Jan 2021 | © Guven

Dijkstra's (Djk) Shortest Path Algorithm

This notebook demonstrates,

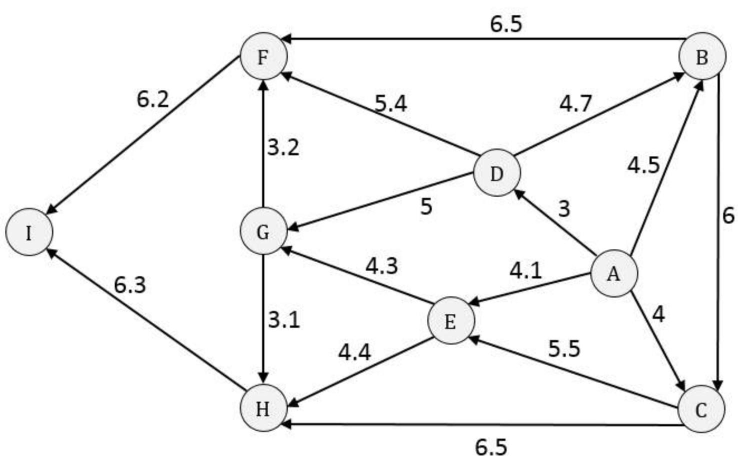
1. Djk algorithm
2. Djk running on example datasets

The algorithm:

$A^{(k)}(i,j) =$ Length of the shortest path from node i to j where the label of every intermediate node is $\leq k$

$A^{(k)}(i,j) =$ **min**$(A^{(k-1)}(i,j), A^{(k-1)}(i,k) + A^{(k-1)}(k,j))$

The following is an example graph with 9 nodes to be traversed by Djk Algorithm. The example uses `networkx` library to draw the graphs.



In [1]:
```python
def dijkstra(_edges, _origin, _destination):
    """finds the shortest path from the origin to destination node
       edges are vertex1 to vertex2 with weight
    """
    from collections import defaultdict
    from heapq import heappush, heappop

    e = defaultdict(list)
    for v1, v2, w in _edges:  # for each v1, fill v2 with weights
        e[v1].append((v2, w))
    # setup the heap - priority queue q
    q, seen, dist = [(0., _origin, ())], set(), {_origin: 0.}
    while q:
        # print(q)  # debug
        (totw, v1, path) = heappop(q)  # picks with the minimum w
        if v1 in seen:
            continue
        #
        seen.add(v1)  # mark as seen
        path += (v1, )
        if v1 == _destination:  # did we reach destination?
            return (totw, path)
        #
        for v2, w in e.get(v1, ()):
            if v2 in seen:
                continue
            if v2 not in dist or totw + w  < dist[v2]:
                dist[v2] = totw + w
                heappush(q, (totw + w, v2, path))
    #
    return float('inf')
```

In [2]:
```python
# The graph from above
E = [
    ('A', 'B', 4.5),
    ('A', 'C', 4.),
    ('A', 'D', 3.),
    ('A', 'E', 4.1),
    ('B', 'C', 6.),
    ('B', 'F', 6.5),
    ('C', 'E', 5.5),
    ('C', 'H', 6.5),
    ('D', 'F', 5.4),
    ('D', 'G', 5.),
    ('E', 'G', 4.3),
    ('E', 'H', 4.4),
    ('F', 'I', 6.2),
    ('G', 'F', 3.2),
    ('G', 'H', 3.1),
    ('H', 'I', 6.3),
]
```

