1. For any integer k, define the k-bit binary graph Q_k by making 2^k vertices, one for each string of 0's and 1's of length k, and connecting two vertices by an edge whenever their strings differ from each other only in one position.

   (a) Find a Hamiltonian cycle in Q_3. (You may report your answer by giving the sequence of binary strings in the cycle.)

   Solution: 000–001–011–010–110–100–101–111–000

   (b) For which values of k does Q_k have an Euler tour, starting and ending at the same point?

   Solution: If k is even, each vertex of Q_k has even degree, so Q_k has an Euler tour. If k is odd, no vertex of Q_k has even degree, so Q_k does not have an Euler tour.

   (c) For which values of k does Q_k have an Euler path that starts and ends at two different vertices?

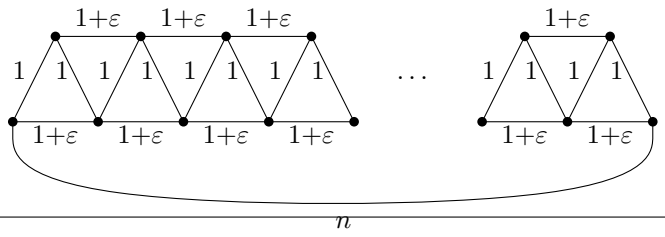
   Solution: To have an Euler path, two vertices must have odd degree and all others must have even degree. Q_k satisfies this property only when $k = 1$.

2. 163 only: Find a weighted undirected graph for which the MST-doubling heuristic can generate a tour that is longer than optimal by a factor of at least 5/3. Show both the optimal tour and the MST-doubling tour for your example. (You may use the simplest version of MST-doubling, without shortcutting repeated vertices.)

   Solution: In a cycle of length 1000 in which all edges have a weight of 1, the MST-doubling heuristic gives a tour with length 1998 (twice the length of a path through all the vertices), whereas the optimal solution has length 1000 (the length of the cycle). The approximation ratio is therefore 1998/1000 which is greater than 5/3.

   263 only: Find a weighted undirected graph for which Christofides' heuristic can generate a tour that is longer than optimal by a factor of at least 4/3. Show both the optimal tour and the Christofides tour for your example.

   Solution: Consider a graph consisting of $2n + 1$ vertices that looks like this:



where $\varepsilon = 1/(2n-1)$. Christofides' heuristic will output the tour consisting of the edges with weight 1 and n, which has weight $3n$. The optimal tour is the one consisting of edges of weight $1+\epsilon$ with two additional edges of weight 1, for a total weight of $2n+2$. Therefore, as $n \to \infty$, the approximation ratio converges to $3n/(2n+2) \to 3/2$. In particular, if $n \geq 8$, the approximation ratio is at least $4/3$.

3. Suppose that we know how to solve problem X in time $O(4^n)$ on inputs with n vertices, but then Professor Bjorklund discovers an improved algorithm that takes time $O(2^n)$. Also suppose that the constant factors in the two O-notations are the same and that both algorithms are limited by computation time rather than by memory or I/O complexity.

   (a) What is the effect of Professor Bjorklund's improvement on the size of the largest problem that can be solved in at most one hour of computer time (on some particular computer)?

   Solution: Let n_1 be the largest problem size that can be solved in time T (say an hour) using the original algorithm and let n_2 be the size of the largest problem that can be solved in the same time using the improved algorithm. Then
   $$c \cdot 4^{n_1} = T = c \cdot 2^{n_2}.$$
   So $n_2 = 2n_1$. In other words, using the improved algorithm, we can solve a problem twice as large.

   (b) If you ran Professor Bjorklund's algorithm on two computers, one of which is twice as fast as the other, what would be the effect of using the faster computer on the size of the largest problem that can be solved in at most one hour of computer time?

   Solution: Let n_1 be the size of the largest problem that can be solved in an hour on the original computer and n_2 the size of the largest problem that can be solved in an hour on the faster computer. On the faster computer, the same problem can be solved in half the time. Therefore,
   $$c \cdot 2^{n_1} = T = c \cdot \frac{2^{n_2}}{2}.$$
   So $n_2 = n_1 + 1$. In other words, by doubling the speed of the computer, we can solve a problem that is 1 unit larger.

4. 163 only: Recall that in class we went through a dynamic programming algorithm to compute $L(S, v)$, the length of the shortest path that starts at vertex 0, ends at vertex v, and covers all the vertices in set S, using the equation
   $$L(S, v) = \min_{w \in S} L(S - v, w) + d(w, v).$$

Here $S - v$ denotes the set formed from S by removing v, and $d(w, v)$ is the shortest path distance. Recall also that the length of the optimal traveling salesman tour is

$$\min_v L(V, v) + d(v, 0)$$

where V is the set of all vertices in the graph. Give pseudocode that takes as input the already-computed array L and uses these equations to backtrack through the array and find the optimal tour itself.

Solution:
input: the graph G, its edge weights $d(v, w)$, and the table $L(S, v)$ filled out
output: a list of the vertices in the optimal TSP order.

```
tour = []
S = V
v = 0
min_D = infinity
repeat (|V| - 1) times:
    for w, D in [w, L(S, w) + d(w, v) for w in S if w is v's neighbor]:
        if D < min_D:
            min_w, min_D = w, D
    tour.append(min_w)
    S.remove(min_w)
    v = min_w
tour.append(0)
return tour
```

Note: an alternative to an infinite value is an upper bound on the length of the tour which can be taken to be n times the length of the heaviest edges in the graph.

263 only: If the n vertices of a graph are assigned numbers from 0 to $n-1$, then we can define the length of any edge to be the difference between the numbers of its endpoints, and define the total length of the numbering to be the sum of all the edge lengths. For instance, if a five-vertex and five-edge graph is numbered so that there are edges 0–1, 0–3, 1–2, 2–3, and 3–4, then the total length is $1 + 3 + 1 + 1 + 1 = 7$. Give a dynamic programming algorithm to find the minimum possible total length of any numbering of a given graph. Your algorithm should take time within a polynomial factor of 2^n.

Solution: If all the vertices in a subset $S \subseteq V$ are assigned numbers from 0 to $|S| - 1$, define the *cut length* of this partial numbering to the sum over all edges with one endpoint in S and the other in $\overline{S}$ of the difference between the number of the

endpoint in S and the value $|S|$, and define the *half length* of this numbering to be the total length of the numbering on S plus its cut length.

Let $L(S)$ denote the minimum possible half length of any numbering of the vertices in the subset S from 0 to $|S| - 1$. Then

$$L(\emptyset) = 0, \quad L(S) = \min_{v \in S} L(S \setminus \{v\}) + |E \cap (S \times \overline{S})|.$$

Each value $L(S)$ can be calculated in time $O(n + m)$ from previous values. So the table can be filled out in time $O((n + m)2^n)$. The optimal value is that of $L(V)$.