Solutions to Problem Set No. 6

1. (a) Let D be the sum of the vertex degrees of T (or of any graph). Then $D = 2m$, where m is the number of edges, since each edge is counted in exactly two vertex degrees. Thus the sum of vertex degrees is even, which means there must be an even number of vertices of odd degree; otherwise the sum of vertex degrees would be odd.

   Thus S, the set of all odd-degree vertices in T, is of even size. Since G is complete, all pairs of vertices in S are connected by an edge, and S has a perfect matching (in general, many perfect matchings).

   (b) For this and part (d), we shall use the following lemma:

   Lemma 1 *Let $v_1, v_2, \ldots, v_{k+1}$ be any path with $k \geq 0$. Then $\sum_{i=1}^{k} w(v_i, v_{i+1}) \geq w(v_1, v_{k+1})$.*

   Proof. By induction on k. For $k = 0$ the result is trivial, and for $k = 1$ the result is just the triangle inequality. Let $k \geq 2$, and suppose the result is true for $k - 1$. Then

$$\begin{aligned}\sum_{i=1}^{k} w(v_i, v_{i+1}) &\geq w(v_1, v_k) + w(v_k, v_{k+1}) \text{ by the induction hypothesis} \\ &\geq w(v_1, v_{k+1}) \text{ by the triangle inequality}\end{aligned}$$

   Thus the result is true for k. Hence it is true for all k by induction. ■

   To handle part (b), let C be a cycle that contains all the vertices such that *weight* $(C) =$ OPT. Let C' be the cycle whose vertices are the vertices of S in the order they first occur around C, staring from an arbitrary vertex. For each pair of vertices s_i, s_{i+1} along C', $w(s_i, s_{i+1}) \leq \sum_{j=1}^{k} (v_j, v_{j+1})$ by Lemma 1, where $s_i = v_1, v_2, \ldots, v_{k+1} = s_{i+1}$ is the part of C from the first occurence of s_i to the first occurrence of s_{i+1}. Summing these inequalities for $i = 1, \ldots, |S|$, we find that *weight* $C') \leq$ *weight* $(C) =$ OPT. The edges of C' partition into two perfect matchings of S, whose edges alternate around C'. At least one these matchings has weight at most OPT/2. Since M is a minimum-cost perfect matching of S, *weight*$(M) \leq$ OPT/2 also. Furthermore *weight* $(T) \leq$ *weight*$(C) \leq$ OPT, since T is a minimum-weight spanning tree, C minus any edge is a spanning tree, and all edges have non-negative weight by assumption. Thus *weight*$(E) =$ *weight*$(M) +$ *weight* $(T) \leq$ OPT/2 + OPT = 1.5 OPT.

   (c) Step 1 takes $O(n^2)$ time using for example the Prim-Dijkstra-Jarnik minimum spanning tree algorithm. Step 2 takes $O(n)$ time; we merely construct an adjacency list representation of

T and count vertex degrees. Step 3 takes $O(n^3)$ time by assumption. Step 4 takes time proportional to the number of edges in $T \cup S$, namely $O(n)$, using the following algorithm. Begin at any vertex and walk along a (not-necessarily-simple) path by traversing edges of $T \cup S$ deleting them, until reaching a vertex with no undeleted edges. This vertex must be the starting vertex, because all vertices dare even degree, and any vertex other than the starting vertex that is entered by an edge must have another edge along which to exit. This traversal creates a cycle C. Repeat the following step until no edges are left. Choose a vertex v along C with degree greater than zero. Traverse and delete edges until returning to C, forming a cycle C'. Augment C by splicing in C'. (The new cycle follows C to v, then goes around C', then continues along C.) This process must eventually use all edges of $T \cup S$, because $T \cup S$ is connected; if some edge were missed, say edge (v, w), there would be an entire path missed joining v to C, and the first edge from C along this path would be available for traversal. Using an adjacency list representation, finding an Euler tour in this way takes $O(n)$ time. The total time for all steps is $O(n^3)$.

(d) Walk along E marking the first time each vertex occurs and making a list of the vertices $v_1, v_2, \ldots, v_n$ in the order they first occur. Let $E' = v_1, v_2, \ldots, v_n, v_1$. As in (b), by Lemma 1, $\mathit{weight}(E') \leq \mathit{weight}(E)$. Computing E' takes $O(n)$ time (less than the $O(n^2)$ bound suggested in the problem statement).

2. The problem is in NP, because if we are given a subgraph, we can check in linear time that it is biconnected and has weight at most k. Hence if the answer is "yes", we can verify in polynomial time that the answer is "yes" given a suitable polynomial-size hint.

   To show that the problem is NP-hard, we reduce the Hamiltonian cycle problem to it. Let G be a graph in which we wish to find a Hamiltonian cycle. Give weight 1 to every edge in G. We claim that G has a Hamiltonian cycle if and only if it has a biconnected spanning subgraph of total edge weight at most n, where n is the number of vertices of G. First, if H is a Hamiltonian cycle, then H is biconnected and spanning, and has total edge weight exactly n. Second, any biconnected spanning subgraph has at least two edges incident to every vertex. If it has weight n, it must have exactly two edges incident to every vertex, which means it is a union of simple cycles. Since it is biconnected, it is connected, which means that it is exactly one cycle, a Hamiltonian cycle.