Lab4_Notebook

October 7, 2019

0.1 Binary Tree

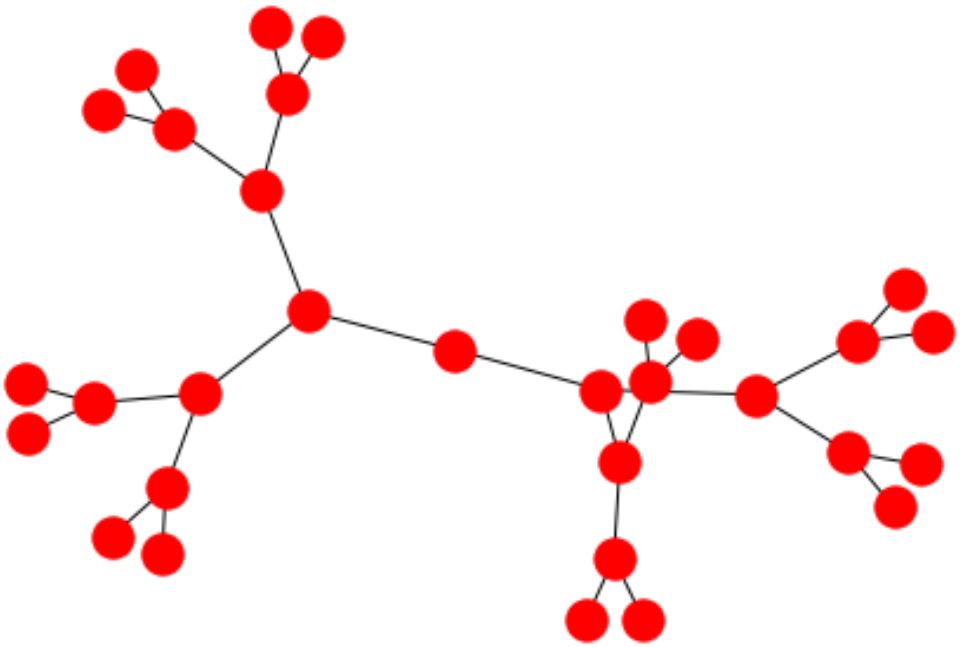
0.1.1 Recursive method

In [1]:
```
import networkx as nx
import time
import collections
import matplotlib.pyplot as plt
```

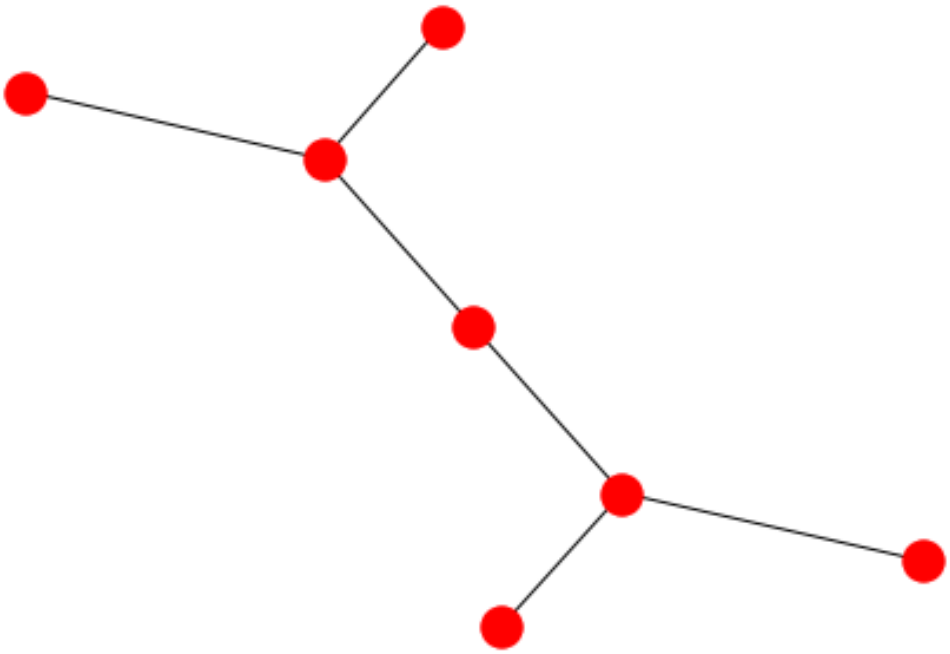
In [5]:
```
def add_balanced_tree(height, g): # returns the root of the subtree
    root = g.number_of_nodes()
    g.add_node(root)# add root node
    if height == 0:
        return root
    else:
        # create two subtrees of smaller height
        # create two subtrees of smaller height
        g.add_edge(root, add_balanced_tree(height-1, g)) # connect root node of subtrees

        g.add_edge(root, add_balanced_tree(height-1, g)) # connect root node of subtrees
        return root
```

