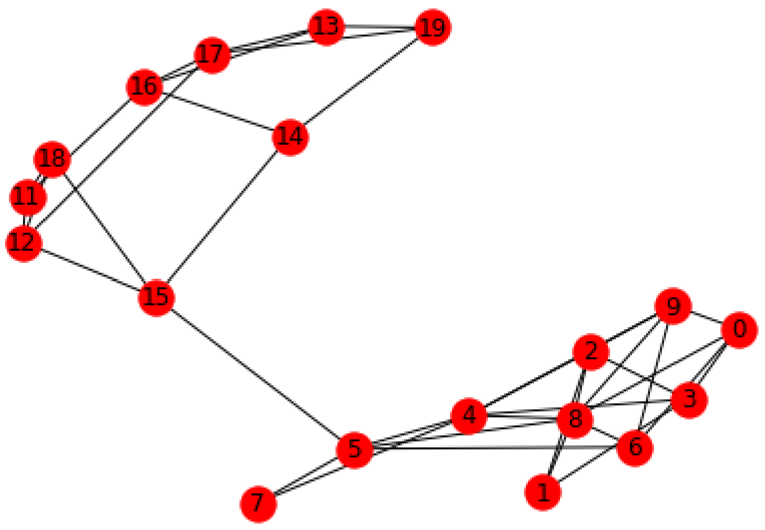
Lec99 Cascading Behavior in network - Impact of communities on cascade

In [13]:

```
import networkx as nx
import random
import matplotlib.pyplot as plt

def create_first_community(G):
    for i in range(10):
        G.add_node(i)
    for i in range(10):
        for j in range(10):
            if i<j:
                r = random.random()
                if r < 0.5: #Probability
                    G.add_edge(i, j)
    return G

def create_second_community(G):
    for i in range(11, 20):
        G.add_node(i)
    for i in range(11, 20):
        for j in range(11, 20):
            if i<j:
                r = random.random()
                if r < 0.5: #Probability
                    G.add_edge(i, j)
    return G


G = nx.Graph()
G = create_first_community(G)
G = create_second_community(G)

#nx.draw(G, with_labels =1)
#plt.show()

G.add_edge(5, 15)

nx.draw(G, with_labels =1)
plt.show()

nx.write_gml(G, 'random_graph_community.gml')
```



In [14]:

```
# All are doing action B. New Action A comes in
def set_all_B(G):
    for each in G.nodes():
        G.node[each]['action'] = 'B'

def set_A(G, list1):
    for each in list1:
        G.node[each]['action'] = 'A'

def get_colors(G):
    list2 = []
    for each in G.nodes():
        if G.node[each]['action'] == 'B':
            list2.append('red')
        else:
            list2.append('green')
    return list2

def find_neigh(each, c, G):
    num = 0
    for each2 in G.neighbors(each):
        if  G.node[each2]['action'] == c:
            num += 1
    return num

def recalculate_options(G):
    dict1 = {}
    # Payoff(A) = a = 4
    # Payoff(B) = b = 3
    a = 15
    b = 5
    for each in G.nodes():
        num_A = find_neigh(each, 'A', G)
        num_B = find_neigh(each, 'B', G)
        payoff_A = a*num_A
        payoff_B = b*num_B
        if payoff_A >= payoff_B:
            dict1[each] = 'A'
        else:
            dict1[each] = 'B'
    return dict1

G = nx.read_gml('random_graph_community.gml')

def reset_node_attributes(G, action_dict):
    for each in action_dict:
        G.node[each]['action'] = action_dict[each]

def terminate_1(c, G):
    f = 1
    for each in G.nodes():
        if G.node[each]['action'] != c:
            f = 0
            break
    return f

def terminate(G, count):
    flag1
    = terminate_1('A', G)

    flag2 = terminate_1('B', G)
    if flag1 == 1 or flag2 == 1 or count >= 100:
        return 1
    else:
        return 0



set_all_B(G)
list1 = [0, 1]
set_A(G, list1)

colors = get_colors(G)
nx.draw(G, with_labels=1, node_color= colors, node_size=800)
plt.show()

flag = 0
count = 0
while(1):



    raw_input("Enter any text to continue..")
    flag = terminate(G, count)
    if flag == 1:
        break
    count += 1
    action_dict = recalculate_options(G)
    reset_node_attributes(G, action_dict)

    colors = get_colors(G)
    nx.draw(G, with_labels=1, node_color= colors, node_size=800)
    plt.show()

c = terminate_1('A', G)
if c == 1:
    print "Complete Cascade"
else:
     print "Incomplete Cascade"
colors = get_colors(G)
nx.draw(G, with_labels=1, node_color= colors, node_size=800)
plt.show()
```

```
  File "<ipython-input-14-be734c7f6510>", line 60
    = terminate_1('A', G)
    ^
SyntaxError: invalid syntax
```

