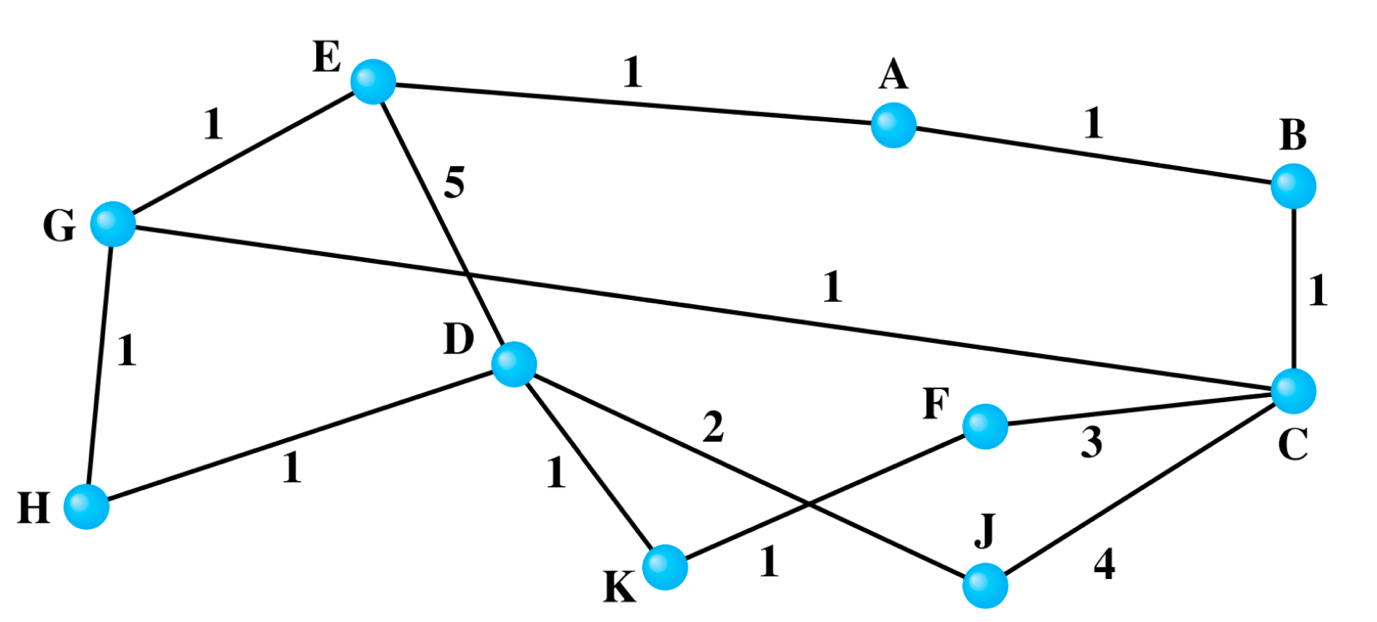
Assignment: 2

Subject: CS971 - Computer networks II

Date: 17th Nov. 2024

Name: P. Sivachandran

**Bellman-ford algorithm to find the shortest path**



According to above graph, the number (of nodes) Vertices found are 10 (A to K).

Representation of Route path:

1. Vertices A to K are mapped to indices 0 to 10
2. Below table prepared and assigned the values to Edges with weights:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | J | K |
| A |  |  |  |  |  |  |  |  |  |  |
| B | 1 |  | 1 |  |  |  |  |  |  |  |
| C |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |
| E | 1 |  |  | 5 |  |  | 1 |  |  |  |
| F |  |  | 3 |  |  |  |  |  |  | 1 |
| G |  |  | 1 |  |  |  |  |  |  |  |
| H |  |  |  | 1 |  |  | 1 |  |  |  |
| J |  |  | 4 | 2 |  |  |  |  |  |  |
| K |  |  |  | 1 |  |  |  |  |  |  |

1. A 🡪 B (1),
2. A 🡪 E(1),
3. C 🡪 B (1),
4. C 🡪 G (1),
5. C 🡪 F (3),
6. C 🡪 J (4),
7. D 🡪 E (5),
8. D 🡪 H (1),
9. D 🡪 K (1),
10. D 🡪 J (2),
11. G 🡪 E (1),
12. G 🡪 H (1),
13. K 🡪 F (1)

Bellman-ford algorithm implementation:

1. Initialized the source distance as zero, and remaining distances are set as infinity
2. Mapping of node label with indices
3. It relaxes all edges |V|-1 times to find the shortest path
4. The shortest path between two vertices can have at most (V-1) edges. Finding the simple path with more than (V-1) edges are difficult. Repeating the relaxation process (V-1) times ensures that all feasible route paths between source and any other destinations are covered.

Get user input and publish the output:

1. Get Source and destination nodes from terminal
2. Check the nodes entered are within the range or not
3. Calculate the shortest with the above iterative process
4. Publish the minimum possible route path calculated

I have prepared python script and executed in Anaconda Jupyter notebook environment.

Sharing below the execution result just for reference:

Enter the source node (A-K): a

Enter the destination node (A-K): k

The shortest path cost from A to K is: 5

(Please refer “*Bellman\_ford\_algorithm.html*” file to get more details)

Python code prepared just for reference:

(Please refer “*Bellman\_ford\_algorithm.ipynb*” to access the code and to find more details)

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices # Number of vertices

self.edges = [] # List to store edges (u, v, weight)

# Assigning an edge to the graph

def add\_edge(self, u, v, weight):

self.edges.append((u, v, weight))

# Bellman-Ford algorithm designed to find the shortest path between and source destination

def bellman\_ford(self, source):

# Step 1: Initialize distances

distances = [float('inf')] \* self.V

distances[source] = 0

# Step 2: Relax edges |V| - 1 times

for \_ in range(self.V - 1):

for u, v, weight in self.edges:

if distances[u] != float('inf') and distances[u] + weight < distances[v]:

distances[v] = distances[u] + weight

# Step 3: Check for negative weight cycles

for u, v, weight in self.edges:

if distances[u] != float('inf') and distances[u] + weight < distances[v]:

print("Graph contains a negative weight cycle.")

return None

return distances

g = Graph(11) # Total 10 vertices: A-K, mapped to indices

# Add edges with weights

edges = [

('A', 'B', 1), ('A', 'E', 1),

('C', 'B', 1), ('C', 'F', 3), ('C', 'J', 4), ('C', 'G', 1),

('D', 'E', 5), ('D', 'H', 1), ('D', 'J', 2), ('D', 'K', 1),

('G', 'E', 1), ('G', 'H', 1),

('K', 'F', 1),

]

# Map node labels to indices (A = 0, B = 1, ..., K = 10)

node\_map = {chr(65 + i): i for i in range(13)}

for u, v, weight in edges:

g.add\_edge(node\_map[u], node\_map[v], weight)

g.add\_edge(node\_map[v], node\_map[u], weight) # Assuming undirected graph

# Get source and destination from user

source\_node = input("Enter the source node (A-K): ").strip().upper()

destination\_node = input("Enter the destination node (A-K): ").strip().upper()

# Calculate shortest path

source\_index = node\_map[source\_node]

destination\_index = node\_map[destination\_node]

distances = g.bellman\_ford(source\_index)

if distances:

cost = distances[destination\_index]

if cost == float('inf'):

print(f"No path exists from {source\_node} to {destination\_node}.")

else:

print(f"The shortest path cost from {source\_node} to {destination\_node} is: {cost}")