style = {}
visual_style["vertex_label"] = karate.vs["id"]
ig.plot(karate, layout=layout, **visual_style)
```

Out[175]: <igraph.drawing.Plot at 0x144b4e34320>

In [176]:
```
layout = karate.layout("tree")
visual_style = {}
visual_style["vertex_label"] = karate.vs["id"]
ig.plot(karate, layout=layout, **visual_style)
```

Out[176]: <igraph.drawing.Plot at 0x144b47198d0>

2.4 Measures, coefficients, transformations

In [160]:
```
karate.density()
```

Out[160]: 0.13903743315508021

In [161]:
```
karate.diameter()
```

Out[161]: 5

In [162]:
```
d = karate.indegree()
```

In [164]:
```
print(d)
```

```
[16, 9, 10, 6, 3, 4, 4, 4, 5, 2, 3, 1, 2, 5, 2, 2, 2, 2, 2, 3, 2, 2, 2, 5, 3, 3, 2, 4, 3, 4, 4, 6, 12, 17]
```

In [165]:
```
b = karate.betweenness()
```

In [166]:
```
print(b)
```

```
[231.0714285714286, 28.478571428571424, 75.85079365079366, 6.288095238095237, 0.3333333333333333, 15.833333333333334, 15.833333333333332, 0.0, 29.529365079365082, 0.44761904761904764, 0.3333333333333333, 0.0, 0.0, 24.215873015873015, 0.0, 0.0, 0.0, 0.0, 0.0, 17.1468253968254, 0.0, 0.0, 0.0, 9.299999999999999, 1.1666666666666665, 2.0277777777777777, 0.0, 11.79206349206349, 0.9476190476190476, 1.5428571428571427, 7.609523809523381, 73.00952380952378, 76.69047619047619, 160.5515873015873]
```

In [167]:
```
len(karate.vs), len(d), len(b)
```

Out[167]: (34, 34, 34)

2.4.1 sort nodes by betweenness

In [174]:
```
karate.vs['between']=b
karate.vs["name"] = karate.vs["id"]

m=sorted(karate.vs, key=lambda z: z['between'], reverse=True)
for e in m[:10]:
    print(f"name: {e['name']}, betweenness: {e['between']}")
```

```
name: n0, betweenness: 231.0714285714286
name: n33, betweenness: 160.5515873015873
name: n32, betweenness: 76.69047619047619
name: n2, betweenness: 75.85079365079366
name: n31, betweenness: 73.00952380952378
name: n8, betweenness: 29.529365079365082
name: n1, betweenness: 28.478571428571424
name: n13, betweenness: 24.215873015873015
name: n19, betweenness: 17.1468253968254
name: n5, betweenness: 15.833333333333334
```

2.5 community detection

In [177]:
```
C=karate.community_infomap()
```

In [178]:
```
print(C)
```

```
Clustering with 34 elements and 3 clusters
[0] n0, n1, n2, n3, n7, n9, n11, n12, n13, n17, n19, n21
[1] n4, n5, n6, n10, n16
[2] n8, n14, n15, n18, n20, n22, n23, n24, n25, n26, n27, n28, n29, n30, n31,
    n32, n33
```

In [184]:
```
type(C[1]), C[1]
```

Out[184]:
```
(list, [4, 5, 6, 10, 16])
```

```
for v in karate.vs:
    n = v["id"].replace("n","")
    if int(n) in C[1]:
        print("In C1")
    else:
        print("out")

```

In [192]:
```
ID_COMM = 2
karate.vs["label"] = karate.vs["id"]
comm_color = []
for v in karate.vs:
    if int(v["id"].replace("n","")) in C[ID_COMM]:
        comm_color.append("green")
    else:
        comm_color.append("red")

visual_style["vertex_color"] = comm_color
ig.plot(karate, layout=karate.layout("kk"), **visual_style)
```

Out[192]:
```
<igraph.drawing.Plot at 0x144b4e34518>
```

2.5.1 detect community in a random graph

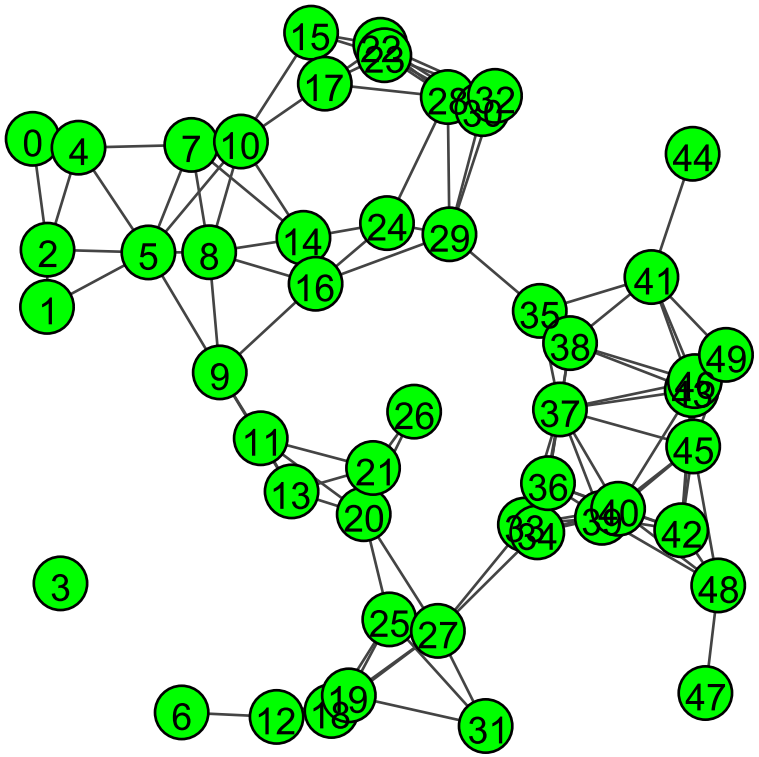
In [159]:
```
C2=gr.community_infomap()
```

In [160]:
```
for n, g in enumerate(C2):
    print(f"n={n}, g={g}")
```