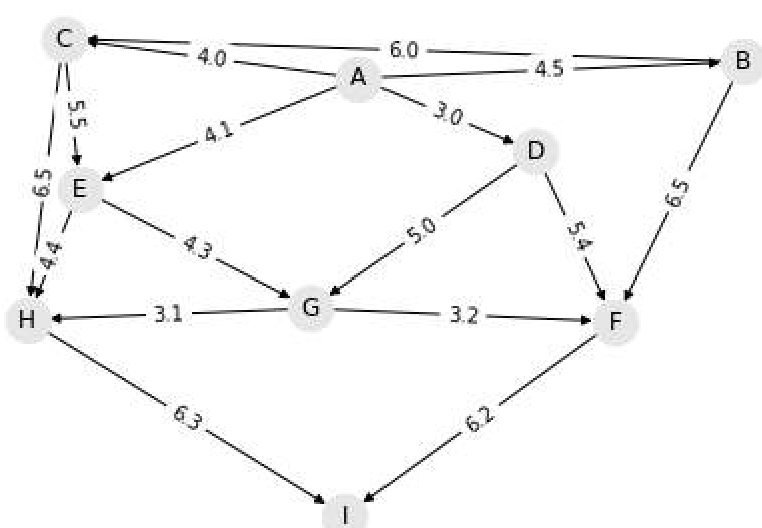
In [3]:
```
dijkstra(E, 'A', 'I')
```
Out[3]:
```
(14.600000000000001, ('A', 'D', 'F', 'I'))
```

In [4]:
```
dijkstra(E, 'C', 'F')
```
Out[4]:
```
(13.0, ('C', 'E', 'G', 'F'))
```

Now let's see `networkx` library to draw weighted directed graphs and apply Djk algorithm.

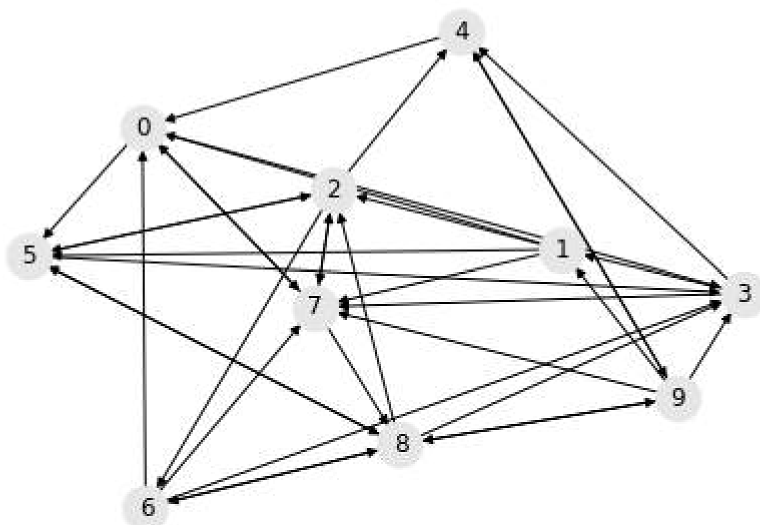
In [5]:
```
%matplotlib inline
import warnings
import matplotlib.cbook
warnings.filterwarnings("ignore",category=matplotlib.cbook.mplDeprecation)  # Future versions will fix this
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
print(f'networkx version {nx.__version__}')

g_di = nx.DiGraph()
g_di.add_weighted_edges_from(E)

def draw(_gdi):
    pos = nx.kamada_kawai_layout(_gdi)
    # pos = nx.planar_layout(_gdi)
    # pos = nx.spring_layout(_gdi)
    # pos = nx.spectral_layout(_gdi)
    # pos = nx.shell_layout(_gdi)
    nx.draw(_gdi, pos, node_size=500, node_color='0.9', with_labels=True)
    return pos

print('Original graph as above')
pos = draw(g_di)

nx.draw_networkx_edge_labels(g_di, pos, {(v1,v2):str(w) for v1, v2, w in E});
```

```
networkx version 2.4
Original graph as above
```



Let's create a random weighted directed graph.

```
In [6]: # Generate the digraph
def gen_digraph(nodes_n):
    g_di = nx.gnp_random_graph(nodes_n, 0.3, directed=True)  # args:
 number of nodes, ratio of number of edges
    E = [(v1,v2) for v1,v2,dt in nx.to_edgelist(g_di)]
    # populate weights from a uniform distribution between [0.5, 1.0]
    for v1, v2 in E:
        g_di[v1][v2]['weight'] = int(10*np.random.uniform()+5)/20.
    # not drawing edge labels
    draw(g_di)
    return g_di

g_di = gen_digraph(10)
```

