CSE221 Algorithms
Assignment 3: Graph Theory

There are four problems in this assignment. Each one carries the same weight.
Total Marks: 5
Deadline: 25th August 2023 (Friday), 11:59 PM (No extension)

Assume that you are given a function called $dijkstra(G, a)$ which takes a graph G, and a source vertex a as input, then finds the shortest path (distances, $dist$ and parents, par) from a to every other vertex in the graph. Use this function to solve problems #1 and #2.
Present your solution idea with a pseudocode, accompanying the necessary explanation.

Problem #1: Optimum meeting place

You, and your friend Benji, live in a city that has 50 areas numbered from 1 to 50. Some of the areas are connected by bi-directional roads of various lengths. Your house is in 1 and Benji's is in 50. Both of you want to meet. Now propose an algorithm to choose the meeting point that minimizes the total travel time. In other words, if the meeting area you chose is X, it has the minimum $dist(1, X)\ +\ dist(50, X)$ among all possible area choices.

Expected complexity: $O(E\ *\ log_2(V))$

Problem #2: Meet at KFC!!

Continuing on the scenario of Problem #1, Benji now has asked you to visit his house in area 50. He likes KFC, so you are thinking about getting a bucket of fried chicken for him. There are KFC outlets available in K different areas of the city. You have to make a stop at one of them. Now propose an algorithm to choose the area from the available K options that minimize your total travel time. In other words, if the area you chose is X, it has the minimum $dist(1, X)\ +\ dist(X, 50)$ among all possible choices.

Expected complexity: $O(E\ *\ log_2(V))$

Note:

1. *Remember that for SPT problems,* $minimum\ dist(A, B)\ might\ not\ be\ equal\ to\ minimum\ \ dist(B,\ A)$
2. *If you run Dijkstra's algorithm from* K *different sources, the complexity becomes* $O(K * E * log_2(V))$*, which exceeds the expected complexity for this problem.*

Problem #3: Dijkstra's algorithm in 'Negative' Graph

Well, we lied. Dijkstra could find the shortest distance for "***<u>some</u>***" of the graphs that contain negative edge weights, **<u>but not for "all"</u>**. That's why Bellman-Ford Algorithm comes into play.

Now, Draw a directed graph as described below.

No. of vertices: 7, No. of edges: 9
List of vertices, V: $\{1, 2, 3, 4, 5, 6, 7\}$
List of edges, E: $\{(1, 2, 8),\ (2, 3, -\ 8),\ (1, 3, 1)\ (3, 4, 4),\ (4, 5, -\ 4),\ (3, 5, 1),\ (5, 6, 2),\ (6, 7, -\ 2),\ (5, 7, 1)\}$

The edges are given in (u, v, w) format which means there is an outgoing edge from u to v with weight w.

a. Show a simulation of Dijkstra's algorithm to find the *shortest path costs* from vertex 1 to all others. You can use the following pseudocode as a reference.
   [Dijkstra's algorithm inserts all vertices in a priority queue/min-heap at the beginning and then repeatedly extracts the vertex with minimum distance. It assumes, whenever a vertex is extracted from the queue, its shortest path cost from the source is finalized.]

```
DIJKSTRA$(G, s)$
for each $v \in V[G]$
    do $d[v] \leftarrow \infty$
       $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN$(Q)$
       $S \leftarrow S \cup \{u\}$
       for each vertex $v \in Adj[u]$
           do if $d[v] > d[u] + w(u, v)$
                 then $d[v] \leftarrow d[u] + w(u, v)$
                      $\pi[v] \leftarrow u$
```

b. Explain why the algorithm could find the correct shortest path costs for some of the vertices.
c. Imagine you have a modified version of Dijkstra's algorithm with the following changes:
   i. At the beginning, it inserts only the source vertex in the priority-queue.
   ii. Whenever the distance of a vertex is updated, it inserts that vertex into the priority-queue.
   iii. It does not assume that extracting a vertex from the queue means that its shortest path cost is finalized.

