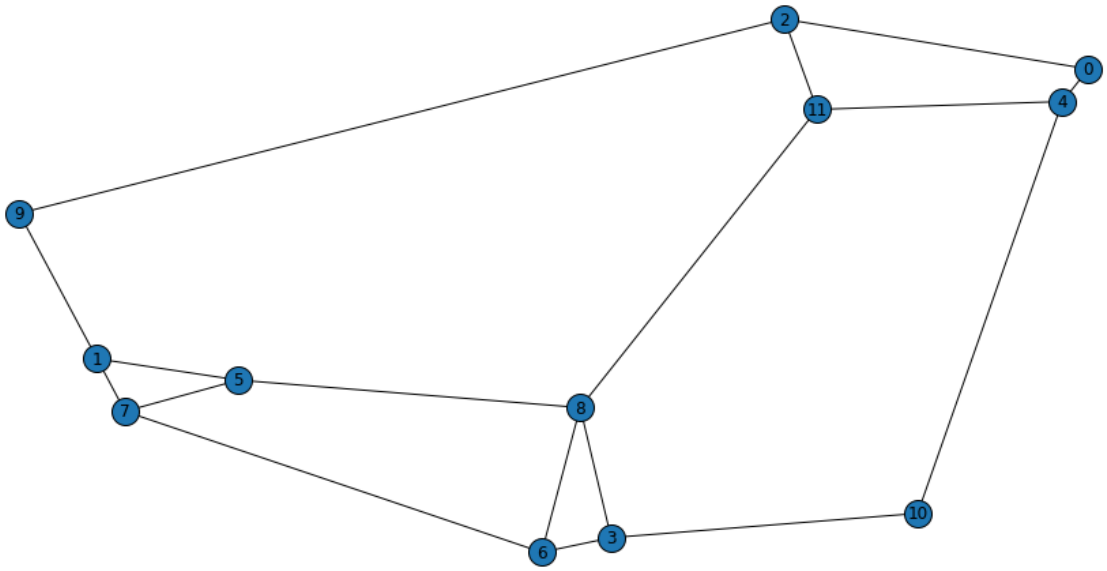
5. Graph Algorithms

In [1]:
```
%matplotlib inline
import networkx as nx
import numpy as np
import matplotlib.pylab as plt
import netlab as nl
```

In [2]:
```
G = nx.read_gexf('data/us-att.gexf', node_type=int)
plt.figure(1,figsize=(12,6))
layout = nl.absolute_layout(G)
nl.draw_atlas(G, pos=layout, node_size=400)
#nl.draw_atlas(G, node_size=400)
```



In [3]:
```
G[0]
```

Out[3]:
```
AtlasView({2: {'distance': 1264, 'capacity': 100, 'id': '0'}, 4: {'distance': 1
52, 'capacity': 100, 'id': '1'}})
```

In [4]:
```
# to find a random MIS
# Note that the MIS is not unique and not a maximum
print (nx.maximal_independent_set(G,[0]))
```

```
[0, 10, 9, 5, 11, 6]
```

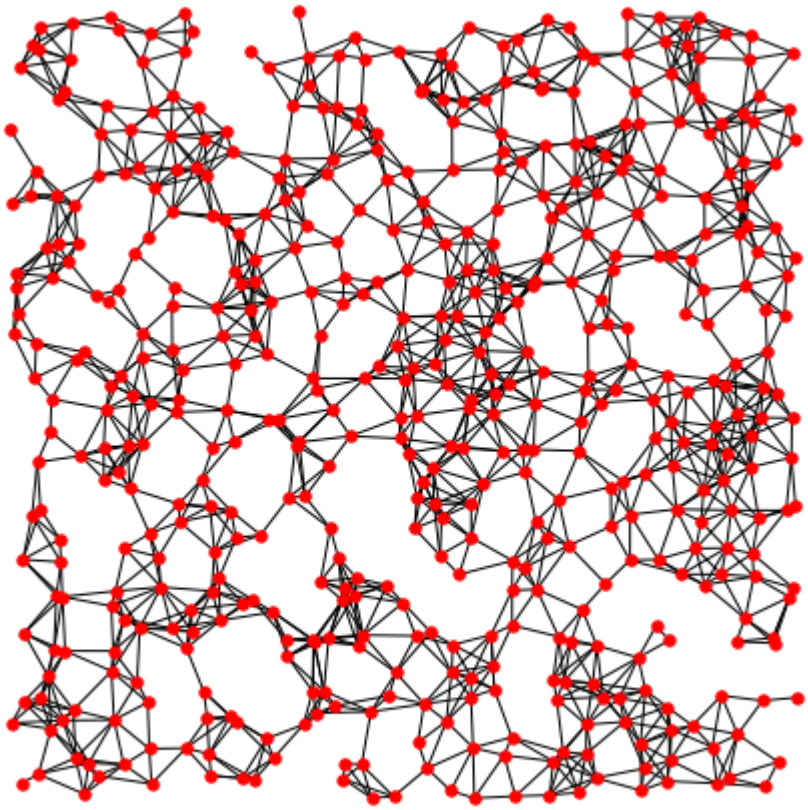
In [5]:
```
# nx.maximal_independent_set??
```

