Complex Networks: Quiz #9

Due on Jan 16th, 2019

RUOPENG XU
18M38179

Problem 1

Make a program of Dijkstra's algorithm without built-in functions (dijkstra_path & dijkstra_path_length).

Answer 1

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import sys
import functools
import operator

G = nx.Graph()
G.add_nodes_from(range(0, 5))
G.add_weighted_edges_from([(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15), (2, 3,
                                                11), (2, 5, 2), (3, 4, 6), (4, 5, 9)])

plt.figure(figsize=(5, 5))
pos = nx.spring_layout(G)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edge_labels(G, pos, font_size=16, edge_labels={(u, v): d["weight"] for u, v
                                                , d in G.edges(data=True)})
nx.draw_networkx_labels(G, pos)
plt.axis('off')
plt.show()

# find the smallest one
dist_estimate = [sys.maxsize] * nx.number_of_nodes(G)
# real distance
dist_certainty = [0] * nx.number_of_nodes(G)
dist_estimate[1] = 0


print(dist_estimate)
print(dist_certainty)

#
# please fill in this part (Dijkstra's algorithm)
# calculte the distance from 1 -> 4

n = 0
distance = 0
count = 1
temp_nodes = 0

for visited in dist_certainty:
  if (visited <= 1 ):
# find the smallest estimate in G
    for nodes in G:
      if (dist_certainty[nodes] == 0):
          temp_min = dist_estimate[nodes]
          break
    for nodes in G:
      if (dist_certainty[nodes] == 0):
        if (dist_estimate[nodes] <= temp_min and dist_estimate[nodes] < 10000):
          temp_min = dist_estimate[nodes]
          temp_node = nodes
    current_distance = temp_min
    current_node = temp_node
```

Problem 1 continued on next page...

```
    print("In step",count,":")
    print("current_distance = ", current_distance)
    print("current_node = ", current_node)

  #     mark this distance as certain
    dist_certainty[current_node] = 1
  #     calculate the distance and compare
    for neighbor in G.neighbors(current_node):
      if (dist_certainty[neighbor] != 1):
        weight = list(dict.values(G.get_edge_data(current_node,neighbor)))
        current_weight = int(weight[0])
        distance = current_distance + current_weight
        if(distance < dist_estimate[neighbor]):
          dist_estimate[neighbor] = distance

    print("dist_estimate",dist_estimate)
    print("dist_certainty",dist_certainty)
    count = count + 1




#######
# print(dist_estimate)
# print(dist_certainty)
print("From built-in function:")
print(nx.dijkstra_path(G,1,4))
print(nx.dijkstra_path_length(G,1,4))
```

The result is:

```
[9223372036854775807, 0, 9223372036854775807, 9223372036854775807, 9223372036854775807,
                                                 9223372036854775807]
[0, 0, 0, 0, 0, 0]
In step 1 :
current_distance =  0
current_node =  1
dist_estimate [7, 0, 10, 15, 9223372036854775807, 9223372036854775807]
dist_certainty [0, 1, 0, 0, 0, 0]
In step 2 :
current_distance =  7
current_node =  0
dist_estimate [7, 0, 10, 15, 9223372036854775807, 21]
dist_certainty [1, 1, 0, 0, 0, 0]
In step 3 :
current_distance =  10
current_node =  2
dist_estimate [7, 0, 10, 15, 9223372036854775807, 12]
dist_certainty [1, 1, 1, 0, 0, 0]
In step 4 :
current_distance =  12
current_node =  5
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 0, 0, 1]
In step 5 :
current_distance =  15
current_node =  3
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 1, 0, 1]
In step 6 :
```

```
current_distance =  21
current_node =  4
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 1, 1, 1]
From built-in function:
[1, 2, 5, 4]
21
```

Problem 2

Show all the statuses of current estimates and their certainties while Dijkstra's algorithm is performed from vertex 0.

Answer 2

According to the algorithm above, when the Dijkstra is from vertex 0, change dist_estimate[0] to 0.
The result is:

```
In step 1 :
current_distance =  0
current_node =  0
dist_estimate [0, 7, 9, 9223372036854775807, 9223372036854775807, 14]
dist_certainty [1, 0, 0, 0, 0, 0]
In step 2 :
current_distance =  7
current_node =  1
dist_estimate [0, 7, 9, 22, 9223372036854775807, 14]
dist_certainty [1, 1, 0, 0, 0, 0]
In step 3 :
current_distance =  9
current_node =  2
dist_estimate [0, 7, 9, 20, 9223372036854775807, 11]
dist_certainty [1, 1, 1, 0, 0, 0]
In step 4 :
current_distance =  11
current_node =  5
dist_estimate [0, 7, 9, 20, 20, 11]
dist_certainty [1, 1, 1, 0, 0, 1]
In step 5 :
current_distance =  20
current_node =  4
dist_estimate [0, 7, 9, 20, 20, 11]
dist_certainty [1, 1, 1, 0, 1, 1]
In step 6 :
current_distance =  20
current_node =  3
dist_estimate [0, 7, 9, 20, 20, 11]
dist_certainty [1, 1, 1, 1, 1, 1]
From built-in function:
[0, 2, 5, 4]
20
```

Problem 3

Start from vertex 1 and show all the statuses & their certainties.

Answer 3

The result is same as the result in Problem 1:

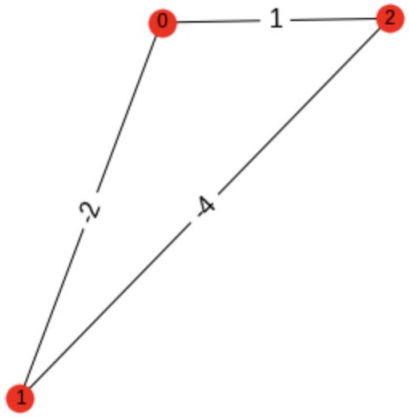
```
In step 1 :
current_distance =  0
current_node =  1
dist_estimate [7, 0, 10, 15, 9223372036854775807, 9223372036854775807]
dist_certainty [0, 1, 0, 0, 0, 0]
In step 2 :
current_distance =  7
current_node =  0
dist_estimate [7, 0, 10, 15, 9223372036854775807, 21]
dist_certainty [1, 1, 0, 0, 0, 0]
In step 3 :
current_distance =  10
current_node =  2
dist_estimate [7, 0, 10, 15, 9223372036854775807, 12]
dist_certainty [1, 1, 1, 0, 0, 0]
In step 4 :
current_distance =  12
current_node =  5
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 0, 0, 1]
In step 5 :
current_distance =  15
current_node =  3
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 1, 0, 1]
In step 6 :
current_distance =  21
current_node =  4
dist_estimate [7, 0, 10, 15, 21, 12]
dist_certainty [1, 1, 1, 1, 1, 1]
From built-in function:
[1, 2, 5, 4]
21
```

Problem 4

Explain the reasons why Dijkstra's algorithm does not work for negative weight edges.

Answer 4

Because when we think a vertex is visited and change its value in *dist_certainty* to 1, it will not be visited anymore. If a vertex is in visited, we will think we have already found the shortest path to this vertex. However, this doesn't work in ngetive path. For example, if a graph with negative edge is like this:



When we start from vertex 0, we think the shortest path to 1 is -2, but it is actually -3(from veertex 2 to vertex 1).