In [6]:
```
g=nx.balanced_tree(2,4)
nx.draw(g)
```

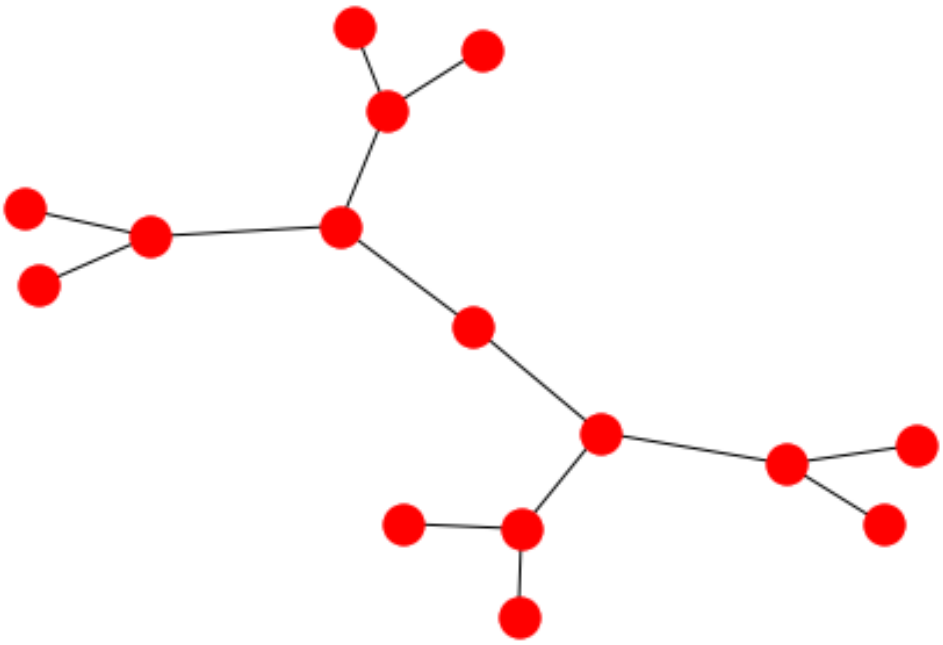
```
/opt/conda/lib/python3.6/site-packages/networkx/drawing/nx_pylab.py:611: MatplotlibDeprecationWa
  if cb.is_numlike(alpha):
```



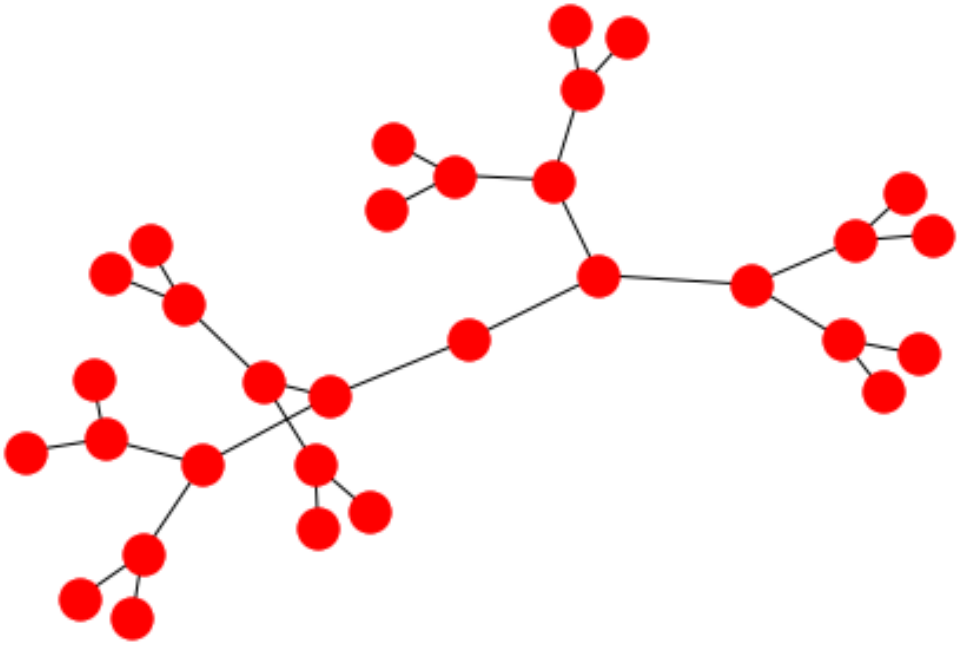
In [7]:
```
g.clear()
add_balanced_tree(2, g)
nx.draw(g)
```



