CMPE252: Artificial Intelligence and Data Engineering

Homework 1: Implementing Dijkstra's algorithm to solve for minimum cost paths on a given graph.

Dijkstra's Algorithm:

- It is used to find a path with minimum cost between two points.
- It is a greedy algorithm
- Time complexity depends on the number of nodes(n) and edges(e), which is o(n*e)
- This algorithm was created and published by Dr. Edsger W. Dijkstra in 1959

Limitations of Dijkstra's Algorithm:

- There is no sense of direction towards the destination due to which it may take longer to compute results, this limitation is overcome by the A* search algorithm.
- The algorithm cannot accommodate negative weights.

Applications

- Maps and Navigation systems, Transportation and Logistics
- Computer Networking (to find the shortest path for data packets)
- Game Development (to find the shortest path between entities)

Implementation

1. **How to run the code.**
    a. In the zip file uploaded on Canvas is a Python script with the name hw1.py. Run this Python script to execute the code. Make sure the input and coords file are in the same directory as the script.
    b. After the script is executed it should generate an output.txt file with the desired output.

2. **How to make a video.**
    a. I captured and saved all the plot figures to a PNG file and then stitched them to make a video using ffmpeg.

3. **Code:**

```
import matplotlib.pyplot as plt

def dijkstra(graph, start, end):
    x=0
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    relaxed_nodes = [(0, start)]
```

```
    while len(relaxed_nodes):
# pop the first element from the relaxed nodes
        current_distance, current_node = relaxed_nodes.pop(0)
# get the neighbor of the current node
        for neighbor, weight in graph[current_node]:
# check if the current distance of the node from the start is greater than the distance
we have, break.
            if current_distance > distances[current_node]:
                break
# current distance is the distance from the start to the previous node and we are
adding the weight to get the exact distance to the neighbor node
            distance = current_distance + weight
# If the distance is less than the distance of the neighbor we have in the dict, update
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                relaxed_nodes.append([distance, neighbor])
# Plot the line between the current and neighbor node
            short_x1  = coords_x[int(current_node)-1]
            short_x2  = coords_x[int(neighbor)-1]
            short_y1  = coords_y[int(current_node)-1]
            short_y2  = coords_y[int(neighbor)-1]
            plt.plot([short_x1, short_x2], [short_y1, short_y2], c="lightgrey", zorder=1)
            x+=1
            relaxed_nodes.sort()
# save graphs to image
            # plt.savefig('hw1_{}.png'.format(x))

    path = []
    current_node = end
# Traverse and check the smallest path from end to start
    while current_node != start:
        path.insert(0, current_node)
        for neighbor, weight in graph[current_node]:
            if distances[current_node] == distances[neighbor] + weight:
                current_node = neighbor
                break

    path.insert(0, start)
# return optimal path, distance dict, and total number of iterations
    return path, distances, x

# Reading coordinates and input file
f_coords = open("coords.txt", "r")
f_input = open("input.txt", "r")

# Contents of coords file
coords = f_coords.readlines()
```

```
coords = [i.replace('\n',"") for i in coords]
coords_x = [int(float(i.split(" ")[0])) for i in coords]
coords_y = [int(float(i.split(" ")[1])) for i in coords]

# contents of input file
total_nodes = int(f_input.readline())
start = int(f_input.readline())
end = int(f_input.readline())
dir_wei = f_input.readlines()
dir_wei = [i.replace('\n',"") for i in dir_wei]

# Creating the graph using a dictionary. Stores node as key, and stores neighbor and weight as values
graph = {i: [] for i in range(1, total_nodes + 1)}
for i in range(len(dir_wei)):
    node, neigh_node, weight = map(float, dir_wei[i].split())
    graph[node].append((neigh_node, weight))

# ploting the graph
plt.xticks(range(0,22,2))
plt.yticks(range(0,22,2))
i=1
for x,y in zip(coords_x,coords_y):
    label = "{}".format(i)
    plt.annotate(label, (x,y), textcoords="offset points", xytext=(-5,5), ha='center', size=9, zorder=3)
    i+=1
    plt.scatter(x, y, c ="blue", zorder=2)

# -1 as the start number is not index
start_x = coords_x[int(start)-1]
start_y = coords_y[int(start)-1]
end_x = coords_x[int(end)-1]
end_y = coords_y[int(end)-1]
# marking start and end with green and red color
plt.scatter(start_x,start_y, c ="green", zorder=4)
plt.scatter(end_x,end_y, c ="red", zorder=4)

# fetch the shortest path between start and end using Dijkstra's algorithm.
shortest_path, distances, x = dijkstra(graph, start, end)
# returns the shortest between the start and the end nodes, and returns distances between all nodes from the start
for i in range(len(shortest_path)-1):
    p1 = int(shortest_path[i]) - 1
    p2 = int(shortest_path[i+1]) - 1
    short_x1  = coords_x[p1]
    short_x2  = coords_x[p2]
    short_y1  = coords_y[p1]
```

```
    short_y2  = coords_y[p2]
    plt.plot([short_x1, short_x2], [short_y1, short_y2], c="orange", zorder=5)
    # plt.savefig('hw1_{}.png'.format(x))
    x+=1
plt.show()
```

 # x is the total number of iterations program took to execute the algorithm

```
with open('output.txt', 'w') as f:
    shortestpath_str = ""
    distances_str = ""
    for i in range(len(shortest_path)):
        shortestpath_str = shortestpath_str + " {}".format(int(shortest_path[i]))
        if(distances[shortest_path[i]] == 0):
            distances_str = distances_str + "0"
        else:
            distances_str = distances_str + " {}".format("{:.4f}".format(distances[shortest_path[i]]))

    shortestpath_str = shortestpath_str.strip()
    distances_str = distances_str.strip()
    f.write(shortestpath_str + "\n" + distances_str)

print(shortestpath_str)
print(distances_str)
```