東京工業大学
Tokyo Institute of Technology

Complex Network

Quiz 9

WANG BIYUAN

18M38156

Department of Computer Science

Lecturer: Tsuyoshi Murata

January 12, 2019

Code

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

# Store edge weights in a list
graph = [
        [0, 7, 9, sys.maxsize, sys.maxsize, 14],
        [7, 0, 10, 15, sys.maxsize, sys.maxsize],
        [9, 10, 0, 11, sys.maxsize, 2],
        [sys.maxsize, 15, 11, 0, 6, sys.maxsize],
        [sys.maxsize, sys.maxsize, sys.maxsize, 6, 0, 9],
        [14, sys.maxsize, 2, sys.maxsize, 9, 0]]


dist_estimate = [sys.maxsize] * nx.number_of_nodes(G)
dist_certainty = [0] * nx.number_of_nodes(G)
dist_estimate[0] = 0 # First node to itself dist = 0

# Print out initialisation
print(dist_estimate)
print(dist_certainty)



print(dist_estimate)
print(dist_certainty)

current = 0 # Start from 1st node
dist_certainty[current] = 1 # First node is visited

# Initialise the distance estimation by looking at neighbours of starting
   node
for i in nx.nodes(G):
  dist_estimate[i] = graph[current][i]

print(dist_estimate)
print(dist_certainty)
prev=0
```

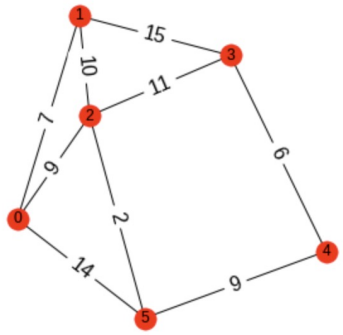
```
#-------------------------------------------#
# Loop throught every node and its neighbours
while functools.reduce(operator.mul, dist_certainty) == 0:
  mini = sys.maxsize
  for n in nx.nodes(G):
    # If the node has not been visited
    if dist_estimate[n] < mini and dist_certainty[n] != 1 and
       graph[prev][n] != sys.maxsize:
      mini = dist_estimate[n] # Update the nearest neighbour
      current = n # Search from the nearest neighbour next time
  dist_certainty[current] = 1 # Mark this nearest neighbour as visited
  prev = current


  for n in nx.nodes(G):
    # If shorter distance is found, update the distance
    if dist_estimate[n] > dist_estimate[current]+graph[current][n]:
      dist_estimate[n] = dist_estimate[current]+graph[current][n]
  print(dist_estimate)
  print(dist_certainty)
```

Results

(1) Make a program of Dijkstra's algorithm without builtin functions (dijkstra_path & dijkstra_path_length).
See the **Code** section above.

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

(3) Start from vertex 1 and show all the statuses their certainties.




```
[0, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
[0, 0, 0, 0, 0, 0]
[0, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
[0, 0, 0, 0, 0, 0]
[0, 7, 9, 9223372036854775807, 9223372036854775807, 14]
[1, 0, 0, 0, 0, 0]
[0, 7, 9, 22, 9223372036854775807, 14]
[1, 1, 0, 0, 0, 0]
[0, 7, 9, 20, 9223372036854775807, 11]
[1, 1, 1, 0, 0, 0]
[0, 7, 9, 20, 20, 11]
[1, 1, 1, 0, 0, 1]
[0, 7, 9, 20, 20, 11]
[1, 1, 1, 0, 1, 1]
[0, 7, 9, 20, 20, 11]
[1, 1, 1, 1, 1, 1]
```

(4) Explain the reasons why Dijkstra's algorithm does not work for negative weight edges.
Dijkstra's algorithm relies on the assumption that all the edge weights are positive and if a shortest path from the starting node is chosen, adding an edge with non-negative path will never make the path length shorter. Therefore, in the end of selection, the selected path will always be the shortest one. If there are negative edge weights, this path selection might not be true because the shortest path with negative weight might appear after a node which does not have the shortest path from last node.