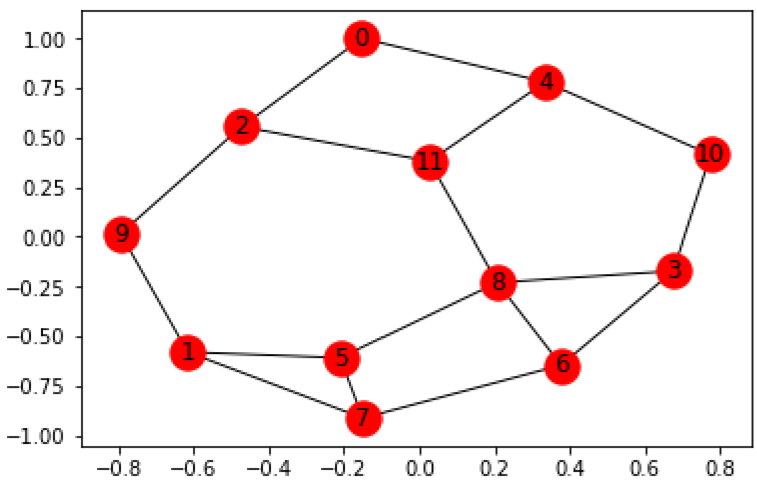
4. Visualization

Author : Dongsoo S. Kim (dskim@iupui.edu (mailto:dskim@iupui.edu))
Date : Feb. 8, 2016; Sep 9, 2019

In [1]:
```
%matplotlib inline
import networkx as nx
import matplotlib.pylab as plt
import random
```

Let's fix node positions

In [2]:
```
G = nx.read_gexf('netlab/data/us-att.gexf', node_type=int)
layout = nx.spring_layout(G)
nx.draw_networkx(G, pos=layout)
```



In [ ]:
```

```

Data struture of the layout

The layout data is a dictionary with

- nodes as *keys* and
- positions as *values* in array format of numpy.

We can make customized layout data. investigate "us-att" data.

In [3]:
```python
G.nodes(data=True)
```

Out[3]:
```
NodeDataView({0: {'latitude': 40.712756, 'abbr': 'nwy', 'name': 'New York, NY',
'longitude': -74.006047, 'population': 8175133, 'label': '0'}, 1: {'latitude':
33.94352, 'abbr': 'lax', 'name': 'Los Angeles, CA', 'longitude': -118.40866, 'p
opulation': 3792621, 'label': '1'}, 2: {'latitude': 41.878247, 'abbr': 'chi',
'name': 'Chicago, IL', 'longitude': -87.629767, 'population': 2695598, 'label':
'2'}, 3: {'latitude': 29.76429, 'abbr': 'hst', 'name': 'Houston, TX', 'longitud
e': -95.3837, 'population': 2099451, 'label': '3'}, 4: {'latitude': 39.952622,
'abbr': 'phl', 'name': 'Philadelphia, PA', 'longitude': -75.165708, 'populatio
n': 1526006, 'label': '4'}, 5: {'latitude': 33.445412, 'abbr': 'phx', 'name':
'Phoenix, AR', 'longitude': -112.073961, 'population': 1445632, 'label': '5'},
6: {'latitude': 29.42373, 'abbr': 'san', 'name': 'San Antonio, TX', 'longitud
e': -98.49438, 'population': 1327407, 'label': '6'}, 7: {'latitude': 32.715786,
'abbr': 'sdg', 'name': 'San Diego, CA', 'longitude': -117.15834, 'population':
1307402, 'label': '7'}, 8: {'latitude': 32.803468, 'abbr': 'dal', 'name': 'Dall
as, TX', 'longitude': -96.769879, 'population': 1197816, 'label': '8'}, 9: {'la
titude': 37.339458, 'abbr': 'sjs', 'name': 'San Jose, CA', 'longitude': -121.89
5022, 'population': 945942, 'label': '9'}, 10: {'latitude': 30.332428, 'abbr':
'jkv', 'name': 'Jacksonville, FL', 'longitude': -81.656165, 'population': 82178
4, 'label': '10'}, 11: {'latitude': 39.768663, 'abbr': 'ind', 'name': 'Indianap
olis, IN', 'longitude': -86.159855, 'population': 820445, 'label': '11'}})
```