In [6]:
```
# to find an random MIS including specific nodes1
print (nx.maximal_independent_set(G, [6,10]))
```

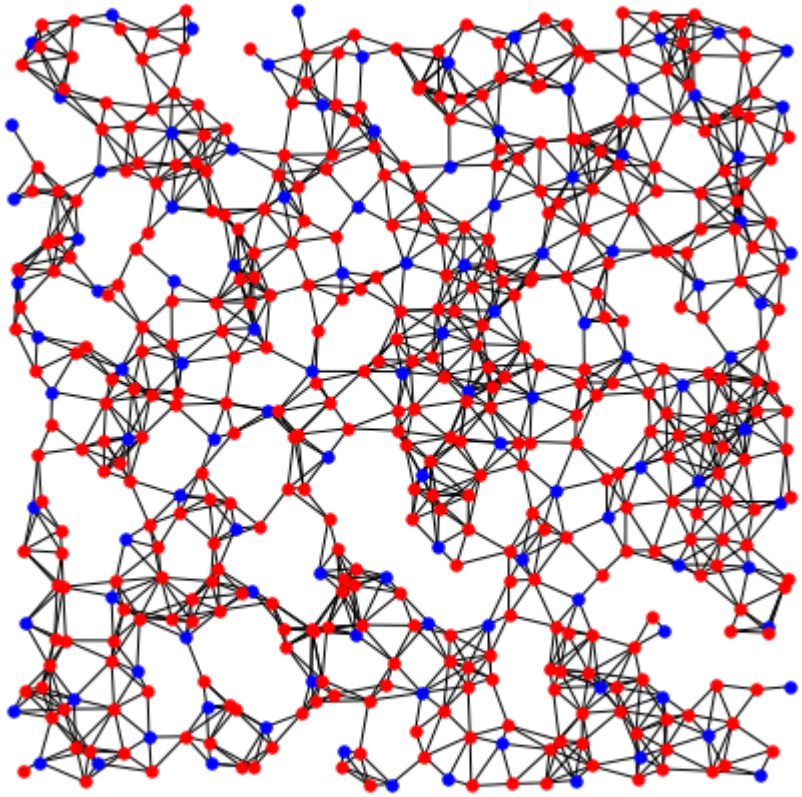
```
[10, 6, 5, 2]
```

In [7]:
```
H = nx.read_gexf('data/SN001.gexf', node_type=int)
# print H.nodes()
layout = nl.absolute_layout(H, xattr='px', yattr='py')
print ('|V| =', H.number_of_nodes())
print ('|E| =', H.number_of_edges())
plt.figure(1,figsize=(6,6))
nl.draw_sensors(H, layout)

```

```
|V| = 521
|E| = 1745
```



In [8]:
```
mis = nx.maximal_independent_set(H)
plt.figure(1,figsize=(6,6))
nl.draw_sensors(H, layout, mis)
```