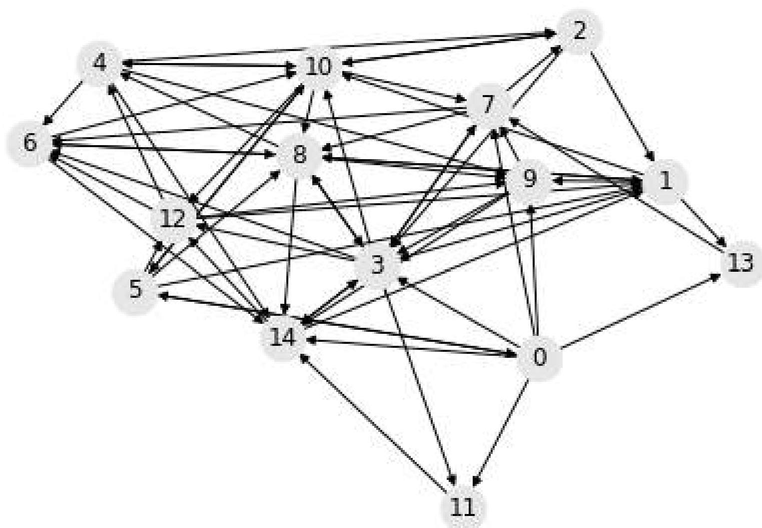



```
In [7]: # Find the path
def find_djk(g_di):
    # Get the edges from the random graph
    E = [(v1,v2,dt['weight']) for v1,v2,dt in nx.to_edgelist(g_di)]
    # Compute Djk from vertex 0 to the rest
    nodes_n = len(g_di)
    for v2 in range(1,nodes_n):
        djk = dijkstra(E, 0, v2)
        extra = ' - no path found' if djk == float('inf') else ''
        print(f'path from node {0} to node {v2}: {djk}'+extra)

find_djk(g_di)
```

```
path from node 0 to node 1: (0.95, (0, 3, 1))
path from node 0 to node 2: (0.7, (0, 7, 2))
path from node 0 to node 3: (0.65, (0, 3))
path from node 0 to node 4: (1.1, (0, 7, 2, 4))
path from node 0 to node 5: (0.4, (0, 5))
path from node 0 to node 6: (1.0499999999999998, (0, 7, 8, 6))
path from node 0 to node 7: (0.4, (0, 7))
path from node 0 to node 8: (0.7, (0, 7, 8))
path from node 0 to node 9: (1.25, (0, 7, 8, 9))
```

In [8]:
```
# Example
g_di = gen_digraph(15)
find_djk(g_di)
```

```
path from node 0 to node 1: (1.0, (0, 14, 1))
path from node 0 to node 2: (0.8999999999999999, (0, 7, 2))
path from node 0 to node 3: (0.35, (0, 3))
path from node 0 to node 4: (1.0499999999999998, (0, 3, 12, 4))
path from node 0 to node 5: (0.65, (0, 5))
path from node 0 to node 6: (0.9999999999999999, (0, 3, 8, 6))
path from node 0 to node 7: (0.6, (0, 7))
path from node 0 to node 8: (0.6499999999999999, (0, 3, 8))
path from node 0 to node 9: (0.7, (0, 9))
path from node 0 to node 10: (1.0499999999999998, (0, 3, 10))
path from node 0 to node 11: (0.4, (0, 11))
path from node 0 to node 12: (0.6499999999999999, (0, 3, 12))
path from node 0 to node 13: (0.3, (0, 13))
path from node 0 to node 14: (0.45, (0, 14))
```



Exercises

Try the above code for higher number of nodes and verify the algorithm actually works.