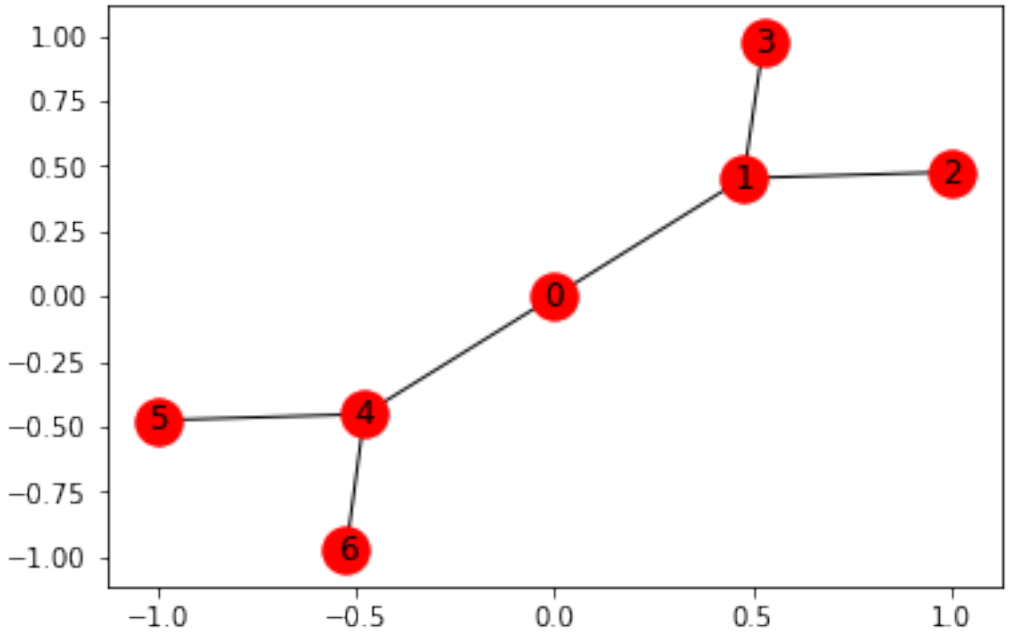
In [8]:
```
g.clear()
add_balanced_tree(3, g)
nx.draw(g)
```



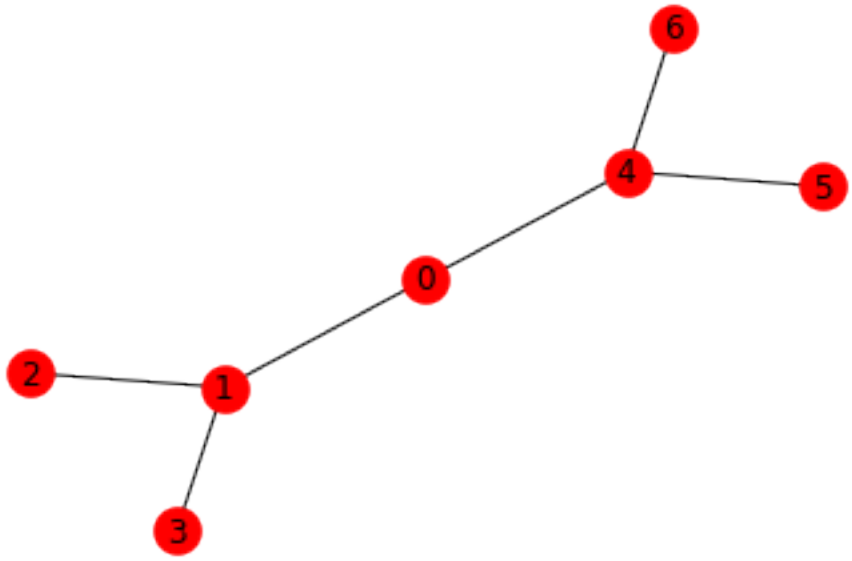
In [9]:
```
g.clear()
add_balanced_tree(4, g)
nx.draw(g)
```