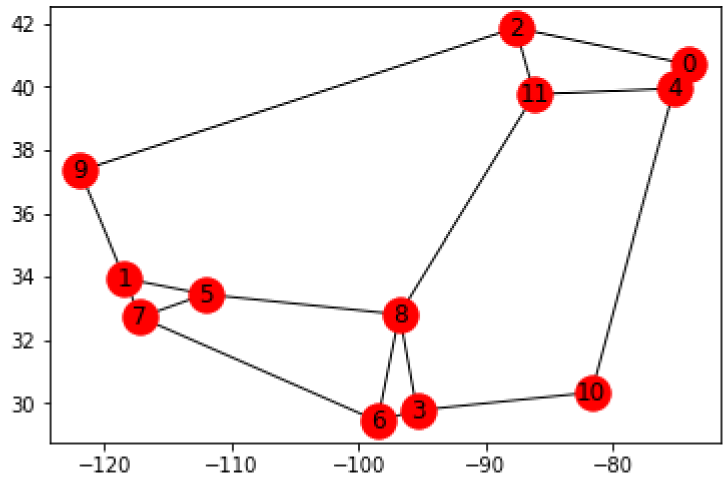
Atlas

The data contains 'latitude' and 'longitude'. Let's use them for node positioning.

In [4]:
```python
import numpy as np
px = nx.get_node_attributes(G, 'longitude').values()
py = nx.get_node_attributes(G, 'latitude').values()
pos = zip(px, py)            # change them to a list of tuples
layout = dict(zip(G, pos))   # combine the node list and the position list
layout
```

Out[4]:
```
{0: (-74.006047, 40.712756),
 1: (-118.40866, 33.94352),
 2: (-87.629767, 41.878247),
 3: (-95.3837, 29.76429),
 4: (-75.165708, 39.952622),
 5: (-112.073961, 33.445412),
 6: (-98.49438, 29.42373),
 7: (-117.15834, 32.715786),
 8: (-96.769879, 32.803468),
 9: (-121.895022, 37.339458),
 10: (-81.656165, 30.332428),
 11: (-86.159855, 39.768663)}
```

In [5]:
```
nx.draw_networkx(G, pos=layout)
```

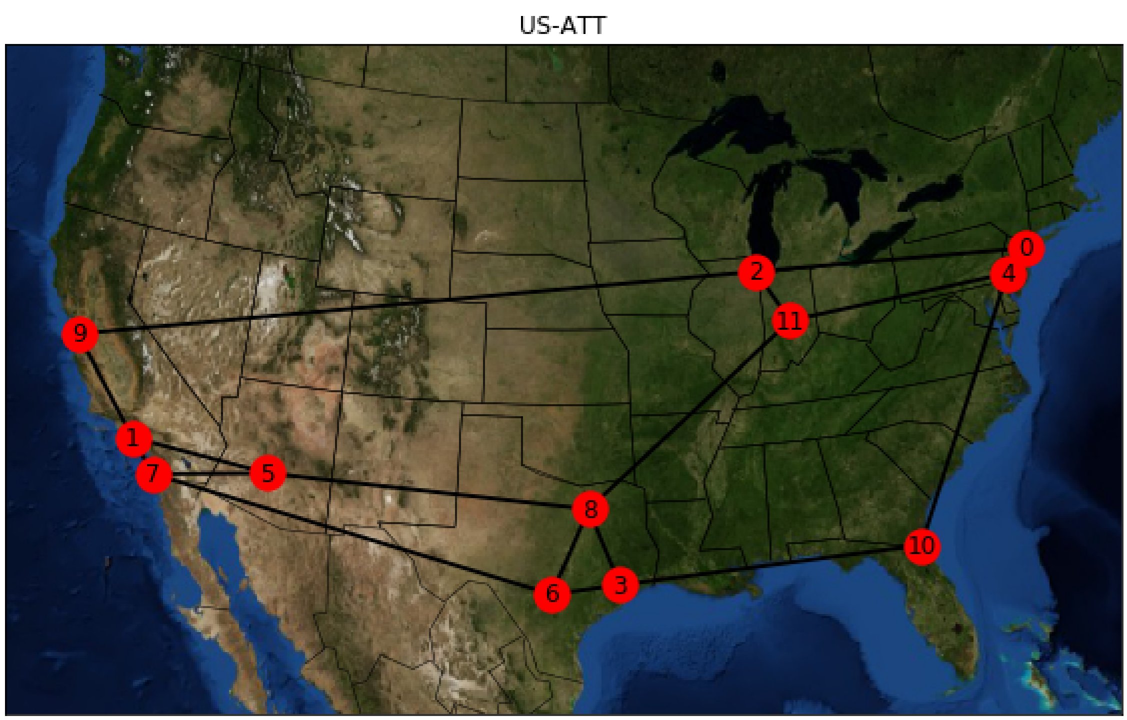


Which one is which one?

