Homework 9: NP-Completeness and Heuristic Algorithms

1. **NP-Completeness:** Consider the Traveling Salesperson (TSP) problem that was covered in the exploration: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? (Given a graph G with V vertices and E edges, determine if the graph has a TSP solution with a cost of at most *k*.)

   Prove that the above stated problem is NP-Complete

   Step 1: Show the Problem is in NP

   To be in NP, we need to show that for any given solution (a route in this context), we can verify whether it satisfies the conditions (visits each city exactly once, returns to the starting point, and has a total cost of at most *k*) in polynomial time.

   Given a candidate route, we can check if each city is visited exactly once and the route returns to the start by simply following the route and marking each visited city. This process takes linear time relative to the number of cities. Then, we sum the costs of each leg of the journey to ensure it does not exceed *k*, which also takes linear time. Since both checks are polynomial in the size of the input, the problem is in NP.

   Step 2: Show the Problem is NP-Hard

   To prove NP-Hardness, we need to reduce a known NP-Hard problem to our problem in polynomial time. For the TSP problem, we can use the Hamiltonian Cycle problem, which is known to be NP-Hard. The Hamiltonian Cycle problem asks whether there is a cycle that visits each vertex exactly once and returns to the origin. The goal is to show that if we could solve our TSP decision problem efficiently (in polynomial time), then we could also solve the Hamiltonian Cycle problem efficiently.

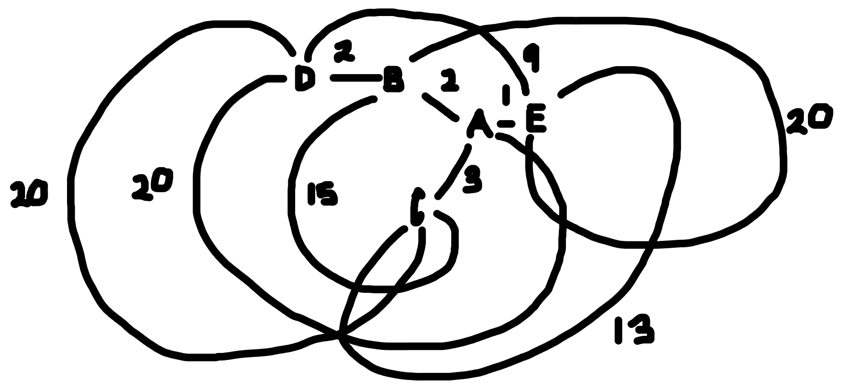
   Take any instance of the Hamiltonian Cycle problem. This instance is a graph G' where we want to find a cycle that visits each vertex exactly once. We transform G' into an instance of our TSP problem by assigning a distance of 1 to each edge in G' and a distance greater than k (making it effectively impossible to choose) to any edge not in G'. Set $k=V$ for our TSP instance, where V is the number of vertices. If we can solve the TSP problem for this graph and find a route with cost at most k, it means we have found a Hamiltonian cycle in G'. Conversely, if we can find a Hamiltonian cycle in G', it corresponds to a valid TSP route in our constructed graph.

Conclusion

Since the TSP decision problem can be reduced from a known NP-Hard problem (showing it's at least as hard as the hardest problems in NP) and since any solution to the problem can be verified in polynomial time (placing it in NP), the TSP decision problem is NP-complete.

2. **Implement Heuristic Algorithm:**
   a. Below matrix represents the distance of 5 cities from each other. Represent it in the form of a graph

	A	B	C	D	E
A	0	2	3	20	1
B	2	0	15	2	20
C	3	15	0	20	13
D	20	2	20	0	9
E	1	20	13	9	0



   b. Apply Nearest-neighbor heuristic to this matrix and find the approximate solution for this matrix if it were for TSP problem.

   a -> e -> d -> b -> c -> a with a total distance of 30

   c. What is the approximation ratio of your approximate solution?

Approximation ration = (Cost of Approximation solution)/(Cost of Optimal Solution)
= 30/ (a -> b -> d -> e -> c -> a with a total distance of 29)

= 1.0345

d. Implement Travelling Salesman Problem using the nearest-neighbor heuristic. Done

Input: The input Graph is provided in the form of a 2-D matrix (adjacency matrix). Consider the first node as the starting point.

Sample input:

```
G = [
    [0, 2, 3, 20, 1],
    [2, 0, 15, 2, 20],
    [3, 15, 0, 20, 13],
    [20, 2, 20, 0, 9],
    [1, 20, 13, 9, 0],
    ]
```

Output: A list of indices indicating the path taken. You must return the sequence of nodes, the path taken starting from node 0. In this example, G is 5x5, indicating there are 5 nodes in this graph: 0-4. You will always begin with node 0, and your path should include every node exactly once, and only go between nodes with a nonzero edge between them. You path will end at the starting node.

Sample output (For above graph G):
[0, 4, 3, 1, 2, 0]

Note: Not all graphs are fully connected: some rows in G may have more than one 0. These indicate absence of an edge.

Name your function solve_tsp(G). Name your file TSP.py.