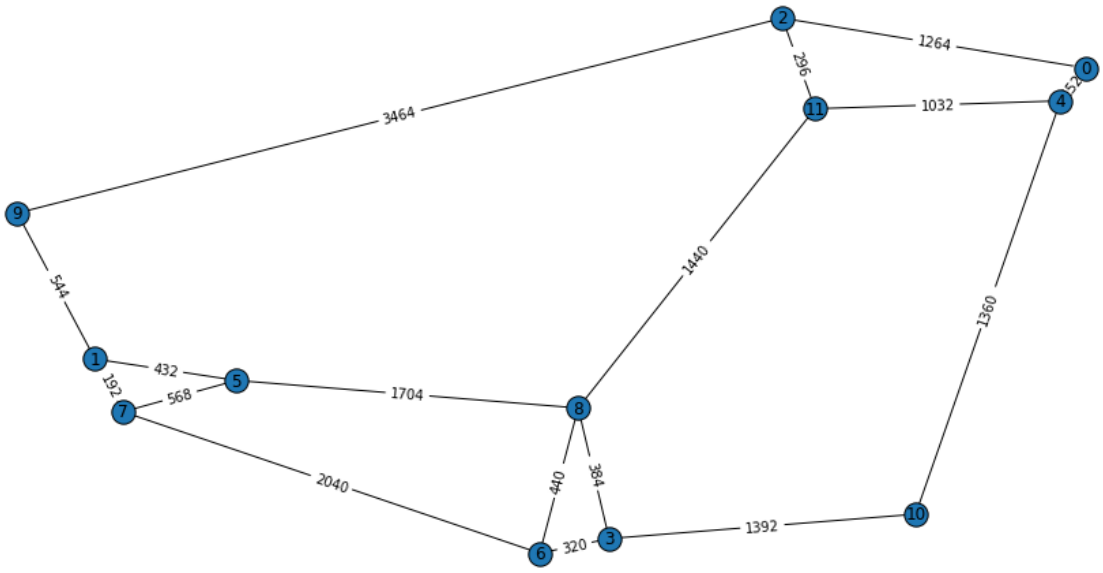
In [9]:
```
for i in range(100):
    mis = nx.maximal_independent_set(H)
    print ('mis %s=' % i, len(mis))
```

```
mis 0= 116
mis 1= 115
mis 2= 117
mis 3= 113
mis 4= 120
mis 5= 116
mis 6= 113
mis 7= 113
mis 8= 114
mis 9= 114
mis 10= 113
mis 11= 113
mis 12= 116
mis 13= 116
mis 14= 113
mis 15= 113
mis 16= 111
mis 17= 114
mis 18= 113
```

In [10]:
```
# to find a dominating set
print ('dominating set=', nx.dominating_set(G))
# the dominating set found is not connected.
```

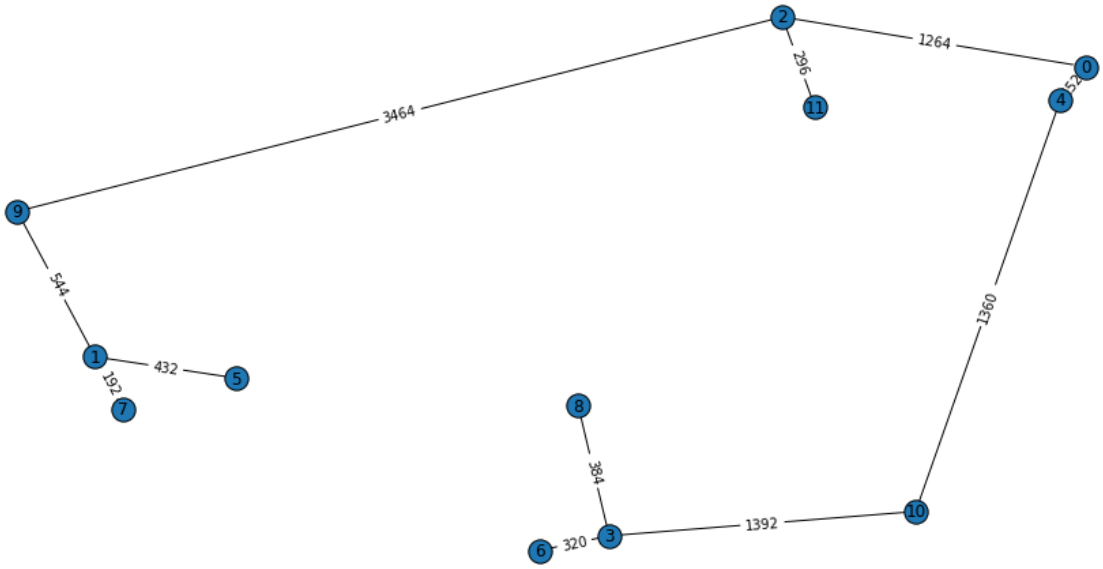
```
dominating set= {0, 1, 3, 11}
```

In [11]:
```
G = nx.read_gexf('netlab/data/us-att.gexf', node_type=int)
layout = nl.absolute_layout(G)
plt.figure(1,figsize=(12,6))
nl.draw_atlas(G, pos=layout, edge_labels='distance')