Even with the positioning, it is quite difficult to imagine the graph representing US information highway.
Why?
Because there is no information about an atlas map.
Let's draw the network on the top of US atlas map.

In [6]:

```
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt

plt.figure(1,figsize=(10,10))
lons = list(nx.get_node_attributes(G, 'longitude').values())
lats = list(nx.get_node_attributes(G, 'latitude').values())

m = Basemap(width=5.e6,height=3.e6,
            projection='gnom',lat_0=38.,lon_0=-98.)

m.drawcountries()
m.drawstates()
m.bluemarble()

px, py = m(lons,lats)               # The coordination must be projected to
pos = zip(px, py)      # Convert the projected data to a dictionary of node I
layout = dict(zip(G, pos))      # and NumPy array.
nx.draw_networkx(G, pos=layout, width=2.0)

plt.title('US-ATT')
plt.show()
```



In [7]:
```
help(plt.figure)
```

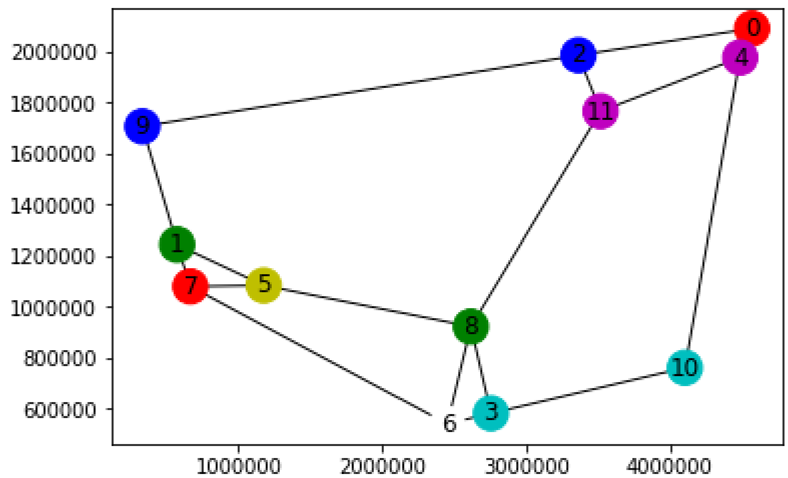
```
Help on function figure in module matplotlib.pyplot:

figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameo
n=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False, **kwargs)
    Creates a new figure.

    Parameters
    ----------

    num : integer or string, optional, default: none
        If not provided, a new figure will be created, and the figure number
        will be incremented. The figure objects holds this number in a `number`
        attribute.
        If num is provided, and a figure with this id already exists, make
        it active, and returns a reference to it. If this figure does not
        exists, create it and returns it.
        If num is a string, the window title will be set to this figure's
        `num`.

    figsize : tuple of integers, optional, default: None
        width, height in inches. If not provided, defaults to rc
        figure.figsize.

    dpi : integer, optional, default: None
        resolution of the figure. If not provided, defaults to rc figure.dpi.

    facecolor :
        the background color. If not provided, defaults to rc figure.facecolor.

    edgecolor :
        the border color. If not provided, defaults to rc figure.edgecolor.

    frameon : bool, optional, default: True
        If False, suppress drawing the figure frame.

    FigureClass : class derived from matplotlib.figure.Figure
        Optionally use a custom Figure instance.

    clear : bool, optional, default: False
        If True and the figure already exists, then it is cleared.

    Returns
    -------
    figure : Figure
        The Figure instance returned will also be passed to new_figure_manager
        in the backends, which allows to hook custom Figure classes into the
        pylab interface. Additional kwargs will be passed to the figure init
        function.

    Notes
    -----
    If you are creating many figures, make sure you explicitly call "close"
    on the figures you are not using, because this will enable pylab
    to properly clean up the memory.
```

```
rcParams defines the default values, which can be modified in the
matplotlibrc file
```

Characterizing node

The node colors can be customized by the name parameter "node_color".