In [10]:
```
g = nx.Graph()
add_balanced_tree(2, g)
nx.draw_networkx(g)
```

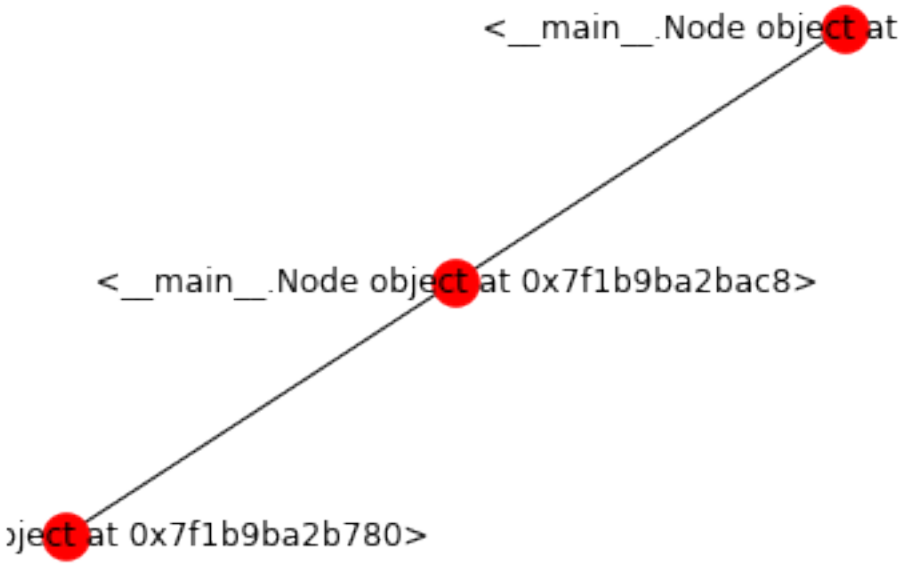


In [11]:
```
_ = plt.axis("off")
nx.draw_networkx(g)
```

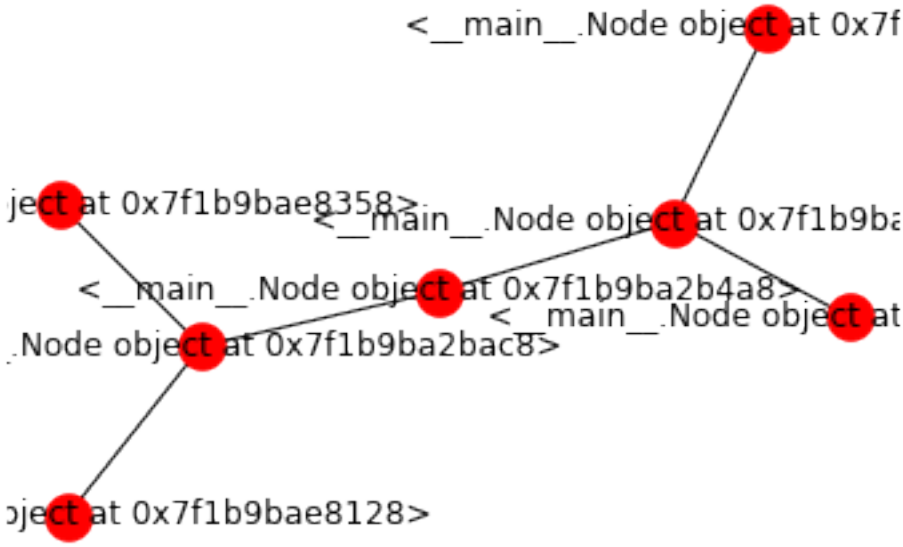


In [12]:
```
class Node():
    pass

def add_balanced_tree2(height, gt): # returns the root of the subtree
    root = Node()
    gt.add_node(root)# add root node
    if height == 0:
        pass
    else:
        # create two subtrees of smaller height
        # create two subtrees of smaller height
        gt.add_edge(root, add_balanced_tree2(height-1, gt)) # connect root node of subt

        gt.add_edge(root, add_balanced_tree2(height-1, gt)) # connect root node of subt
    return root
```

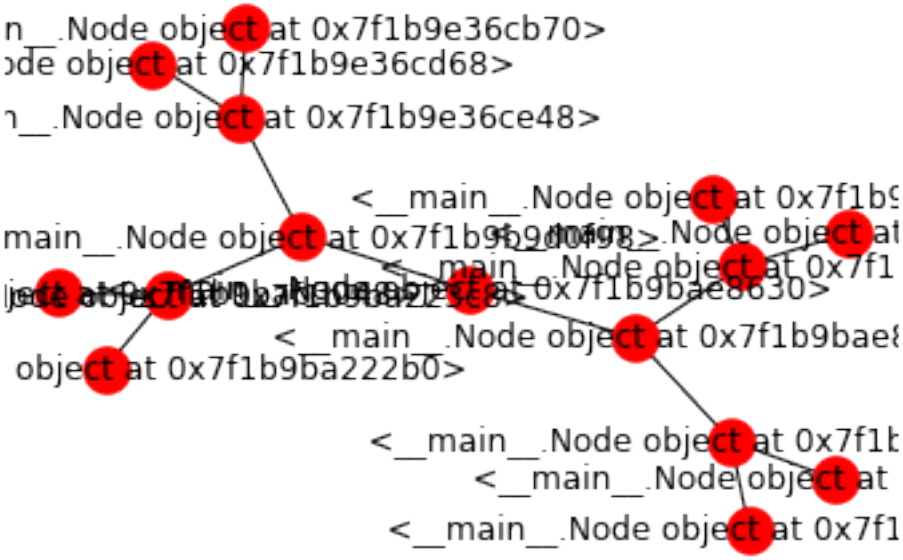
In [13]:
```
g2 = nx.Graph()
add_balanced_tree2(1, g2)
nx.draw_networkx(g2)
_ = plt.axis("off")
```