```



In [12]:
```
# to find an MST with hop count only
m0 = nx.minimum_spanning_tree(G)
# to calculate the sum of some edge attribute, eg. distance
print ('sum of distance m0=', sum([x[2]['distance'] for x in m0.edges(data=T

plt.figure(1,figsize=(12,6))
nl.draw_atlas(m0, pos=layout, edge_labels='distance')
```

```
sum of distance m0= 9800
```

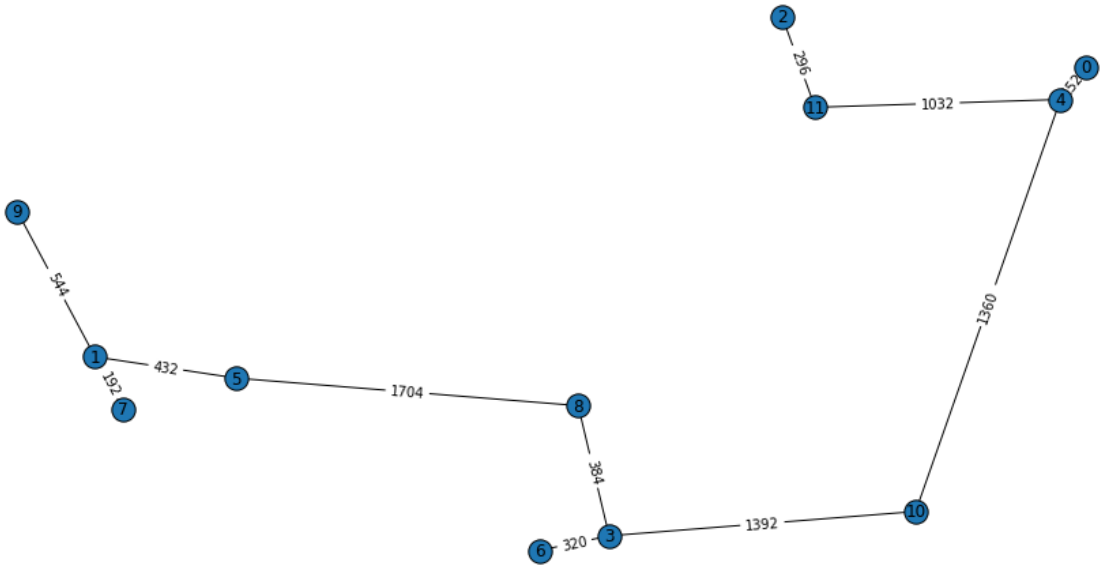


In [13]:
```
m0.nodes(data=True)
```

Out[13]:
```
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'}})
```

In [14]:
```
# to find an MST with edge attribute
m1 = nx.minimum_spanning_tree(G, weight='distance')
print ('sum of distance m1=', sum([x[2]['distance'] for x in m1.edges(data=T

plt.figure(1,figsize=(12,6))
nl.draw_atlas(m1,pos=layout, edge_labels='distance')
plt.show()
```