```
n=0, g=[0, 1, 2, 4, 5, 7, 8, 9, 10, 14, 16]
n=1, g=[3]
n=2, g=[6, 12, 18, 19, 25, 27, 31]
n=3, g=[11, 13, 20, 21, 26]
n=4, g=[15, 17, 22, 23, 24, 28, 29, 30, 32]
n=5, g=[33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]
```

In [161]:
```
ig.plot(gr, **default_visual_style(gr, {}))
```

Out[161]:



In [179]:
```
1 g_c0 = C2.subgraph(0)
```

In [180]:
```
1 print(g_c0)
```

```
IGRAPH U--- 11 22 --
+ attr: label (v), x (v), y (v)
+ edges:
 0 --  2  3                4 --  1  2  3  5  6  7  8    8 --  4  5  6  9
 1 --  2  4                5 --  3  4  6  8  9          9 --  5  6  8 10
 2 --  0  1  3  4           6 --  4  5  7  8  9 10      10 --  6  7  9
 3 --  0  2  4  5           7 --  4  6 10
```

In [184]:
```
1 g_c0.vs["label"]
```

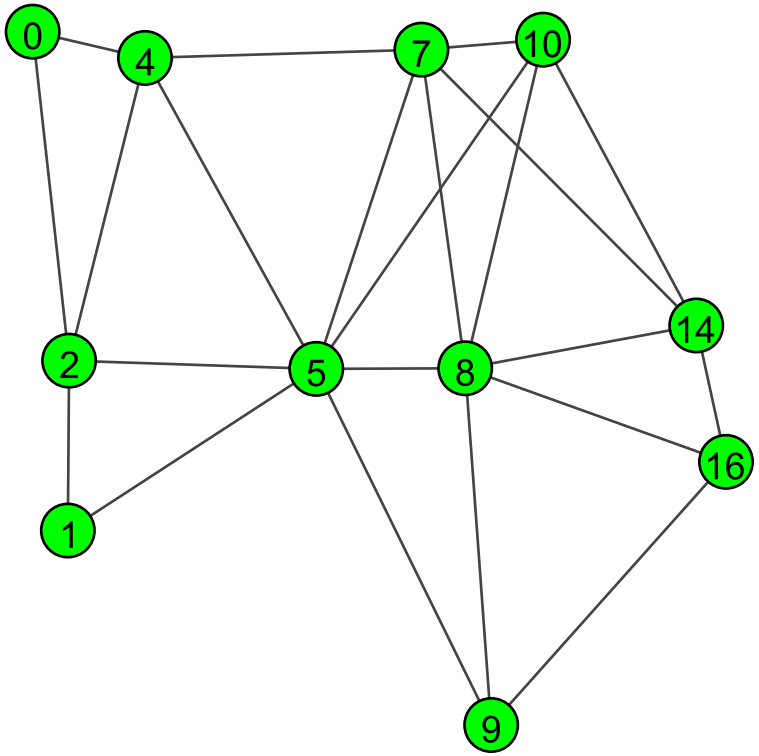
Out[184]: ['0', '1', '2', '4', '5', '7', '8', '9', '10', '14', '16']

In [182]:
```
1 g_c0.vs.attributes()
```

Out[182]: ['x', 'y', 'label']

In [190]:
```
1 ig.plot(g_c0, **default_visual_style(g_c0))
```

Out[190]:



In [194]:
```
1 C0_sub=g_c0.community_infomap()
2
3 if len(C0_sub) > 1:
4     for n, g in enumerate(C0_sub):
5         print(f"n={n}, g={g}")
```

2.6 Big Data graph

Carlos G. Figuerola: A walk on Python-igraph

Big Data ...

- we have data in CSV format, which is also known as *ncol* in the graph world

```
ABELARDO_COLLAZO_ARAUJO JUAN_MARTIN_LUNA    6
ABELARDO_COLLAZO    FERNANDO_HIERRO_CHOMON  7
ABEL_CABALLERO  ALFONSO_GUERRA  30
ABEL_CABALLERO  CARLOS_SOLCHAGA 14
ABEL_CABALLERO  ERNEST_LLUCH    8
ABEL_CABALLERO  FELIPE_GONZALEZ 33
```

- loading 0.42M vertices and 1.34M edges takes only 2 seconds in my notebook (intel *i3*, 4G RAM)

In [ ]:
```
1
```

In [ ]:
```
1
```