   Will this modified Dijkstra algorithm be able to find the correct shortest path costs in the graph given above? Explain with reasoning or a simulation.
d. Analyze the time complexity of this modified algorithm.

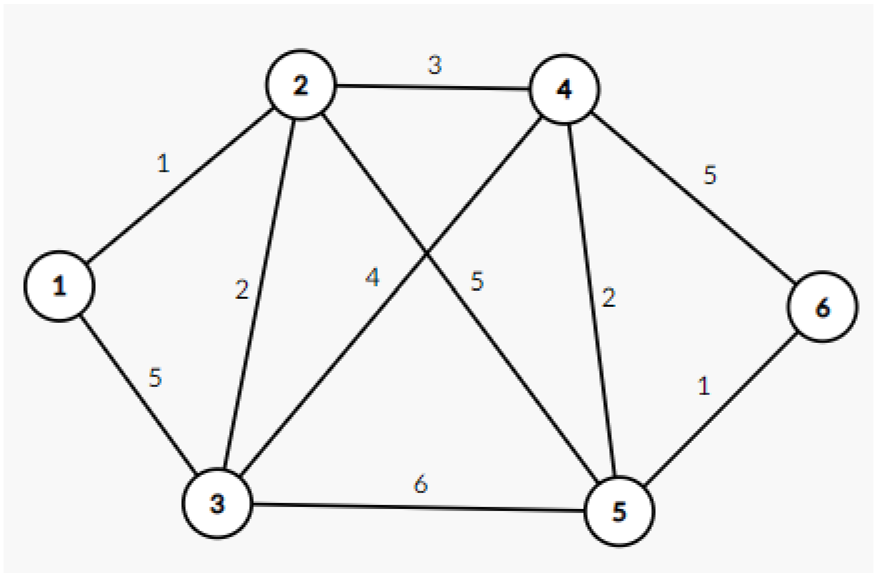
Problem #4: A divide-and-conquer MST algorithm (!?!) ¯_(ツ)_/¯

Your friend Mriaslun loves the divide and conquer strategy! He recently invented a divide-and-conquer algorithm to find the Minimum Spanning Tree of a given undirected weighted graph. The algorithm is quite simple, just having 4 steps in it:

1. Divide the set of vertices into two equal halves (one side can have an extra, like we do in merge sort). Assume that this divides the graph in three parts:

   a. G_{left} which contains all the vertices in the left half and the edges between them
   b. G_{right} containing the vertices in the right half, and the corresponding edges
   c. $G_{remains}$ containing the edges where one vertex is from the left half, and another one is from the right half
2. Use a recursive approach (repeating steps 1 to 4) to find the MST of G_{left}, let's call it MST_{left}
3. Do the same to find MST_{right}
4. Now you only need to select one edge from $G_{remains}$ to connect MST_{left} to MST_{right}. So you pick the one with the lowest weight value and you have the MST of the original graph!

Let's see an example.



In G_{left}, we have three vertices $\{1, 2, 3\}$ and three edges $\{(1, 2),\ (1, 3),\ (2, 3)\}$
In G_{right}, we have vertices $\{4, 5, 6\}$ and edges $\{(4, 5),\ (4, 6),\ (5, 6)\}$
And $G_{remains}$ is defined by the edges $\{(2, 4),\ (2, 5),\ (3, 4),\ (3, 5)\}$

If we can find MST_{left} and MST_{right} separately, then we can connect them with the minimum weight edge from $G_{remains}$, which is $(2, 4)$. And we can use the same divide and conquer approach recursively to know that MST_{left} consists of the edges $\{(1, 2),\ (1, 3)\}$ and MST_{right} contains the edges $\{(4, 5),\ (5, 6)\}$.
So, the MST of the original graph is $\{(1, 2),\ (1, 3),\ (2, 4),\ (4, 5),\ (5, 6)\}$ which is the correct answer!

Now your job is to prove that this algorithm will not always give a correct answer. Provide the necessary reasons to do that. You can also create an input graph and show that the algorithm fails or provides a wrong answer for this input.

END!