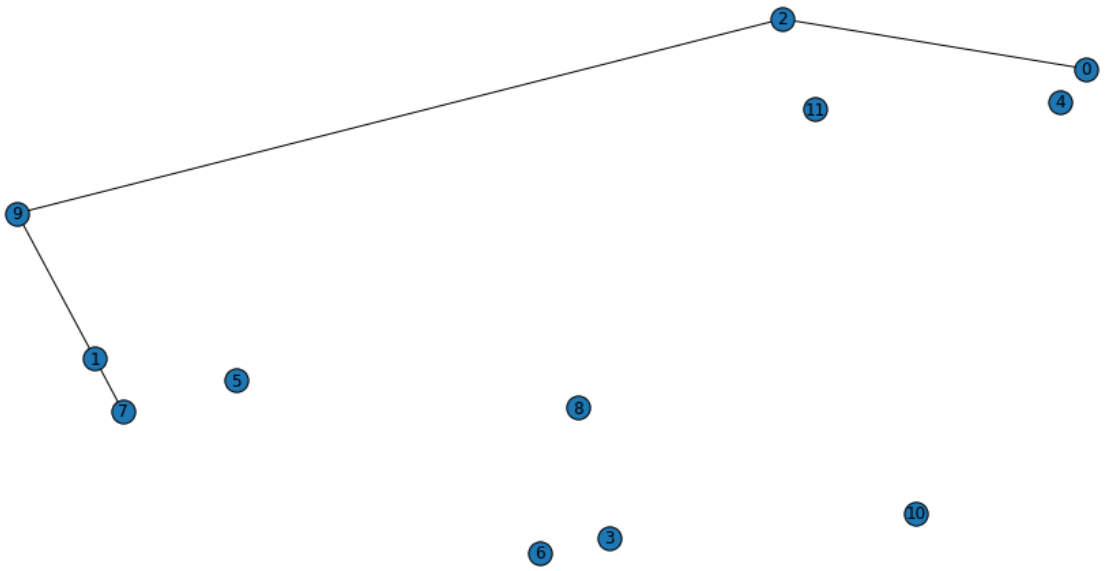
```
sum of distance m1= 7808
```



In [15]:
```
# to find a shortest path with hop count only
p = nx.shortest_path(G, 0, 7)
# to calculate the sum of edge attribute specified by a path
print ('path =', p)
print ('path length =', len(p))
print ('path length unweighted=', sum([G[p[i]][p[i+1]]['distance'] for i in
```

```
path = [0, 2, 9, 1, 7]
path length = 5
path length unweighted= 5464
```

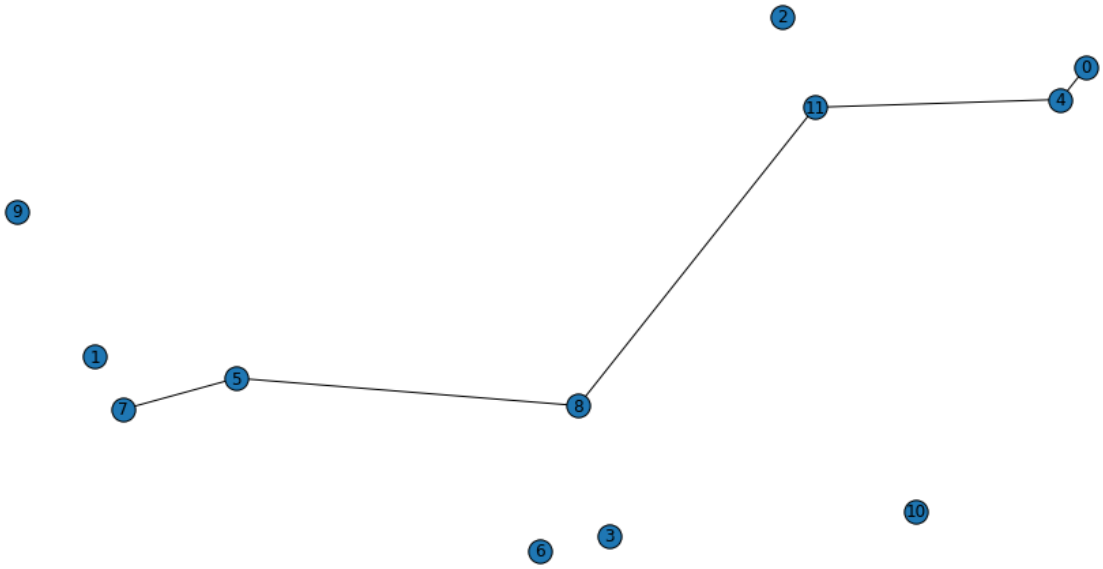
In [16]:
```
in_edges = [(p[i],p[i+1]) for i in range(len(p)-1)]
plt.figure(1,figsize=(12,6))
layout=nl.absolute_layout(G)
nl.draw_atlas(m1, pos=layout, edgelist=in_edges)
```



In [17]:
```
# to find a shortest path with an edge attribute
q = nx.shortest_path(G, 0, 7, 'distance')
print ('path =', q)
print ('path length =', len(q))
print ('path length weighted=', sum([G[q[i]][q[i+1]]['distance'] for i in ra

```

```
path = [0, 4, 11, 8, 5, 7]
path length = 6
path length weighted= 4896
```

In [18]:
```
in_edges = [(q[i],q[i+1]) for i in range(len(q)-1)]
plt.figure(1,figsize=(12,6))
nl.draw_atlas(m1, pos=layout, edgelist=in_edges)
```