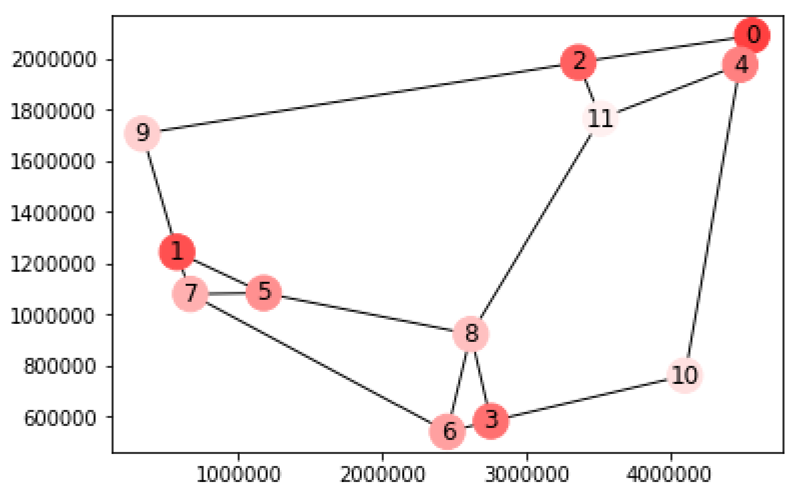
In [8]:
```python
color = ['r', 'g', 'b', 'c', 'm', 'y', 'w', 'r', 'g', 'b', 'c', 'm']
nx.draw_networkx(G, pos=layout, node_color=color)
```



Fine control of the color

Moreover, the hexadecimal RGB color code can be used.

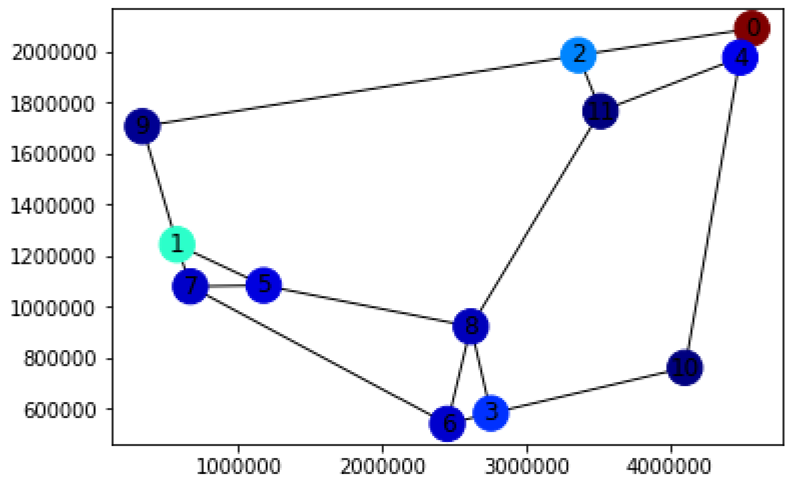
In [9]:
```python
color = ['#FF4040', '#FF5050','#FF6060', '#FF7070', '#FF8080', '#FF9090',
         '#FFA0A0', '#FFB0B0','#FFC0C0', '#FFD0D0', '#FFE0E0', '#FFF0F0']
nx.draw_networkx(G, pos=layout, node_color=color)
```



Visualize the "Population"

In [10]:
```
import matplotlib.colors as colors
import matplotlib.cm as cm

pop = list(nx.get_node_attributes(G, 'population').values())
cval = np.array(pop)
print (cval)
color_map = cm.get_cmap('jet')    # [1]
nx.draw_networkx(G, pos=layout,
                 node_color=cval,
                 cmap=color_map, vmin=np.min(cval), vmax=np.max(cval))
```

```
[8175133 3792621 2695598 2099451 1526006 1445632 1327407 1307402 1197816
  945942  821784  820445]
```



[1] http://matplotlib.org/examples/color/colormaps_reference.html
(http://matplotlib.org/examples/color/colormaps_reference.html)