In [14]:
```
g2.clear()
add_balanced_tree2(2, g2)
nx.draw_networkx(g2)
_ = plt.axis("off")
```



```
In [15]: g2.clear()
         add_balanced_tree2(3, g2)
         nx.draw_networkx(g2)
         _ = plt.axis("off")
```




```
In [18]: list(g2.nodes())
```

```
Out[18]: [<__main__.Node at 0x7f1b9bae8630>,
          <__main__.Node at 0x7f1b9bae8ac8>,
          <__main__.Node at 0x7f1b9bae8128>,
          <__main__.Node at 0x7f1b9bae8358>,
          <__main__.Node at 0x7f1b9ba2bac8>,
          <__main__.Node at 0x7f1b9ba2b4a8>,
          <__main__.Node at 0x7f1b9b9d0cc0>,
          <__main__.Node at 0x7f1b9b9d0e48>,
          <__main__.Node at 0x7f1b9b9d0f98>,
          <__main__.Node at 0x7f1b9ba223c8>,
          <__main__.Node at 0x7f1b9ba222b0>,
          <__main__.Node at 0x7f1b9bab3048>,
          <__main__.Node at 0x7f1b9e36ce48>,
          <__main__.Node at 0x7f1b9e36cb70>,
          <__main__.Node at 0x7f1b9e36cd68>]
```

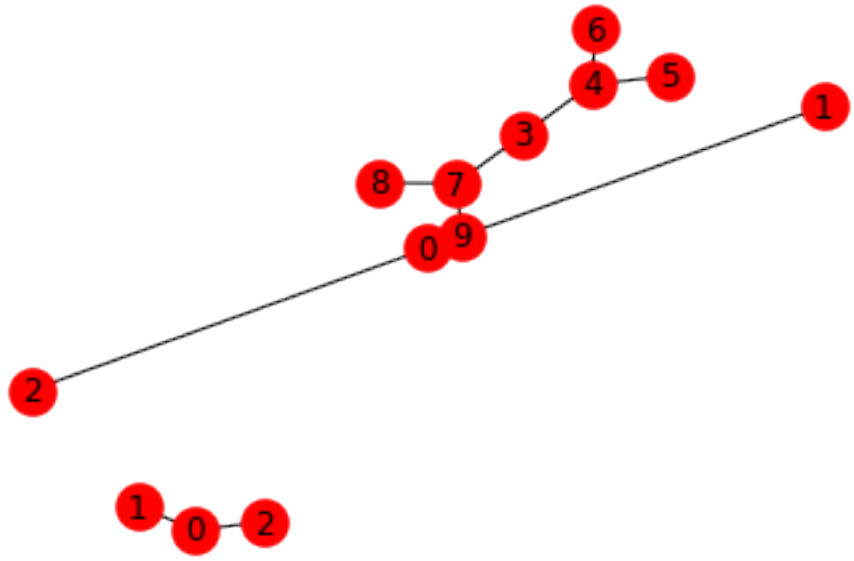
```
In [17]: list(g.nodes())
```

```
Out[17]: [0, 1, 2, 3, 4, 5, 6]
```

```
In [19]: list(g.edges())
```

```
Out[19]: [(0, 1), (0, 4), (1, 2), (1, 3), (4, 5), (4, 6)]
```

```
In [28]: g.clear()
         # g = nx.Graph()
         add_balanced_tree(1,g)
         nx.draw_networkx(g)
         _ = plt.axis('off')
         add_balanced_tree(2, g)
         nx.draw_networkx(g, pos=nx.spring_layout(g))
```




```
In [26]: pos = nx.spring_layout(g)
         pos
```

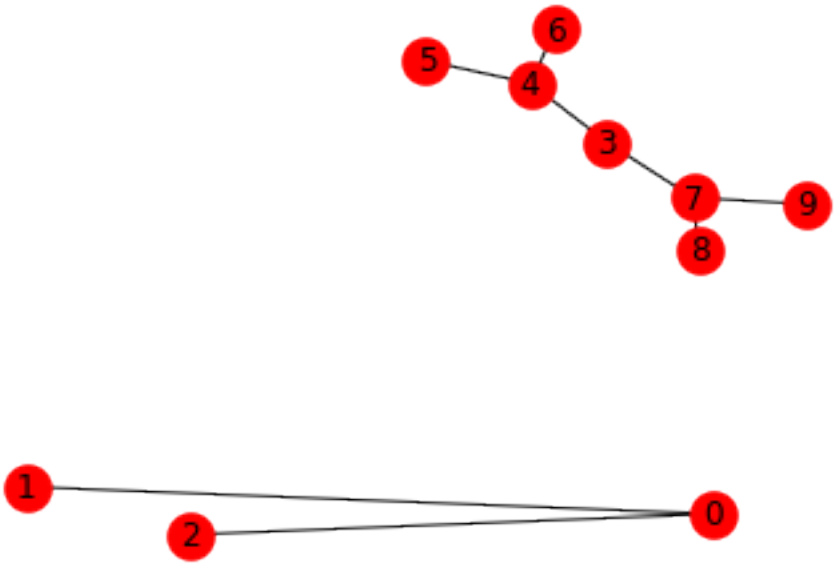
```
Out[26]: {0: array([-0.58965815, -0.92690693]),
          1: array([-0.7157853 , -0.83160401]),
          2: array([-0.44933957, -1.        ]),
          3: array([0.23422443, 0.38046466]),
          4: array([0.10918619, 0.58754975]),
          5: array([-0.06219162,  0.66656795]),
          6: array([0.1498883, 0.7758508]),
          7: array([0.37568708, 0.18539011]),
          8: array([0.38442995, 0.00070111]),
          9: array([0.5635587 , 0.16198656])}
```