Lec99 Cascading Behavior in network - Cascades and Clusters

In [12]:

```
import networkx as nx
import random
import matplotlib.pyplot as plt

# All are doing action B. New Action A comes in
def set_all_B(G):
    for each in G.nodes():
        G.node[each]['action'] = 'B'

def set_A(G, list1):
    for each in list1:
        G.node[each]['action'] = 'A'

def get_colors(G):
    list2 = []
    for each in G.nodes():
        if G.node[each]['action'] == 'B':
            list2.append('red')
        else:
            list2.append('green')
    return list2

def find_neigh(each, c, G):
    num = 0
    for each2 in G.neighbors(each):
        if  G.node[each2]['action'] == c:
            num += 1
    return num

def recalculate_options(G):
    dict1 = {}
    # Payoff(A) = a = 4
    # Payoff(B) = b = 3
    a = 3
    b = 2
    for each in G.nodes():
        num_A = find_neigh(each, 'A', G)
        num_B = find_neigh(each, 'B', G)
        payoff_A = a*num_A
        payoff_B = b*num_B
        if payoff_A >= payoff_B:
            dict1[each] = 'A'
        else:
            dict1[each] = 'B'
    return dict1



def reset_node_attributes(G, action_dict):
    for each in action_dict:
        G.node[each]['action'] = action_dict[each]

def terminate_1(c, G):
    f = 1
    for each in G.nodes():
        if G.node[each]['action'] != c:
            f = 0
            break
    return f
```

```
def terminate(G, count):
    flag1 = terminate_1('A', G)
    flag2 = terminate_1('B', G)
    if flag1 == 1 or flag2 == 1 or count >= 100:
        return 1
    else:
        return 0



G = nx.Graph()
G.add_edges_from([(0,1), (0,6), (1,2), (1,8), (1,12), (2,9), (2,12),(3,4), (3,9),(3
                )
list2 = [[0,1,2,3], [0,2,3,4], [1,2,3,4], [3,4,5,6], [2,3,4,5], [4,5,6,12], [2,3,4,

for list1 in list2:
    print list1

    set_all_B(G)
    set_A(G, list1)

    colors = get_colors(G)
    nx.draw(G, with_labels=1, node_color= colors, node_size=800)
    plt.show()

    flag = 0
    count = 0
    while(1):
        #raw_input("Enter any text to continue..")
        flag = terminate(G, count)
        if flag == 1:
            break
        count += 1
        action_dict = recalculate_options(G)
        reset_node_attributes(G, action_dict)

        #colors = get_colors(G)
        #nx.draw(G, with_labels=1, node_color= colors, node_size=800)
        #plt.show()

    c = terminate_1('A', G)
    if c == 1:
        print "Complete Cascade"
    else:
         print "Incomplete Cascade"
    colors = get_colors(G)
    nx.draw(G, with_labels=1, node_color= colors, node_size=800)
    plt.show()
    raw_input("Enter any text to continue..")
```

```
[0, 1, 2, 3]
```