Big gap between 1st and 2nd cities

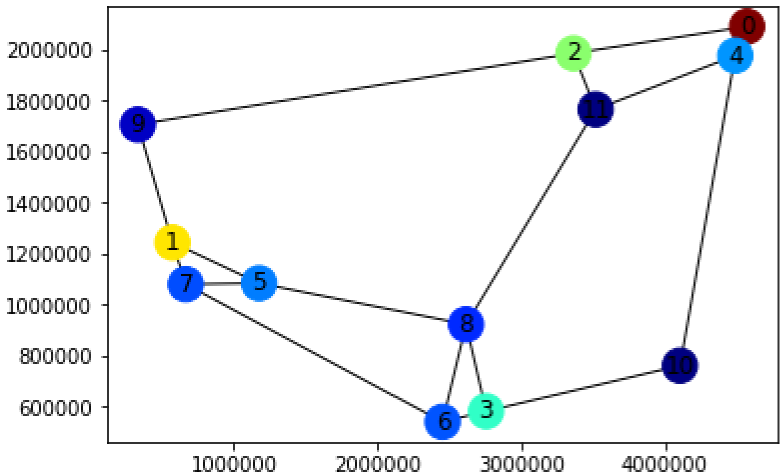
There are big jumps between

- NWY and LAX
- LAX and CHI
  But, there are not much difference among the second half. Instead of the linear mapping, let's use a logarithmic mapping.

In [11]:

```
cval = np.log(np.array(pop))
print (cval)
nx.draw_networkx(G, pos=layout,
                 node_color=cval,
                 cmap=color_map, vmin=np.min(cval), vmax=np.max(cval))
```

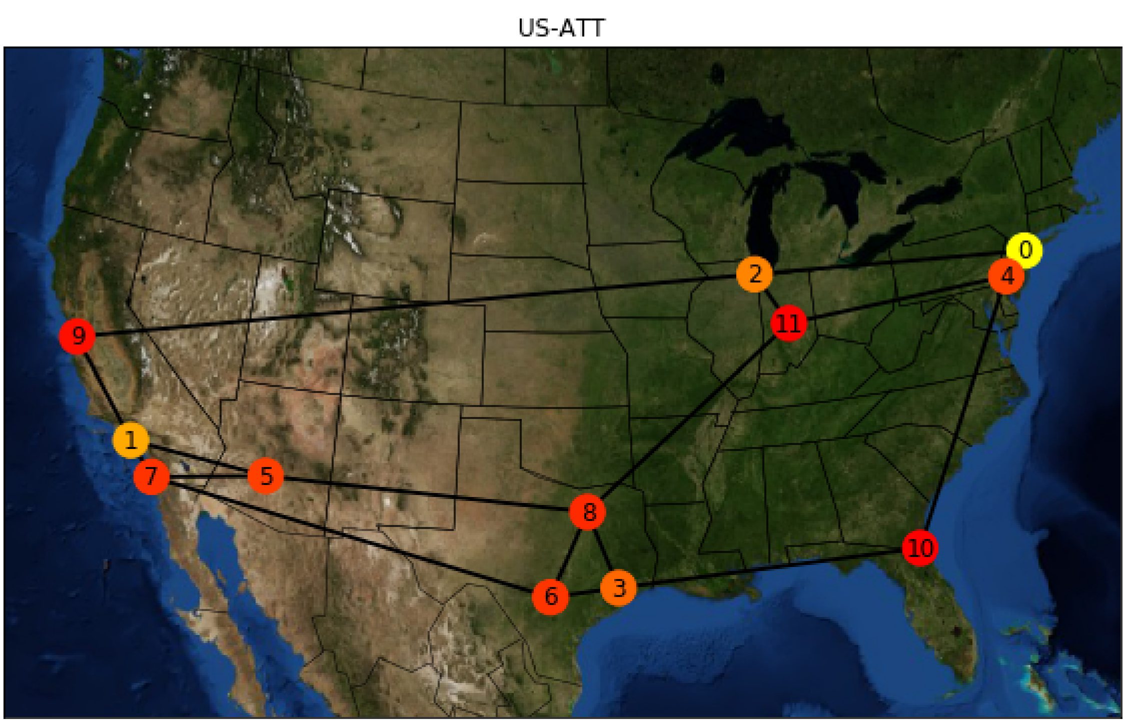
```
[15.91660754 15.14856789 14.80713063 14.55718644 14.23816442 14.18405715
 14.09873797 14.08355252 13.99601046 13.75993654 13.61923287 13.61760215]
```