0.2 Advanced:ű

export to cytoscape for manual tuning, use file, or API and export the result from cytoscape back to networkx

```
In [66]: nx.draw_networkx(g, pos=pos)
         plt.axis("off")

Out[66]: (-0.8082061441934238,
          0.6559795448795895,
          -1.132603611517202,
          0.908454406832836)
```


```
In [67]: pos[0][0]+=1
         nx.draw_networkx(g, pos=pos)
         plt.axis("off")

Out[67]: (-0.8082061441934238,
          0.6559795448795895,
          -1.132603611517202,
          0.908454406832836)
```



0.3 Ring Network

0.3.1 Connect both first and second neighbours

Number of nodes should be a parameter

```
In [72]: n = int(input("Number of nodes: "))
```

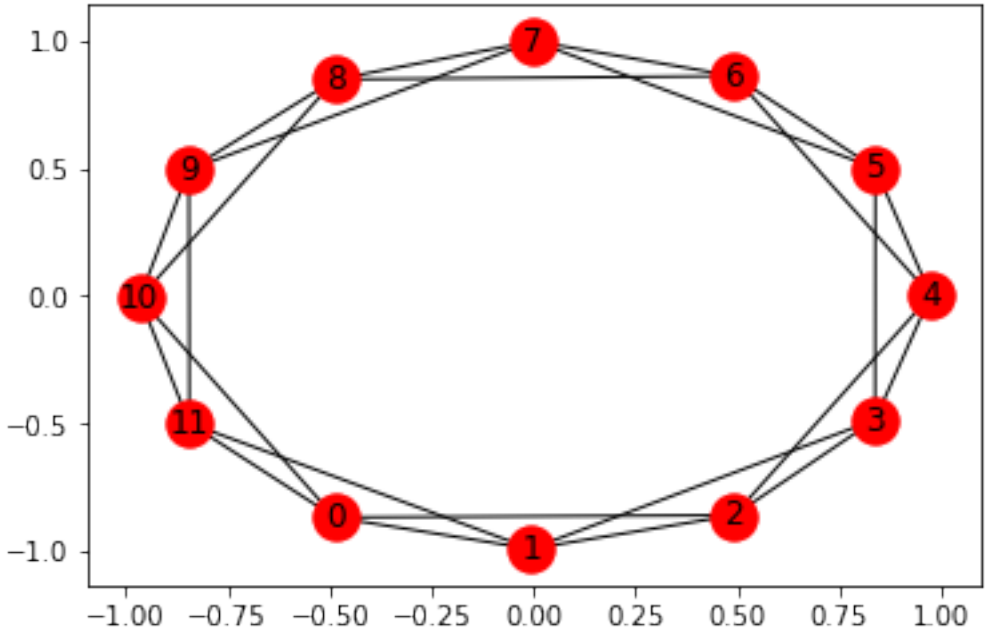
```
Number of nodes: 12
```

```
In [73]: def ring_network(num_nodes):
             "Create a ring network with first- and second neighbor connections"
             ring = networkx.Graph()
             for i in range(num_nodes):
                 ring.add_edge(i, (i+1)%num_nodes)
                 ring.add_edge(i, (i+2)%num_nodes)
             return ring
```

```
         def ring_network(n):
             graph = nx.Graph()
             graph.add_node(0)
             for i in range(1,n):
                 graph.add_node(i)
                 graph.add_edge(i-1,i)
                 if i >= 2:
```

