Consider the below attempt to render a complete graph of six vertices readable.

The weight on each edge is given by the taxicab distance between the points. To be clear, the taxicab distance between two point \((a, b)\) and \((c, d)\) is \(|c - a| + |d - b|\).

**Problem 1.** Draw a minimum spanning tree for the graph (ideally, show your work step by step).

**Solution 1.**
Problem 2. Perform a DFS of your minimum spanning tree. The visit function is called when a vertex is first discovered. It adds the newly discovered vertex to the end of an initially empty list (i.e. it records the order vertices are discovered).

What is this list when your DFS is done?

Solution 2. (1,1) (2,1) (2,2) (0,0) (0,3) (-1,2)

Problem 3. Calculate the following:

a) The total weight of your minimum spanning tree.

b) Take the list your DFS generated and add a duplicate of the first item to the end of the list. Calculate the total weight of the resulting cycle.

Solution 3.

a) 9

b) 14

Problem 4. The triangle inequality is a property of metrics (a.k.a. distance functions). It states that the distance between any two points A and B is at most the distance between A to any other point C first and then to B. In other words, taking a detour only makes a distance larger. The taxicab distance is a metric, and it obeys the triangle inequality.

A tour of a graph is a cycle that visits every vertex. The traveling salesman problem requires you to find a minimum weight tour of a complete graph (one where every edge is present). In Problem 3, you created a tour that is (probably) not the minimum weight tour. However, it does have a special property when your graph obeys the triangle inequality.

a) What type of tree are you left with when you delete an edge from a tour?

b) Explain why the minimum weight tour’s total weight must be at least the total weight of your minimum spanning tree.
c) Using the triangle inequality, explain why your tour could not have had a total weight greater than twice the minimum spanning tree’s total weight.

d) Finally, explain why your tour’s total weight could not have been more than twice the minimum weight tour’s total weight.

**Solution 4.**

a) A spanning tree.

b) A tour is one edge larger than a spanning tree, so it necessarily must have greater weight than a minimum weight spanning tree.

c) The tour was generated via a DFS traversal of the minimum weight spanning tree. If we followed each edge forward and backward along with the DFS’s traversal of the graph, then the total weight of this circuit would be exactly twice the minimum spanning tree’s weight. Our tour took the direct route between discovered vertices, however, and the triangle inequality guarantees that this is faster, i.e. the total weight is at least no worse (but is probably reduced).

d) We know that the minimum weight tour’s total weight is at least the minimum spanning tree’s total weight. We also know that our tour’s weight is no more than twice the minimum spanning tree’s weight. Thus our tour’s total weight is at most twice the minimum spanning tree’s total weight, which is at most twice the minimum weight tour’s total weight.

**Problem 5.** Give an efficient algorithm to find a tour of any complete graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{Q}_{\geq 0}$ satisfying

$$\forall r, s, t \in V, w(r,s) + w(s,t) \leq w(r,t)$$

whose total weight is no greater than twice the minimum weight tour’s total weight.

Your algorithm will be a polynomial time 2-approximation of the traveling salesman problem for the special case of graphs obeying the triangle inequality.

Fun fact: the unrestricted traveling salesman problem provably has no polynomial time approximation algorithm! It is a **hard** problem.

**Solution 5.**
Algorithm 1: TSP2Approx($G = (V, E, w)$)

**Input:** A complete graph $G$ with weighting $w$ satisfying the triangle inequality

**Output:** A tour of $G$ whose total weight is at most twice the minimum tour’s total weight

// Use Prim’s or Kruskal’s to get an MST
Let $T$ a minimum spanning tree of $G$

// The DFS visit function adds newly discovered vertices to $L$
Perform a DFS of $T$
Let $L$ be the order vertices were discovered in $T$
Add a copy of $L[0]$ to the end of $L$

// The cycle generated is our approximate tour
return $L$