Put-it-all-together

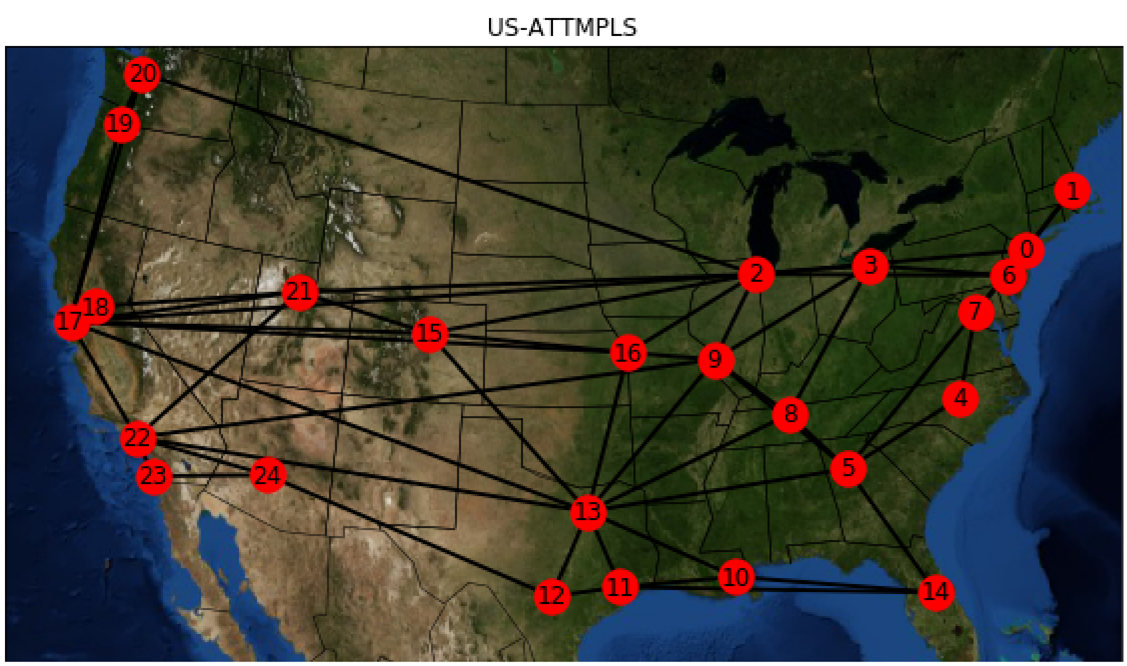
In [12]:

```python

plt.figure(1,figsize=(10,10))
px = list(nx.get_node_attributes(G, 'longitude').values())
py = list(nx.get_node_attributes(G, 'latitude').values())
pop = list(nx.get_node_attributes(G, 'population').values())

cval = np.log(np.array(pop))
color_map = cm.get_cmap('autumn')

m = Basemap(width=5.e6,height=3.e6,\
            projection='gnom',lat_0=38.,lon_0=-98.)

m.drawcountries()
m.drawstates()
m.bluemarble()

px, py = m(px,py)                # The coordination must be projected to the
pos = zip(px, py)      # Convert the projected data to a dictionary of node I
layout = dict(zip(G, pos))       # and NumPy array.

nx.draw_networkx(G, pos=layout,
                 node_color=cval,
                 cmap=color_map, vmin=np.min(cval), vmax=np.max(cval),
                 width=2.0)

plt.title('US-ATT')
plt.show()
```



In [13]:

```python
import networkx as nx
import numpy as np
from mpl_toolkits.basemap import Basemap
import matplotlib.pylab as plt
import matplotlib.colors as colors
import matplotlib.cm as cm
import random

%matplotlib inline

G = nx.read_gexf('netlab/data/attmpls.gexf')

plt.figure(1,figsize=(10,10))
lons = list(nx.get_node_attributes(G, 'longitude').values())
lats = list(nx.get_node_attributes(G, 'latitude').values())

m = Basemap(width=5.e6,height=3.e6,\
            projection='gnom',lat_0=38.,lon_0=-98.)

m.drawcountries()
m.drawstates()
m.bluemarble()

px, py = m(lons, lats)               # The coordination must be projected to
pos = zip(px, py)     # Convert the projected data to a dictionary of node I
layout = dict(zip(G, pos))

nx.draw_networkx(G, pos=layout, width=2.0)

plt.title('US-ATTMPLS')
plt.show()
```



In [ ]:

```python

```