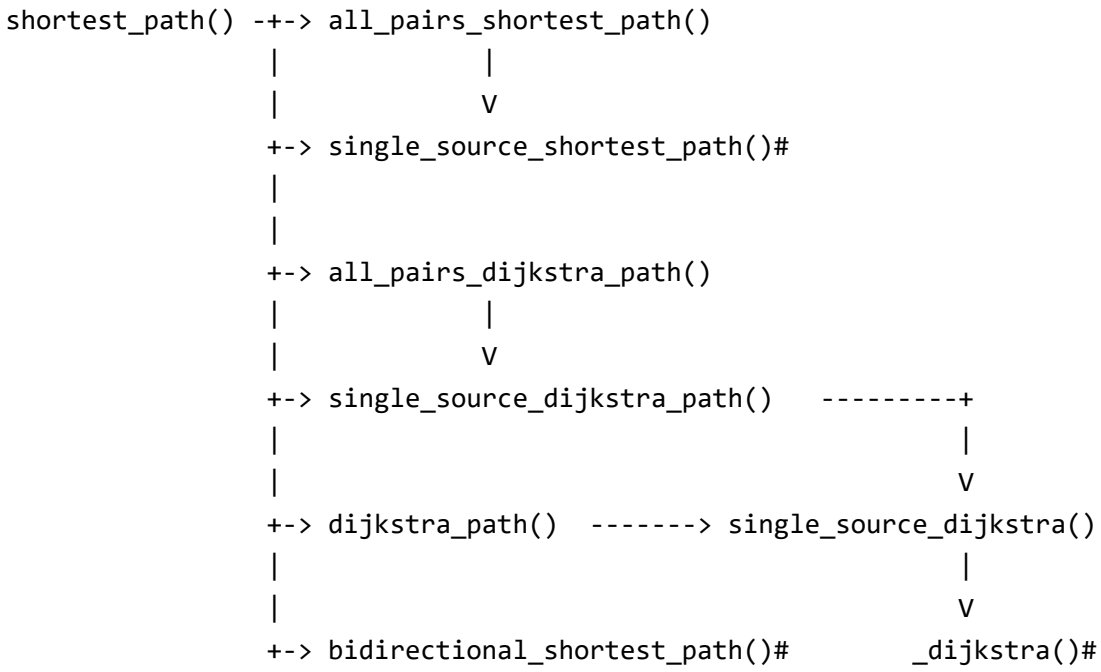
In [19]:
```
# to find all-pair shortest paths with hop count
list(nx.all_pairs_shortest_path(G))
```

```
(2,
 {2: [2],
  0: [2, 0],
  9: [2, 9],
  11: [2, 11],
  4: [2, 0, 4],
  1: [2, 9, 1],
  8: [2, 11, 8],
  10: [2, 0, 4, 10],
  5: [2, 9, 1, 5],
  7: [2, 9, 1, 7],
  3: [2, 11, 8, 3],
  6: [2, 11, 8, 6]}),
(3,
 {3: [3],
  8: [3, 8],
  10: [3, 10],
  6: [3, 6],
  5: [3, 8, 5],
  11: [3, 8, 11],
```

In [20]:
```
# to find all-pair shortest path on edge weight
D = nx.floyd_warshall_numpy(G, weight='distance')
print (D)
```

```
[[   0. 4760. 1264. 2904.  152. 4328. 3064. 4896. 2624. 4728. 1512. 1184.]
 [4760.    0. 3872. 2520. 4608.  432. 2232.  192. 2136.  544. 3912. 3576.]
 [1264. 3872.    0. 2120. 1328. 3440. 2176. 4008. 1736. 3464. 2688.  296.]
 [2904. 2520. 2120.    0. 2752. 2088.  320. 2360.  384. 3064. 1392. 1824.]
 [ 152. 4608. 1328. 2752.    0. 4176. 2912. 4744. 2472. 4792. 1360. 1032.]
 [4328.  432. 3440. 2088. 4176.    0. 2144.  568. 1704.  976. 3480. 3144.]
 [3064. 2232. 2176.  320. 2912. 2144.    0. 2040.  440. 2776. 1712. 1880.]
 [4896.  192. 4008. 2360. 4744.  568. 2040.    0. 2272.  736. 3752. 3712.]
 [2624. 2136. 1736.  384. 2472. 1704.  440. 2272.    0. 2680. 1776. 1440.]
 [4728.  544. 3464. 3064. 4792.  976. 2776.  736. 2680.    0. 4456. 3760.]
 [1512. 3912. 2688. 1392. 1360. 3480. 1712. 3752. 1776. 4456.    0. 2392.]
 [1184. 3576.  296. 1824. 1032. 3144. 1880. 3712. 1440. 3760. 2392.    0.]]
```

Networkx shortest path algorithms calling



In [21]:
```
a = [0,1,2,3]
b = a[0:2]
print (b)
```

```
[0, 1]
```

In [22]:
```
nx.shortest_path??
```

In [ ]:
```

```