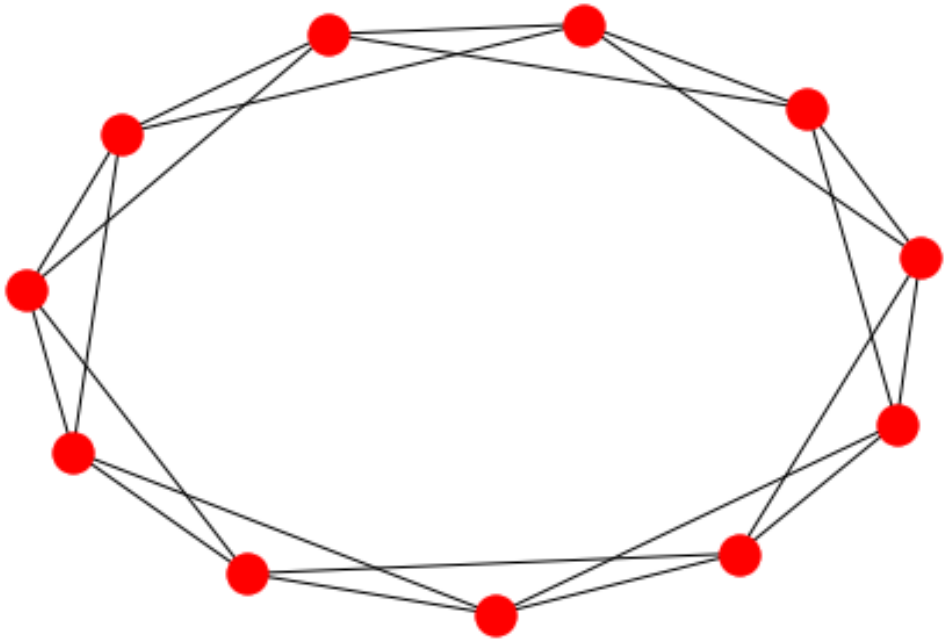
```
            graph.add_edge(i, i-2)
        if i == n - 1 :
            graph.add_edge(i,0)
            graph.add_edge(i,1)
        if i == n - 2:
            graph.add_edge(i,0)
    return graph
```

In [74]:
```
nx.draw_networkx(ring_network(n))
```

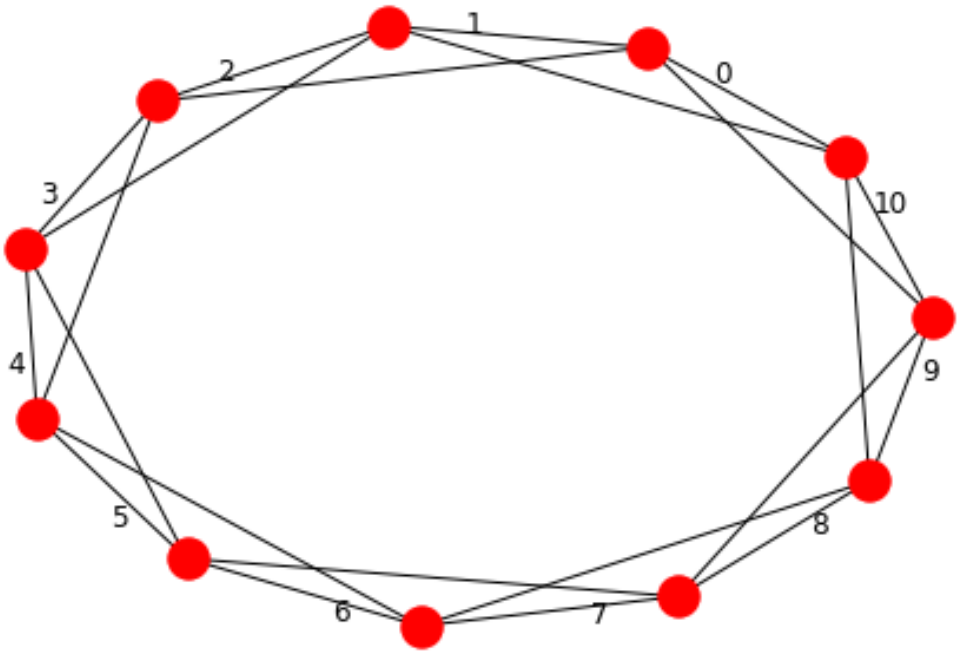


In [75]:
```
ring_network?
```

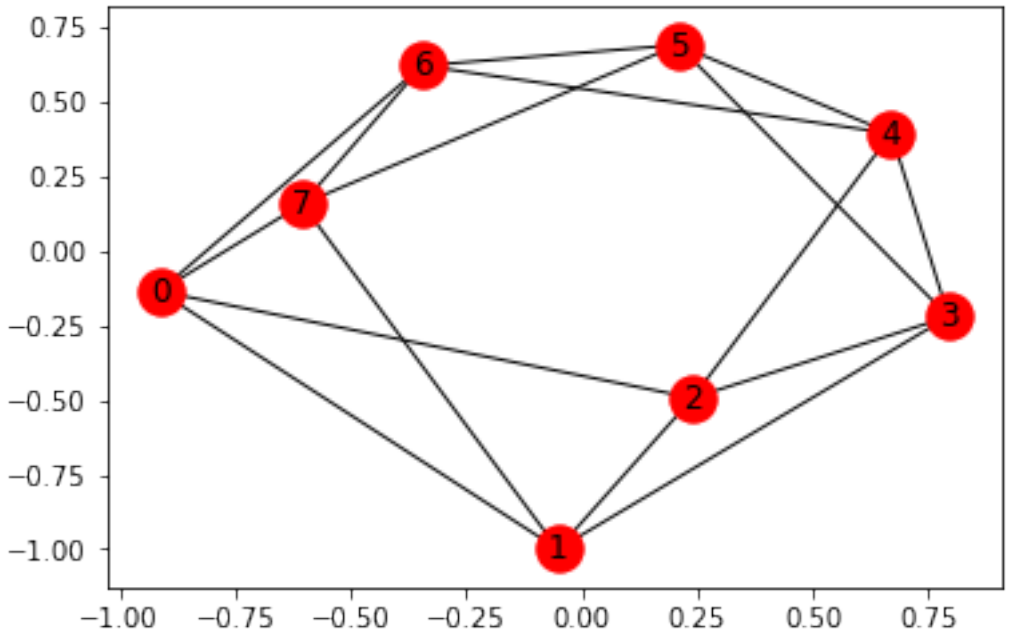
In [77]:
```
nx.draw(ring_network(11))
```



```
In [78]: g=ring_network(11)
         nx.draw(g)
         pos = nx.spring_layout(g)
         nx.draw_networkx_labels(g,pos=pos)

Out[78]: {0: Text(0.5435208347443904, 0.8349943720402971, '0'),
          1: Text(0.006860483522966265, 0.9999999999999999, '1'),
          2: Text(-0.5343606661508051, 0.8439280958920684, '2'),
          3: Text(-0.9165464884494507, 0.42956877639410607, '3'),
          4: Text(-0.9879977365081297, -0.12832682317137237, '4'),
          5: Text(-0.760921575927415, -0.6415758051091925, '5'),
          6: Text(-0.2847946285421735, -0.9568896913064655, '6'),
          7: Text(0.277466491541255, -0.9650539565325327, '7'),
          8: Text(0.7542062878218528, -0.6641177389881053, '8'),
          9: Text(0.9936413380631587, -0.15115356648612643, '9'),
          10: Text(0.9089256598843498, 0.3986263372673211, '10')}
```




```
In [81]: def ring_network2(num_nodes):
             "Create a ring network with first- and second neighbor connections and weights"
             ring = nx.Graph()
             for i in range(num_nodes):
                 ring.add_edge(i, (i+1)%num_nodes, weight=i)
                 ring.add_edge(i, (i+2)%num_nodes, weight=)
             return ring

In [90]: nx.draw_networkx(ring_network2(8))
```




```
In [98]: g = ring_network2(10)

In [99]: list(g.edges())

Out[99]: [(0, 1),
          (0, 2),
          (0, 8),
          (0, 9),
          (1, 2),
          (1, 3),
          (1, 9),
          (2, 3),
          (2, 4),
          (3, 4),
          (3, 5),
          (4, 5),
          (4, 6),
          (5, 6),
          (5, 7),
          (6, 7),
          (6, 8),
          (7, 8),
          (7, 9),
          (8, 9)]

In [100]: list(g.nodes())
```

```
Out[100]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [101]: list(g.edges(data = True))

Out[101]: [(0, 1, {'weight': 0}),
           (0, 2, {}),
           (0, 8, {}),
           (0, 9, {'weight': 9}),
           (1, 2, {'weight': 1}),
           (1, 3, {}),
           (1, 9, {}),
           (2, 3, {'weight': 2}),
           (2, 4, {}),
           (3, 4, {'weight': 3}),
           (3, 5, {}),
           (4, 5, {'weight': 4}),
           (4, 6, {}),
           (5, 6, {'weight': 5}),
           (5, 7, {}),
           (6, 7, {'weight': 6}),
           (6, 8, {}),
           (7, 8, {'weight': 7}),
           (7, 9, {}),
           (8, 9, {'weight': 8})]

In [102]: list(g[0])

Out[102]: [1, 2, 8, 9]

In [108]: list(g[0][1])


        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-108-0877e325582b> in <module>
    ----> 1 list(g[0][0])


        /opt/conda/lib/python3.6/site-packages/networkx/classes/coreviews.py in __getitem__(self
         52
         53     def __getitem__(self, key):
    ---> 54         return self._atlas[key]
         55
         56     def copy(self):


        KeyError: 0
```

```
In [103]: list(g.edges(data = "weight"))

Out[103]: [(0, 1, 0),
           (0, 2, None),
           (0, 8, None),
           (0, 9, 9),
           (1, 2, 1),
           (1, 3, None),
           (1, 9, None),
           (2, 3, 2),
           (2, 4, None),
           (3, 4, 3),
           (3, 5, None),
           (4, 5, 4),
           (4, 6, None),
           (5, 6, 5),
           (5, 7, None),
           (6, 7, 6),
           (6, 8, None),
           (7, 8, 7),
           (7, 9, None),
           (8, 9, 8)]

In [109]: g.edges(data="weight",default=-1)

Out[109]: EdgeDataView([(0, 1, 0), (0, 2, -1), (0, 8, -1), (0, 9, 9), (1, 2, 1), (1, 3, -1), (1,
```